Algebra for Infinite Forests with an Application to the Temporal Logic EF*

Mikołaj Bojańczyk and Tomasz Idziaszek

University of Warsaw, Poland
{bojan,idziaszek}@mimuw.edu.pl

Abstract. We define an extension of forest algebra for ω-forests. We show how the standard algebraic notions (free object, syntactic algebra, morphisms, etc.) extend to the infinite case. To prove its usefulness, we use the framework to get an effective characterization of the ω-forest languages that are definable in the temporal logic that uses the operator EF (exists finally).

1 Introduction

The goal of this paper is to explore an algebraic approach to infinite trees. We have decided to take a two-pronged approach:

- Develop a concept of forest algebra for infinite trees, extending to infinite trees the forest algebra defined in [8].
- Use the algebra to get an effective characterization for some logic (that is, an algorithm that decides which regular languages can be defined in the logic).

A good effective characterization benefits the algebra. Effective characterizations are usually difficult problems, and require insight into to the structure of the underlying algebra. We expected that as a byproduct of an effective characterization, we would discover what are the important ingredients of the algebra.

A good algebra benefits effective characterizations. A good algebra makes proofs easier and statements more elegant. We expected that an effective characterization would be a good test for the quality of an algebraic approach. In the previously studied cases of (infinite and finite) words and finite trees, some of the best work on algebra was devoted to effective characterizations.

We hope the reader will find that these expectations have been fulfilled.

Why the logic EF? What tree logic should we try to characterize? Since we are only beginning to explore the algebra for infinite trees, it is a good idea to start with some logic that is very well understood for finite trees. This is why for our case study we chose the temporal logic EF. For finite trees, this was one of the first nontrivial tree logic to get an effective characterization, for binary trees in [7], and for unranked trees [8]. Moreover, when stated in algebraic terms – as

* Work partially funded by the Polish government grant no. N206 008 32/0810.

M. Bravetti and G. Zavattaro (Eds.): CONCUR 2009, LNCS 5710, pp. 131–145, 2009.

in [8] – this characterization is simple: there are two identities $h + g = g + h$ and $vh = vh + h$ (these will be explained later in the paper).

We were also curious how some properties of the logic EF would extend from finite trees to infinite trees. For instance, for finite trees, a language can be defined in the logic EF if and only if it is closed under EF-bisimulation (a notion of bisimulation that uses the descendant relation instead of the child relation). What about infinite trees? (We prove that this is not the case.) Another example: for finite trees, a key proof technique is induction on the size of a tree. What about infinite trees? (Our solution is to use only regular trees, and do the induction on the number of distinct subtrees.)

Our approach to developing an algebra. Suppose you want to develop a new kind of algebra. The algebra should be given by a certain number of operations and a set of axioms that these operations should satisfy. For instance, in the case of finite words, there is one operation, concatenation, and one axiom, associativity (such a structure, of course, is called a semigroup). Given a finite alphabet A, the set of all nonempty words A^+ is simply the free semigroup. Regular languages are those that are recognized by morphisms from the free semigroup into a finite semigroup.

This approach was used in [8] to define forest algebra, an algebraic framework for finite unranked trees. The idea was to develop operations and axioms such that the free object would contain all trees. One idea was to have a two-sorted algebra, where one sort described forests (sequences of unranked trees), and the other sort described contexts (forests with a hole). Forest algebra has been successfully applied in a number of effective characterizations, including fragments of first-order logic [6,5] and temporal logics [3], see [4] for a survey. An important open problem is to find an effective characterization of first-order logic with the descendant relation (first-order with the child relation was effectively characterized in [1]).

When developing an algebraic framework for infinite words (and even worse, infinite trees), we run into a problem. For an alphabet A with at least two letters, the set A^ω of all infinite words is uncountable. On the other hand, the free object will be countable, as long as the number of operations is countable. There are two solutions to this problem: either have an uncountable number of operations, or have a free object that is different from A^ω. The first approach is called an ω-semigroup (see the monograph [10]). The second approach is called a Wilke semigroup [11]. Like in forest algebra, a Wilke semigroup is a two-sorted object. The axioms and operations are designed so that the free object will have all finite words on the first sort, and all ultimately periodic words on the second sort. Why is it possible to ignore words that are not ultimately periodic? The reason is that any ω-regular language $L \subseteq A^\omega$ is uniquely defined by the ultimately periodic words that it contains. In this sense, a morphism from the free Wilke semigroup into a finite Wilke semigroup contains all the information about an ω-regular language.

Our approach to infinite trees combines forest algebra and Wilke semigroups. As in forest algebra, we have two sorts: forests and contexts. Both the forests

and the contexts can contain infinite paths, although the hole in a context has to be at finite depth (since there is no such thing as a hole at infinite depth). As in a Wilke semigroup, the free object does not contain all forests or all contexts, but only contain the regular ones (a forest or context is regular if it has a finite number of nonisomorphic subtrees, which is the tree equivalent of ultimately periodic words).

Organization of the paper. In Section 2 we present the basic concepts, such as trees, forests, contexts and automata. The algebra is defined in Section 3. In Section 4, we define the logic EF and present a characterization, which says that a language can be defined in EF if and only if: (a) it is invariant under EF-bisimulation; and (b) its syntactic algebra satisfies a certain identity. There are three tasks: prove the characterization, decide condition (a), and decide condition (b). Each of these tasks is nontrivial and requires original ideas. Due to a lack of space, the algorithms for deciding (a) and (b) are relegated to the appendix. We end the paper with a conclusions section. Apart from the usual ideas for future work, we try to outline the limitations of our approach.

2 Preliminaries

2.1 Trees and Contexts

This paper mainly studies forests, which are ordered sequences of trees. Forests and trees can have both infinite and finite maximal paths, but each node must have finitely many siblings. Formally, a forest over finite alphabet A is a partial map $t : \mathbb{N}^+ \to A$ whose domain (the set of nodes) is closed under nonempty prefixes, and such that for each $x \in \mathbb{N}^*$, the set $\{i : x \cdot i \in \mathrm{dom}(t)\}$ is a finite prefix of $\mathbb{N}$. We use letters s, t for forests. An empty forest is denoted by 0. We use the terms root (there may be several, these are nodes in $\mathbb{N}$), leaf, child, ancestor and descendant in the standard way. A *tree* is an forest with one root. If t is a forest and x is a node, we write $t|_x$ for the subtree of t rooted in x, defined as $t|_x(y) = t(xy)$.

A *context* is a forest with a hole. Formally, a context over an alphabet A is a forest over the alphabet $A \cup \{\Box\}$ where the label $\Box$, called the hole, occurs exactly once and in a leaf. We use letters p, q to denote contexts. A context is called *guarded* if the hole is not a root.

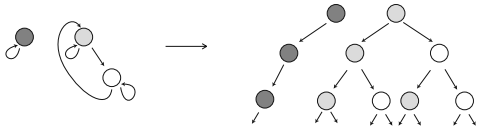
Operations on forests and contexts. We define two types of operations on forests and contexts: a *(horizontal) concatenation operation*, written additively, and a *(vertical) composition operation*, written multiplicatively. In general, neither concatenation nor composition is commutative.

What objects can be concatenated? We can concatenate two forests s, t, the result is a forest $s + t$ that has as many roots as s and t combined. (We do not assume that concatenation is commutative, so $s + t$ need not be the same

as $t + s$.) Since contexts can be interpreted as forests with a hole label, we can also concatenate a forest s with a context p, with the being result is a context $s + p$. There is also the symmetric concatenation $p + s$. In other words, we can concatenate anything with anything, as long as it is not two contexts (otherwise we could get two holes).

What objects can be composed? We can compose a context p with a forest t, the result is a forest pt obtained by replacing the hole of p with the forest t. For instance, if p is a context with a in the root and a hole below, written as $a\Box$, and t is a forest (and also tree) with a single node labeled b, written as b, then $p(t+t) = a(b + b)$ is a tree with the root labeled a and two children with label b. We can also compose a context p with another context q, the resulting context pq satisfies $pqt = p(qt)$ for all forests t. We cannot compose a forest t with a context p, or another forest s, since t has no hole.

Regular forests and recursion schemes. A forest is called *regular* if it has finitely many distinct subtrees. Regular forests are important for two reasons: a) they can be represented in a finite way; and b) regular languages are uniquely determined by the regular forests they contain. One way of representing a regular forest is as a forest with backward loops, as in the picture below.



The formal definition of forests with backward loops is presented below, under the name of recursion schemes. Let $Z = Z_H \cup Z_V$ be a set of *label variables*. The set Z_H represents forest-sorted variables and the set Z_V represents context-sorted variables. Let Y be a set of *recursion variables*. *Recursion terms* are built in the following way:

1. 0 is a recursion term.
2. If $\tau_1, \ldots, \tau_n$ are recursion terms, then so is $\tau_1 + \cdots + \tau_n$.
3. Every forest-sorted label variable $z \in Z_H$ is a recursion term.
4. If $z \in Z_V$ is a context-sorted label variable and τ is a recursion term, then $z\tau$ is a recursion term.
5. Every recursion variable $y \in Y$ is a recursion term.
6. If $y \in Y$ a recursion variable and τ is a recursion term where y is guarded, then $\nu y.\tau$ is a recursion term. We say a recursion variable y is guarded in a recursion term τ if there is no decomposition $\tau = \tau_1 + y + \tau_2$.

A *recursion scheme* is a recursion term without free recursion variables, i.e. a recursion term where every recursion variable y occurs in as a subterm of a term $\nu y.\tau$. We denote recursion schemes and terms using letters τ, σ. We also assume that each recursion variable is bound at most once, to avoid discussing scope of variables.

Let η be a function (called a *valuation*) that maps forest-sorted label variables to forests and context-sorted label variables to guarded contexts. We define $\mathrm{unfold}_\tau[\eta]$ to be the (possibly infinite) forest obtained by replacing the label z with their values $\eta(z)$, and unfolding the loops involving the recursion variables. The formal definition is in the appendix. If the recursion scheme uses only m forest-sorted variables $z_1, \ldots, z_m$ and n context-sorted variables $z'_1, \ldots, z'_n$ (we call this an (m, n)-ary recursion scheme), then only the values of η on these variables are relevant. In such a case, we will interpret unfold_τ as a function from tuples of m forests and n contexts into forests.

For instance, suppose that z' is a context-sorted variable and z_1, z_2 are forest sorted variables. For $\tau = z'z_1 + z_2$, $\mathrm{unfold}_\tau(t_1, t_2, p) = pt_1 + t_2$, and if $\tau = \nu y.z'z'y$ then $\mathrm{unfold}_\tau(p)$ is the infinite forest $ppp\cdots$. Note that the notation $\mathrm{unfold}_\tau(t_1, t_2, p)$ uses an implicit order on the label variables.

Note 1. Why only allow guarded contexts as inputs for the unfolding? For the same reason as restricting the binding $\nu y.\tau$ to terms τ where y is guarded. Take, for instance a recursion scheme $\tau = \nu y.zy$. What should the result of $\mathrm{unfold}_\tau(a + \Box)$ be, for the unguarded context $a + \Box$? We could say that this is the a forest $a + a + \cdots$ that is infinite to the right. But then, in a similar way, we could generate the forest $\cdots + a + a$ that is infinite to the left. What would happen after concatenating the two forests? In order to avoid such problems, we only allow contexts where the hole is not in the root. Another solution would be to suppose that the order on siblings is an arbitrary linear ordering.

Lemma 2.1. *Regular forests are exactly the unfoldings of recursion schemes.*

2.2 Automata for Unranked Infinite Trees

A (nondeterministic parity) *forest automaton* over an alphabet A is given by a set of states Q equipped with a monoid structure, a transition relation $\delta \subseteq Q \times A \times Q$, an initial state $q_I \in Q$ and a parity condition $\Omega : Q \to \{0, \ldots, k\}$. We use additive notation $+$ for the monoid operation in Q, and we write 0 for the neutral element.

A *run* of this automaton over a forest t is a labeling $\rho : \mathrm{dom}(t) \to Q$ of forest nodes with states such that for any node x with children $x_1, \ldots, x_n$,

$$(\rho(x_1) + \rho(x_2) + \cdots + \rho(x_n), t(x), \rho(x)) \in \delta.$$

Note that if x is a leaf, then the above implies $(0, t(x), \rho(x)) \in \delta$.

A run is *accepting* if for every infinite path $\pi \subseteq \mathrm{dom}(t)$, the highest value of $\Omega(q)$ is even among those states q which appear infinitely often on the path π. The *value* of a run over a forest t is the obtained by adding, using $+$, all the states assigned to roots of the forest. A forest is *accepted* if it has an accepting run whose value is the initial state q_I. The set of trees accepted by an automaton is called the *regular* language *recognized* by the automaton.

Theorem 2.2
Languages recognized by forest automata are closed under boolean operations and projection. Every nonempty forest automaton accepts some regular forest.

Two consequences of the above theorem are that forest automata have the same expressive power as the logic MSO, and that a regular forest language is determined by the regular forests it contains. We can also transfer complexity results from automata over binary trees to forest automata.

3 Forest Algebra

In this section we define ω-forest algebra, and prove some of its basic properties.

Usually, when defining an algebraic object, such as a semigroup, monoid, Wilke semigroup, or forest algebra, one gives the operations and axioms. Once these operations and axioms are given, a set of generators (the alphabet) determines the free object (e.g. nonempty words in the case of semigroups). Here, we use the reverse approach. We begin by defining the free object. Then, we choose the operations and axioms of ω-forest algebra so that we get this free object.

Let A be an alphabet. Below, we define a structure $A^{\triangle}$. The idea is that $A^{\triangle}$ will turn out to be the free ω-forest algebra generated by A. The structure $A^{\triangle}$ is two-sorted, i.e. it has two domains H_A and V_A. The first domain H_A, called the *forest sort*, consists of all (regular) forests over the alphabet A. The second domain V_A, called the *context sort* consists of all (regular) guarded contexts over the alphabet A. From now on, when writing forest, tree or context, we mean a regular forest, regular tree, or regular guarded context, respectively.

What are the operations? There are eight *basic operations*, as well as infinitely many *recursion operations*.

Basic operations. There is a constant $0 \in H_A$ and seven binary operations

$$\begin{aligned}
s, t \in H_A &\quad\mapsto\quad s + t \in H_A \\
p, q \in V_A &\quad\mapsto\quad pq \in V_A \\
p \in V_A, s \in H_A &\quad\mapsto\quad ps \in H_A \\
p \in V_A, s \in H_A &\quad\mapsto\quad p + s \in V_A \\
p \in V_A, s \in H_A &\quad\mapsto\quad s + p \in V_A \\
p \in V_A, s \in H_A &\quad\mapsto\quad p(\Box + s) \in V_A \\
p \in V_A, s \in H_A &\quad\mapsto\quad p(s + \Box) \in V_A
\end{aligned}$$

If we had all contexts, instead of only guarded contexts, in the context sort, we could replace the last four operations by two unary operations $s \mapsto s + \Box$ and $s \mapsto \Box + s$. However, having unguarded contexts would cause problems for the recursion operations.

Recursion operations. For each (m, n)-ary recursion scheme τ, there is an $(m + n)$-ary operation

$$\begin{aligned}
&s_1, \ldots, s_m \in H_A \\
&p_1, \ldots, p_n \in V_A
\end{aligned} \quad\mapsto\quad \mathrm{unfold}_\tau(s_1, \ldots, s_m, p_1, \ldots, p_n) \in H_A.$$

We use p^∞ as syntactic sugar for $\mathrm{unfold}_{\nu y.zy}(p)$.

Generators. The operations are designed so that every forest and context over alphabet A can be generated from single letters in A. It is important however, that the alphabet, when treated as a generator for $A^{\triangle}$, is considered as part of the context sort.

More formally, for an alphabet A, we write $A\Box$ for the set of contexts $\{a\Box : a \in A\}$. By Lemma 2.1, the domain H_A is generated by $A\Box \subseteq V_A$. It is also easy to see that every context in V_A is also generated by this set, it suffices to construct the path to the hole in the context and generate all remaining subtrees. Therefore $A^{\triangle}$ is generated by $A\Box$.

Definition of ω-forest algebra. We are now ready to define what an ω-forest algebra is. It is a two sorted structure (H, V). The operations are the same as in each structure $A^{\triangle}$: eight basic operations and infinitely many recursion operations. The axioms that are required in an ω-forest algebra are described in the appendix. These axioms are designed so as to make the following theorem true. Homomorphisms (also called morphisms here) are defined in the appendix.

Theorem 3.1
Let A be a finite alphabet, and let (H, V) be an ω-forest algebra. Any function $f : A\Box \to V$ uniquely extends to a morphism $\alpha : A^{\triangle} \to (H, V)$.

Recognizing Languages with an ω-Forest Algebra

A set L of A-forests is said to be *recognized* by a surjective morphism $\alpha : A^{\triangle} \to (H, V)$ onto a finite ω-forest algebra (H, V)f membership $t \in L$ depends only on the value $\alpha(t)$. The morphism α, and also the target ω-forest algebra (H, V), are said to recognize L.

The next important concept is that of a syntactic ω-forest algebra of a forest language L. This is going to be an ω-forest algebra that recognizes the language, and one that is optimal (in the sense of 3.3) among those that do.

Let L be a forest language over an alphabet A. We associate with a forest language L two equivalence relations (à la Myhill-Nerode) on the free ω-forest algebra $A^{\triangle}$. The first equivalence, on contexts is defined as follows. Two contexts p, q are called *L-equivalent* if for every forest-valued term ϕ over the signature of ω-forest algebra, any valuation $\eta : X \to A^{\triangle}$ of the variables in the term, and any context-sorted variable x, either both or none of the forests

$$\phi[\eta[x \mapsto p]] \qquad \text{and} \qquad \phi[\eta[x \mapsto q]]$$

belong to L. Roughly speaking, the context p can be replaced by the context q inside any regular forest, without affecting membership in the language L. The notion of L-equivalence for forest s, t is defined similarly. We write $\sim_L$ for L-equivalence. Using universal algebra, it is not difficult to show:

Lemma 3.2. *L-equivalence, as a two-sorted equivalence relation, is a congruence with respect to the operations of the ω-forest algebra $A^{\triangle}$.*

The *syntactic algebra* of a forest language L is the quotient (H_L, V_L) of $A^\triangle$ 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 algebra is an ω-forest algebra thanks to Lemma 3.2. The *syntactic morphism* α_L assigns to every element of $A^\triangle$ its equivalence class under L-equivalence. The following proposition shows that the syntactic algebra has the properties required of a syntactic object.

Proposition 3.3. A forest language L over A is recognized by its syntactic morphism α_L. Moreover, any morphism $\alpha : A^\triangle \to (H, V)$ that recognizes L can be extended by a morphism $\beta : (H, V) \to (H_L, V_L)$ so that $\beta \circ \alpha = \alpha_L$.

Note that in general the syntactic ω-forest algebra may be infinite. However, Proposition 3.3 shows that if a forest language is recognized by some finite forest algebra, then its syntactic forest algebra must also be finite. In the appendix we show that all regular forest languages have finite ω-forest algebras, which can furthermore be effectively calculated (since there are infinitely many operations, we also specify what it means to calculate an ω-forest algebra).

We use different notation depending on whether we are dealing with the free algebra, or with a (usually finite) algebra recognizing a language. In the first case, we use letters s, t for elements of H and p, q, r for elements of V, since these are "real" forests and contexts. In the second case, we will use letters f, g, h for elements of H and u, v, w for elements of V, and call them forest types and context types respectively.

4 EF for Infinite Trees

In this section we present the main technical contribution of this paper, which is an effective characterization of the forest and tree languages that can be defined in the temporal logic EF. We begin by defining the logic EF. Fix an alphabet A.

- Every letter $a \in A$ is an EF formula, which is true in trees with root label a.
- EF formulas admit boolean operations, including negation.
- If φ is an EF formula, then $\mathsf{EF}\varphi$ is an EF formula, which is true in trees that have a proper subtree where φ is true. EF stands for Exists Finally.

A number of operators can be introduced as syntactic sugar:

$$\mathsf{AG}\varphi = \neg\mathsf{EF}\neg\varphi, \qquad \mathsf{AG}^*\varphi = \varphi \wedge \mathsf{AG}\varphi, \qquad \mathsf{EF}^*\varphi = \varphi \vee \mathsf{EF}\varphi.$$

Since we deal with forest languages in this paper, we will also want to define forest languages using the logic. It is clear which forests should satisfy the formula EF^*a (some node in the forest has label a). It is less clear which forests should satisfy $\mathsf{EF}a$ (only non-root nodes are considered?), and even less clear which forests should satisfy a (which root node should have label a?). We will only use boolean combinations of formulas of the first kind to describe forests. That is, a *forest EF formula* is a boolean combination of formulas of the form $\mathsf{EF}^*\varphi$.

For finite forests, the logic EF was characterized in [8]. The result was:

Theorem 4.1
Let L be a language of finite forests. There is a forest formula of EF that is equivalent, over finite forests, to L if and only if the syntactic forest algebra (H_L, V_L) of L satisfies the identities

$$vh = vh + h \qquad \text{for } h \in H_L, v \in V_L, \tag{1}$$

$$h + g = g + h \qquad \text{for } g, h \in H_L. \tag{2}$$

A corollary to the above theorem is that, for finite forests, definability in EF is equivalent to invariance under EF-bisimulation. This is because two finite trees that are EF-bisimilar can be rewritten into each other using the identities (1) and (2).

Our goal in this paper is to test ω-forest algebra by extending Theorem 4.1 to infinite forests. There are nontrivial properties of infinite forests that can be expressed in EF. Consider for example the forest formula $\mathsf{AG}^*(a \wedge \mathsf{EF}a)$. This formula says that all nodes have label a and at least one descendant. Any forest that satisfies this formula is bisimilar to the tree $(a\Box)^\infty$. In this paper, we will be interested in a weaker form of bisimilarity (where more forests are bisimilar), which we will call EF-bisimilarity, and define below.

EF game. We define a game, called the *EF game*, which is used to test the similarity of two forests under forest formulas of EF. The name EF comes from the logic, but also, conveniently, is an abbreviation for Ehrenfeucht-Fraïssé.

Let t_0, t_1 be forests. The EF game over t_0 and t_1 is played by two players: Spoiler and Duplicator. The game proceeds in rounds. At the beginning of each round, the state in the game is a pair of forests, (t_0, t_1). A round is played as follows. First Spoiler selects one of the forests t_i $(i = 0, 1)$ and its subtree s_i, possibly a root subtree. Then Duplicator selects a subtree s_{1-i} in the other tree t_{1-i}. If the root labels a_0, a_1 of s_0, s_1 are different, then Spoiler wins the whole game. Otherwise the round is finished, and a new round is played with the state updated to (r_0, r_1) where the forest r_i is obtained from the tree s_i by removing the root node, i.e $s_i = a_i r_i$.

This game is designed to reflect the structure of EF formulas, so the following theorem, which is proved by induction on m, should not come as a surprise.

Fact 4.2. Spoiler wins the m-round EF game on forests t_0 and t_1 if and only if there is a forest EF formula of EF-nesting depth m that is true in t_0 but not t_1.

We will also be interested in the case when the game is played for an infinite number of rounds. If Duplicator can survive for infinitely many rounds in the game on t_0 and t_1, then we say that the forests t_0 and t_1 are *EF-bisimilar*. A forest language L is called *invariant under EF-bisimulation* if it is impossible to find two forests $t_0 \in L$ and $t_1 \notin L$ that are EF-bisimilar.

Thanks to Fact 4.2, we know that any language defined by a forest formula of EF is invariant under EF-bisimulation. Unlike for finite forests, the converse

does not hold[1]. Consider, for instance the language "all finite forests". This language is invariant under EF-bisimulation, but it cannot be defined using a forest formula of EF, as will follow from our main result, stated below.

Theorem 4.3 (Main Theorem: Characterization of EF)
A forest language L can be defined by a forest formula of EF if and only if

- *It is invariant under EF-bisimulation;*
- *Its syntactic ω-forest algebra (H_L, V_L) satisfies the identity*

$$v^\omega h = (v + v^\omega h)^\infty \qquad \text{for all } h \in H_L, v \in V_L. \tag{3}$$

Recall that we have defined the ∞ exponent as syntactic sugar for unfolding the context infinitely many times. What about the ω exponent in the identity? In the syntactic algebra, V_L is a finite monoid (this is proved in the appendix). As in any finite monoid, there is a number $n \in \mathbb{N}$ such that v^n is an idempotent, for any $v \in V_L$. This number is written as ω. Note that identity (3) is not satisfied by the language "all finite forests". It suffices to take v to be the image, in the syntactic algebra, of the forest $a\Box$. In this case, the left side of (3) corresponds to a finite forest, and the right side corresponds to an infinite forest.

In the appendixwe show that the two conditions (invariance and the identity) are necessary for definability in EF. The proof is fairly standard. The more interesting part is that the two conditions are sufficient, this is done in following section.

Corollary 4.4 (of Theorem 4.3). The following problem is decidable. The input is a forest automaton. The question is: can the language recognized by the forest automaton be defined by a forest formula of EF.

Proof
In appendix we show how to decide property (3) (actually, we show more: how to decide any identity). In appendix we show how to decide invariance under EF-bisimulation. An ExpTime lower bound holds already for deterministic automata over finite trees, see [7], which can be easily adapted to this setting. Our algorithm for (3) is in ExpTime, but we do not know if invariance under EF-bisimulation can also be tested in ExpTime (our algorithm is doubly exponential). □

Note 2. Theorem 4.3 speaks of forest languages defined by forest EF formulas. What about tree languages, defined by normal EF formulas? The latter can be reduced to the former. The reason, not difficult to prove, is that a tree language L can be defined by a normal formula of EF if and only if for each label $a \in A$, the forest language $\{t : at \in L\}$ is definable by a forest formula of EF. A tree version of Corollary 4.4 can also be readily obtained.

[1] In the appendix, we show a weaker form of the converse. Namely, for any fixed regular forest t, the set of forests that are EF-bisimilar to t can be defined in EF.

Note 3. In Theorem 4.3, invariance under EF-bisimulation is used as a property of the language. We could have a different statement, where invariance is a property of the syntactic algebra (e.g. all languages recognized by the algebra are invariant under EF-bisimulation). The different statement would be better suited towards variety theory, à la Eilenberg [9]. We intend to develop such a theory.

Invariance under EF-bisimulation and (3) are sufficient

We now show the more difficult part of Theorem 4.3. Fix a surjective morphism $\alpha : A^{\triangle} \to (H, V)$ and suppose that the target ω-forest algebra satisfies condition (3) and that the morphism is invariant under EF-bisimulation (any two EF-bisimilar forests have the same image). For a forest t, we use the name *type of* h for the value $\alpha(h)$. We will show that for any $h \in H$, the language L_h of forests of type h is definable by a forest formula of EF. This shows that the two conditions in Theorem 4.3 are sufficient for definability in EF, by taking α to be the syntactic morphism of L.

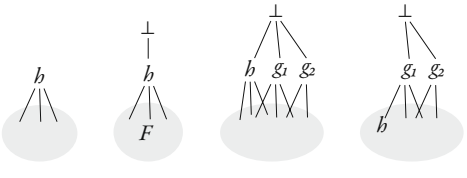
The proof is by induction on the size of H. The induction base, when H has only one element, is trivial. In this case all forests have the same type, and the appropriate formula is *true*.

We now proceed to the induction step. We say that an element $h \in H$ is *reachable* from $g \in H$ if there is some $v \in V$ with $h = vg$.

Lemma 4.5. *The reachability relation is transitive and antisymmetric.*

We say that an element $h \in H$ is *minimal* if it is reachable from all $g \in H$. (The name minimal, instead of maximal, is traditional in algebra. The idea is that the set of elements reachable from h is minimal.) There is at least one minimal element, since for every $v \in V$ an element $v(h_1 + \cdots + h_n)$ is reachable from each h_i. Since reachability is antisymmetric, this minimal element is unique, and we will denote it using the symbol $\bot$. An element $h \neq \bot$ is called *subminimal* if the elements reachable from h are a subset of $\{h, \bot\}$.

Recall that our goal is to give, for any $h \in H$, a formula of EF that defines the set L_h of forests with type h. Fix then some $h \in H$. We consider four cases (shown on the picture):



1. h is minimal.
2. h is subminimal, and there is exactly one subminimal element.
3. h is subminimal, but there are at least two subminimal elements.
4. h is neither minimal nor subminimal.

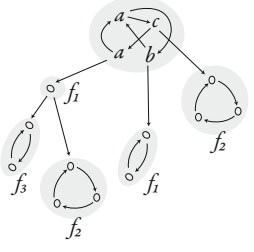
Note that the first case follows from the remaining three, since a forest has type $\bot$ if and only if it has none of the other types. Therefore the formula for $h = \bot$ is obtained by negating the disjunction of the formulas for the other types. Cases 3 and 4 are treated in the appendix. The more interesting case is 2, which is treated below.

Case 2. We now consider the case when h is a unique subminimal element.

Let F be the set of all elements different from h, from which h is reachable. In other words, F is the set of all elements from H beside h and $\bot$. Thanks to cases 3 and 4, for every $f \in F$ we have a formula φ_f that defines the set L_f of forests with type f. We write φ_F for the disjunction $\bigvee_{f \in F} \varphi_f$.

Let t be a forest. We say two nodes x, y of the forest are *in the same component* if the subtree $t|_x$ is a subtree of the subtree $t|_y$ and vice versa. Each regular forest has finitely many components. There are two kinds of component: *connected components*, which contain infinitely many nodes, and *singleton components*, which contain a single node. Since any two nodes in the same component are EF-bisimilar (i.e. their subtrees are EF-bisimilar), we conclude that two nodes in the same component have the same type. Therefore, we can speak of the type of a component. A tree is called *prime* if it has exactly one component with a type outside F. Note that the component with a type outside F must necessarily be the root component (the one that contains the root), since no type from F is reachable from types outside F. Depending on the kind of the root component, a prime tree is called a *connected prime* or *singleton prime*.

The *profile* of a prime tree t is a pair in $P(F) \times (A \cup P(A))$ defined as follows. On the first coordinate, we store the set $G \subseteq F$ of types of components with a type in F. On the second coordinate, we store the labels that appear in the root component. If the tree is connected prime, this is a set $B \subseteq A$ of labels (possibly containing a single label), and if the tree is singleton prime, this is a single label $b \in A$. In the first case, the profile is called a *connected profile*, and in the second case the profile is called a *singleton profile*. The picture shows a connected prime tree with connected profile $(\{f_1, f_2, f_3\}, \{a, b, c\})$.



In the appendix, we show that all prime trees with the same profile have the same type. Therefore, it is meaningful to define the set prof_h of profiles of prime trees of type h.

Proposition 4.6. Let π be a profile. There is an EF formula φ_π such that
- Any prime tree with profile π satisfies φ_π.
- Any regular forest satisfying φ_π has type h if $\pi \in \mathit{prof}_h$ and $\bot$ otherwise.

The above proposition is shown in the appendix. We now turn our attention from prime trees to arbitrary forests. Let t be a forest. The formula which says when the type is h works differently, depending on whether t has a prime subtree or not. This case distinction can be done in EF, since not having a prime subtree is expressed by the formula $\mathsf{AG}^* \varphi_F$.

There is no prime subtree. We write $\sum\{g_1, \ldots, g_n\}$ for $g_1 + \cdots + g_n$. By invariance under EF-bisimulation, this value does not depend on the order or multiplicity of elements in the set. Suppose $\sum G = \bot$ and t has subtrees with each type in G. Thanks to (1), the type of t satisfies $\alpha(t) + \sum G = \alpha(t)$, and hence $\alpha(t) = \bot$.

Therefore, a condition necessary for having type h is

$$\neg \bigvee_{G \subseteq F,\ \sum G = \bot} \bigwedge_{g \in G} \mathsf{EF}^* \varphi_g. \tag{4}$$

By the same reasoning, a condition necessary for having type h is

$$\bigvee_{G \subseteq F,\ \sum G = h} \bigwedge_{g \in G} \mathsf{EF}^* \varphi_g. \tag{5}$$

It is not difficult to show that conditions (4) and (5) are also sufficient for a forest with no prime subtrees to have type h.

There is a prime subtree. As previously, a forest t with type h cannot satisfy (4). What other conditions are necessary? It is forbidden to have a subtree with type $\bot$. Thanks to Proposition 4.6, t must satisfy:

$$\neg \bigvee_{\pi \notin prof_h} \mathsf{EF}^* \varphi_\pi. \tag{6}$$

Since t has a prime subtree, its type is either h or $\bot$. Suppose that t has a subtree with type $f \in F$ such that $f + h = \bot$. By (1), we would have $\alpha(t) + f = \alpha(t)$, which implies that the type of t is $\bot$. Therefore, t must satisfy

$$\bigwedge_{f \in F,\ f+h=\bot} \neg \mathsf{EF}^* \varphi_f. \tag{7}$$

Let us define C to be the labels that preserve h, i.e. the labels $a \in A$ such that $\alpha(a)h = h$. If a forest has type h then it cannot have a subtree as where $a \notin C$ and s has type h or $\bot$. This is stated by the formula:

$$\mathsf{AG}^* \bigwedge_{c \notin C} (c \to \mathsf{AG} \varphi_F). \tag{8}$$

As we have seen, conditions (4) and (6)–(8) are necessary for a forest with a prime subtree t having type h. In the following lemma, we show that the conditions are also sufficient. This completes the analysis of Case 2 in the proof of Theorem 4.3, since it gives a formula that characterizes the set L_h of forests whose type is the unique subminimal element h.

Lemma 4.7. *A forest with a prime subtree has type h if it satisfies conditions* (4) *and* (6)–(8).

5 Concluding Remarks

We have presented an algebra for infinite trees, and used it to get an effective characterization for the logic EF. In the process, we developed techniques for dealing with the algebra, such as algorithms deciding identities, or a kind of

Green's relation (the reachability relation). It seems that an important role is played by what we call connected components of a regular forest.

There are some natural ideas for future work. These include developing a variety theory, or finding effective characterizations for other logics. Apart from logics that have been considered for finite trees, there are some new interesting logics for infinite trees, which do not have counterparts for finite trees. These include weak monadic-second order logic, or fragments of the μ-calculus with bounded alternation.

However, since we are only beginning to study the algebra for infinite trees, it is important to know if we have the "right" framework (or at least one of the "right" frameworks). Below we discuss some shortcomings of ω-forest algebra, and comment on some alternative approaches.

Shortcomings of ω-forest algebra. A jarring problem is that we have an infinite number of operations and, consequently, an infinite number of axioms. This poses all sorts of problems.

It is difficult to present an algebra. One cannot, as with a finite number of operations, simply list the multiplication tables. Our solution, as presented in the appendix, is to give an algorithm that inputs the name of the operation and produces its multiplication table. In particular, this algorithm can be used to test validity of identities, since any identity involves a finite number of operations.

It is difficult to prove that something is an ω-forest algebra, since there are infinitely many axioms to check. Our solution is to define algebras as homomorphic images of the free algebra, which guarantees that the axioms hold. We give an algorithm that computes the syntactic algebra of a regular forest language.

We have proved that any regular language is recognized by a finite ω-forest algebra. A significant shortcoming is that we have not proved the converse. We do not, however, need the converse for effective characterizations, as demonstrated by our proof for EF. The effective characterization begins with a regular language, and tests properties of its syntactic algebra (therefore, algebras that recognize non-regular languages, if they exist, are never used).

An important advantage of using algebra is that properties can be stated in terms of identities. Do we have this advantage in ω-forest algebra? The answer is mixed, as witnessed by Theorem 4.3. One of the conditions is an identity, namely (3). However, for the other condition, invariance under EF-bisimulation, we were unable to come up with an identity (or a finite set of identities). This contrasts the case of finite trees, where invariance under EF-bisimulation is characterized by two identities. Below we describe an idea on to modify the algebra to solve this problem.

A richer algebra? In preliminary work, we have tried to modify the algebra. In the modified algebra, the context sort is richer, since contexts are allowed to have multiple occurrences of the hole (we still allow only one type of hole). This abundance of contexts changes the interpretation of identities, since the context variables quantify over a larger set. Preliminary results indicate that invariance

under EF-bisimulation is described by the identities (1) and (2) – but with the new interpretation – as well as the following two identities:

$$(v(u+w))^\infty = (vuw)^\infty, \qquad (vuw)^\infty = (vwu)^\infty.$$

We intend to explore this richer algebra in more detail. However, allowing a richer context sort comes at a cost. First, it seems that the size of the context sort is not singly exponential, as here, but doubly exponential. Second, there are forest languages definable in first-order logic that are not aperiodic, i.e. do not satisfy $v^\omega = v^{\omega+1}$.

Where is the Ramsey theorem? An important tool in algebra for infinite words is the Ramsey theorem, which implies the following fact: for every morphism $\alpha : A^* \to S$ into a finite monoid, every word $w \in A^\omega$ has a factorization $w = w_0 w_1 w_2 \cdots$ such that all the words $w_1, w_2, \ldots$ have the same image under α. This result allows one to extend a morphism into a Wilke algebra from ultimately periodic words to arbitrary words.

It would be very nice to have a similar theorem for trees. This question has been independently investigated by Blumensath [2], who also provides an algebraic framework for infinite trees. Contrary to our original expectations, we discovered that a Ramsey theorem for trees is not needed to provide effective characterizations.

Acknowledgement. We thank Wojciech Czerwiński for many helpful discussions.

References

1. Benedikt, M., Segoufin, L.: Regular tree languages definable in FO. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 327–339. Springer, Heidelberg (2005); A revised version, correcting an error from the conference paper, `www.lsv.ens-cachan.fr/~segoufin/Papers/`
2. Blumensath, A.: Recognisability for algebras of infinite trees (unpublished, 2009)
3. Bojańczyk, M.: Two-way unary temporal logic over trees. In: Logic in Computer Science, pp. 121–130 (2007)
4. Bojańczyk, M.: Effective characterizations of tree logics. In: PODS, pp. 53–66 (2008)
5. Bojańczyk, M., Segoufin, L.: Tree languages defined in first-order logic with one quantifier alternation. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 233–245. Springer, Heidelberg (2008)
6. Bojańczyk, M., Segoufin, L., Straubing, H.: Piecewise testable tree languages. In: Logic in Computer Science, pp. 442–451 (2008)
7. Bojańczyk, M., Walukiewicz, I.: Characterizing EF and EX tree logics. Theoretical Computer Science 358(2-3), 255–273 (2006)
8. Bojańczyk, M., Walukiewicz, I.: Forest algebras. In: Automata and Logic: History and Perspectives, pp. 107–132. Amsterdam University Press (2007)
9. Eilenberg, S.: Automata, Languages and Machines, vol. B. Academic Press, New York (1976)
10. Perrin, D., Pin, J.-É.: Infinite Words. Elsevier, Amsterdam (2004)
11. Wilke, T.: An algebraic theory for languages of finite and infinite words. Inf. J. Alg. Comput. 3, 447–489 (1993)