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

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.

Acknowledgement. We thank Wojciech Czerwiński for many helpful discussions.

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.

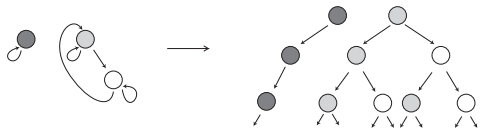
Operations on forests and contexts. We define two types of operations on forests and contexts: a *(horizontal) concatenation operation*, written additively, and a *(vertical) composition operation*, written multiplicatively. In general, neither concatenation nor composition is commutative.

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

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

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

$$\begin{array}{l} s_1, \ldots, s_m \in H_A \\ p_1, \ldots, p_n \in V_A \end{array} \quad \mapsto \quad \text{unfold}_\tau(s_1, \ldots, s_m, p_1, \ldots, p_n) \in H_A.$$

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

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.

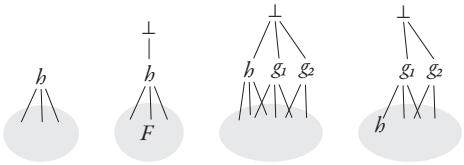
The proof is by induction on the size of H. The induction base, when H has only one element, is trivial. In this case all forests have the same type, and the appropriate formula is *true*.

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.

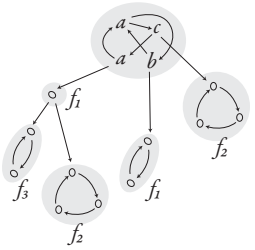
Note that the first case follows from the remaining three, since a forest has type $\bot$ if and only if it has none of the other types. Therefore the formula for $h = \bot$ is obtained by negating the disjunction of the formulas for the other types. Cases 3 and 4 are treated in the appendix. The more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

References

1. M. Benedikt and L. Segoufin. Regular languages definable in FO. In *Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *Lecture Notes in Computer Science*, pages 327–339, 2005. A revised version, correcting an error from the conference paper, is available at `www.lsv.ens-cachan.fr/ segoufin/Papers/`.
2. A. Blumensath. Recognisability for algebras of infinite trees. *Unpublished*, 2009.
3. M. Bojańczyk. Two-way unary temporal logic over trees. In *Logic in Computer Science*, pages 121–130, 2007.
4. M. Bojańczyk. Effective characterizations of tree logics. In *PODS*, pages 53–66, 2008.
5. M. Bojańczyk and L. Segoufin. Tree languages definable with one quantifier alternation. In *International Colloquium on Automata, Languages and Programming*, volume 5126 of *Lecture Notes in Computer Science*, pages 233–245, 2008.
6. M. Bojańczyk, L. Segoufin, and H. Straubing. Piecewise testable tree languages. In *Logic in Computer Science*, pages 442–451, 2008.
7. M. Bojańczyk and I. Walukiewicz. Characterizing EF and EX tree logics. *Theoretical Computer Science*, 358(2-3):255–273, 2006.
8. M. Bojańczyk and I. Walukiewicz. Forest algebras. In *Automata and Logic: History and Perspectives*, pages 107–132. Amsterdam University Press, 2007.
9. S. Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, New York, 1976.
10. D. Perrin and J.-É. Pin. *Infinite Words*. Elsevier, 2004.
11. T. Wilke. An algebraic theory for languages of finite and infinite words. *Inf. J. Alg. Comput.*, 3:447–489, 1993.

A Appendix to Section 2

A.1 Definition of unfolding

Let τ be a recursion term, where the free recursion variables are Y. The *unfolding* $\mathrm{unfold}_\tau[\eta]$ of a recursion term τ is a forest over $A \cup Y$, with labels from Y only in the leaves, which is defined in the natural way by induction on the structure of τ:

$$\begin{aligned}
\mathrm{unfold}_0[\eta] &= 0 \\
\mathrm{unfold}_{\tau_1+\cdots+\tau_n}[\eta] &= \mathrm{unfold}_{\tau_1}[\eta] + \cdots + \mathrm{unfold}_{\tau_n}[\eta] \\
\mathrm{unfold}_z[\eta] &= \eta(z) \\
\mathrm{unfold}_{z\tau}[\eta] &= \eta(z) \cdot \mathrm{unfold}_\tau[\eta] \\
\mathrm{unfold}_y[\eta] &= y
\end{aligned}$$

The only interesting case is $\mathrm{unfold}_{\nu y.\tau}[\eta]$. The result of $\mathrm{unfold}_\tau[\eta]$ is a forest t, where the recursion variable y may occur in some leaves. For such a forest t, and a forest s, we define $t[y \mapsto s]$ to be the forest obtained from t by replacing each leaf with label y by the forest s. We define $\mathrm{unfold}_{\nu y.\tau}[\eta]$ to be the unique forest s that satisfies the fixpoint property $s = t[y \mapsto s]$. This fixpoint can be obtained as the limit of the sequence

$$s_1 = t, \qquad s_{i+1} = t[y \mapsto s_i].$$

Since the variable y is guarded inside τ, it does not occur in the root of the forest t. In particular, the limit of the sequence is well defined, and is a forest of bounded degree.

A.2 Proof of Lemma 2.1

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

Proof
The if implication is proved by induction on the size of the recursion scheme. (Therefore, the induction has to be extended to recursion terms, which also generate forests, but over an alphabet containing recursion variables.)

We only need to prove the only if implication for regular trees, since a regular forest is a finite concatenation of regular trees, and recursion schemes admit concatenation. Let T be the set of all subtrees of a given regular tree t. We will build a recursion scheme which unfolds to t. For each $t \in T$, we will have a recursion variable y_t and a context-sorted label variable z_t. Every nonempty subtree of t has a form

$$t = a_t(t_{t,1} + \cdots + t_{t,k_t})$$

for some $a_t \in A$ and $t_{t,1}, \ldots, t_{t,k_t} \in T$. Let $S \subseteq T$. By induction on $T \setminus S$, we define for each $t \in T$ a recursion term $\tau_{S,t}$, which unfolds to t provided that the

trees in $S \subseteq T$ can be referred to using appropriate recursion variables. We put $\varphi_{S,0} = 0$, and

$$\tau_{S,t} = \begin{cases} \nu y_t.z_t\big(\varphi_{S\cup\{t\}}(t_{t,1}) + \cdots + \varphi_{S\cup\{t\}}(t_{t,k_t})\big) & \text{for } t \notin S \\ y_t & \text{for } t \in S \end{cases}$$

It is easy to see for any $t \in T$, the tree t is equal to $\tau_{\emptyset,t}[\eta]$, where η is a valuation that maps each z_t to $a_t\Box$. □

A.3 Proof of Theorem 2.2

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

Proof
We can encode a forest t in a binary tree $enc(t)$ using the first-child, next-sibling encoding. If a forest language L is recognized by a forest automaton, then $enc(L)$ is recognized by a parity automaton over binary trees, of polynomial size. If a language of binary trees L is recognized by a parity automaton over binary trees, then $enc^{-1}(L)$ is recognized by a forest automaton (there might be an exponential blowup in this direction, due to translating an automaton into a monoid). Both enc and enc^{-1} preserve regularity of forests. Hence the statement of the theorem follows from the analogous statements for binary trees. □

B Appendix to Section 3

In this part of the appendix, we discuss the axioms of ω-forest algebra. These axioms are designed so that $A^{\triangle}$ becomes the free object generated by an alphabet A, as stated in Theorem 3.1. We choose a simple way to enforce this: the axioms are simply all equalities that are true in objects of the form $A^{\triangle}$. Before we present these axioms, we need to say what an axiom is, which requires some universal algebra.

Some universal algebra. Recall that the notion of homomorphism is defined only in terms of the operations that are allowed in the signature, so we can define homomorphisms before stating the axioms required from an ω-forest algebra. A *homomorphism* (we will also call it a *morphism*) is defined as usual for two-sorted structures. That is, a homomorphism α is a pair of functions (α_H, α_V), with α_H mapping the first forest sort of the source structure to the forest sort of the target structure, and α_V mapping the context sort of the source structure to the context sort of the target structure. The mappings are required to preserve all the operations in the usual sense. In particular, for any (m, n)-ary recursion

scheme τ and any $h_1, \ldots, h_m$ in the source forest sort and $v_1, \ldots, v_n$ in the source context sort, we have

$$\alpha_H(\mathrm{unfold}_\tau(h_1, \ldots, h_m, v_1, \ldots, v_n)) = \\ \mathrm{unfold}_\tau(\alpha_H(h_1), \ldots, \alpha_H(h_m), \alpha_V(v_1), \ldots, \alpha_V(v_n)).$$

To avoid clutter, we omit the subscript from the notation α_H, α_V and simply write α where there is no confusion as to the sort of the argument (e.g. $\alpha(v)$ for $v \in V$ means $\alpha_V(v)$). If (H, V) and (G, W) are structures over the signature of ω-forest algebra, we write

$$\alpha : (H, V) \to (G, W)$$

for a homomorphism from (H, V) to (G, W). We also use the name morphism instead of homomorphism.

What is an axiom? Intuitively, an axiom is a pair of terms with free variables that should be made equal for all possible substitutions. Since we are working in a two-sorted algebra, when defining terms, we need to keep track of the sort used for the subterms and the variables. We fix two disjoint countably infinite sets X_H and X_V of variables, which are intended to represent forest-sorted and context-sorted variables, respectively. We write X for the union of $X_H \cup X_V$. We define terms over the variables X by induction on the size of the term, in the natural way. Terms are either *forest-valued* or *context-valued*, depending on the target sort of the outermost operation. The variables from X_H are forest-valued terms, and the variables from X_V are context-valued terms. A *forest-valued axiom* is a pair of forest-valued terms (ϕ, π) over the variables X. Such an axiom is said to hold in a two-sorted structure (H, V) over the signature of ω-forest algebra if for every valuation η that assigns elements of H to X_H and elements of V to X_V, the values $\phi[\eta], \pi[\eta] \in H$ are the same. We define context-valued axioms in a similar way.

The axioms of ω-forest algebra. Abusing somewhat the notation, we write $X^\triangle$ for the ω-forest algebra where the forest (context) sort contains all forests (contexts) with labels from X_H in the leaves and labels from X_V in inner nodes.

Definition 1. *The* axioms of ω-forest algebra *consist of all axioms which hold in* $X^\triangle$.

The axioms include, for instance,

$$\mathrm{unfold}_{\nu y.zzy}(x) = \mathrm{unfold}_{\nu y.zy}(x)$$

for each context-sorted variable x, since both sides evaluate in $X^\triangle$ to an infinite tree $xxx\cdots$.

Fact B.1 shows that the basic operations of ω-forest algebra, such as concatenation and composition, are mainly syntactic sugar for recursion schemes.

Fact B.1 For every forest-valued term ϕ there is a recursion scheme τ such that $\phi = \text{unfold}_\tau$ is an axiom of ω-forest algebra.

Proof
Let ϕ be a term over variables $x_1, \ldots, x_m \in X_H$ and $x'_1, \ldots, x'_n \in X_V$. We want to build an equivalent recursion scheme, which we denote by $\varphi(\phi)$, i.e.

$$\phi = \text{unfold}_{\varphi(\phi)}(x_1, \ldots, x_m, x'_1, \ldots, x'_n). \tag{9}$$

The proof is by induction on two parameters, first the size of the term ϕ, and second the size of the left subterm of ϕ (0 if there is no left subterm). The term ϕ is forest-valued, but since some of its subterms might be context-valued, we must define φ also for context-valued terms. We do this by slightly abusing definition and allowing a hole in recursion schemes (there can only be a single hole, and this hole cannot be under the scope of a recursion νy). Below, we write ϕ_s, ϕ_t for forest-valued terms, ϕ_p, ϕ_s for context-valued terms, and ϕ for any term.

For basic operations we write (symmetric cases were ommited):

$$\begin{aligned}
\varphi(0) &= 0 & \varphi(\phi + \phi_s) &= \varphi(\phi) + \varphi(\phi_s) \\
\varphi(x_i) &= z_i & \varphi((\phi_p \phi_q)\phi) &= \varphi(\phi_p(\phi_q \phi)) \\
\varphi(x'_i) &= z'_i \Box & \varphi((\phi_p + \phi_s)\phi) &= \varphi(\phi_p \phi) + \varphi(\phi_s) \\
\varphi(x'_i \phi) &= z'_i \varphi(\phi) & \varphi(\phi_p(\Box + \phi_s)\phi) &= \varphi(\phi_p(\phi + \phi_s))
\end{aligned}$$

For an (i, j)-ary recursion scheme τ using label variables $z_1, \ldots, z_i$ and $z'_1, \ldots, z'_j$ we write

$$\begin{aligned}
\varphi\big(\text{unfold}_\tau(\phi^1_t, \ldots, \phi^i_t, \phi^1_p, \ldots, \phi^j_p)\big) = \tau\big[&z_1 \mapsto \varphi(\phi^1_t), \ldots, z_i \mapsto \varphi(\phi^i_t), \\
&z'_1 \tau_1 \mapsto \varphi(\phi^1_p)[\Box \mapsto \tau_1], \ldots, z'_j \tau_j \mapsto \varphi(\phi^j_p)[\Box \mapsto \tau_j]\big].
\end{aligned}$$

It is easy to see that (9) is an axiom of ω-forest algebra. □

B.1 Proof of Theorem 3.1

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

Proof
Uniqueness follows from $A\Box$ generating $A^\triangle$. We want to define the value $\alpha(t)$ for a forest $t \in A^\triangle$ (the construction for contexts is similar). There exists a term ϕ over context-sorted variables X_V and a valuation $\eta\colon X_V \to A\Box$ such that $\phi[\eta] = t$. We define $\alpha(t) = \phi[f \circ \eta]$. Thanks to the axioms of ω-forest algebra, this definition does not depend on the choice of ϕ. It is easy to that α is a morphism. □

B.2 Proof of Proposition 3.3

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 $\beta \circ \alpha = \alpha_L$.

Proof
As in Theorem 3.2, this result follows from universal algebra, and does not depend on the operations of ω-forest algebra. Nevertheless, we include a proof.

If two forests are L-equivalent, then they either both belong to L, or both are outside L (otherwise, they could be distinguished by the very simple term $x \in X_H$). Hence, the syntactic morphism recognizes L.

We now proceed to the second part of the proposition. Suppose that a morphism α recognizes β. We claim that if α_L gives different results for two forests s, t, then so does β. This claim shows that for every element $h \in H$, there is a unique candidate for $\beta(h) \in H_L$ (a similar proof works for contexts). To prove the claim, suppose that α_L gives different results for s, t. Then there must exist a forest-valued term ϕ, a valuation η, and a forest-sorted variable x, such that only one of the forests $\phi[\eta[x \mapsto s]]$ and $\phi[\eta[x \mapsto t]]$ belongs to L. If β would give the same result for s, t, it would also give the same result for $\phi[\eta[x \mapsto t]]$ and $\phi[\eta[x \mapsto t]]$, contradicting the assumption that β recognizes L. □

C Appendix to Section 4

C.1 Invariance under EF-bisimulation and (3) are necessary.

In this section we show that the two conditions in Theorem 4.3 are necessary for definability in EF. Fix a forest language L that is definable by a forest formula of EF. Let the syntactic morphism of L be

$$\alpha_L : A^{\triangle} \to (H, V).$$

Invariance under EF-bisimulation follows immediately from Theorem 4.2.

We now turn to condition (3). Let n be the idempotent power ω of V. That is, for any context type $v \in V$, the contexts v^n and $v^n \cdot v^n$ are equal. Such a power exists, since V is a finite monoid. Let $i \cdot n$ be a multiple of n that is larger than the EF-nesting depth of the forest formula defining L. We will show that

$$v^{i \cdot n} h = (v + v^{i \cdot n} h)^{\infty} \qquad \text{for any } v \in V \text{ and } h \in H.$$

This establishes (3), since $v^{i \cdot n} = v^n = v^{\omega}$. By unraveling the definition of the syntactic ω-forest algebra, and using Fact B.1, we need to show that for any guarded context p, any forest t, any recursion scheme τ, any valuation η of the variables in τ with elements of $A^{\triangle}$, and any forest-sorted z variable in τ, we have

$$\mathrm{unfold}_{\tau}[\eta[z \mapsto p^{i \cdot n} t]] \in L \qquad \text{iff} \qquad \mathrm{unfold}_{\tau}[\eta[z \mapsto (p + p^{i \cdot n} t)^{\infty}]] \in L. \tag{10}$$

We will show a stronger result, namely that Duplicator wins the $i \cdot n$-round EF game on the two forests

$$\mathrm{unfold}_\tau[\eta[z \mapsto p^{i \cdot n} t]] \qquad \text{and} \qquad \mathrm{unfold}_\tau[\eta[z \mapsto p + p^{i \cdot n} t)^\infty]].$$

To get rid of the unfolding above, we will use the following fact, which basically says that winning the m-round game behaves like a congruence with respect to unfolding:

Fact C.1 Let τ be a recursion scheme with a single free forest-sorted variable, and let s, t be forests. If player Duplicator wins the m-round EF-game on s and t, then for any τ and η he also wins the m-round EF-game on

$$\mathrm{unfold}_\tau[\eta[z \mapsto s]] \qquad \text{and} \qquad \mathrm{unfold}_\tau[\eta[z \mapsto t]].$$

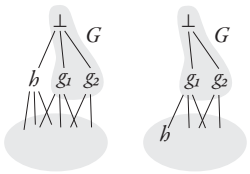
Thanks to this fact, instead of (10), we only have to prove that Duplicator wins the $i \cdot n$-round EF game on the two forests

$$p^{i \cdot n} t \qquad \text{and} \qquad (p + p^{i \cdot n} t)^\infty.$$

It is not difficult to come up with a winning strategy for Duplicator in the above game.

C.2 Cases 3 and 4.

These cases can be illustrated as:



For these cases, we use a standard method. Let G be the elements that are different from h, and from which h is not reachable. We claim that this set has at least two elements in both cases 3 and 4. In case 3 G contains $\bot$ and the other subminimal element are in G. In case 4, G contains $\bot$ and all subminimal elements.

The idea is to squash all elements from G into a single element. Since G has at least two elements, we can use the induction assumption on a smaller algebra. The trick is showing that this squashing is a congruence, which requires dealing with the unfoldings.

Let us define the following equivalence relation on H: $g \sim f$ holds if either $g = f$ or both g, f belong to G. In other words $\sim$ identifies all types in G. We extend this relation to V by setting $v \sim w$ if either $v = w$ or both v, w belong to the set

$$W = \{w \in V : wH \subseteq G\}.$$

The relation $\sim$ is a two-sorted equivalence relation, i.e. a pair of equivalence relations, one for each of the two sorts H and V of ω-forest algebra. The following lemma shows that it is actually a congruence, i.e. for every operation of the algebra, the equivalence class of the result is uniquely determined by the equivalence classes of the arguments.

Lemma C.2 The relation $\sim$ is a congruence.

Proof
We only do the case for unfoldings of recursion schemes. Fix an (m, n)-ary recursion scheme τ. We need to show that for any elements

$$g_1 \sim f_1, \quad \ldots, \quad g_m \sim f_m, \qquad v_1 \sim w_1, \quad \ldots, \quad v_n \sim w_n$$

respectively from H and V we have

$$\text{unfold}_\tau(g_1, \ldots, g_m, v_1, \ldots, v_n) \sim \text{unfold}_\tau(f_1, \ldots, f_m, w_1, \ldots, w_n). \tag{11}$$

If all equivalent elements are equal (i.e. $g_i = f_i$, $v_i = w_i$) then (11) is obvious. If it is not the case then we have that for some i either $g_i, f_i \in G$ or $v_i, w_i \in W$. We claim that in this situation both unfolds belong to G. It is true, since every unfold which uses $g \in G$ can be decomposed as $\phi[\eta] \cdot g$ for some term ϕ and valuation η and from the definition of G we have $Vg \subseteq G$. Similarly every unfold which uses $v \in W$ can be decomposed as $\phi_1[\eta_1] \cdot v \cdot \phi_2[\eta_2]$ and $VvH \subseteq G$. $\square$

Let β be the function which maps an element of the forest algebra (H, V) to its equivalence class. By the above lemma, β is a homomorphism

$$\beta : (H, V) \to (H, V)_{/\sim},$$

where the elements of the ω-forest algebra on the right hand side are equivalence classes under $\sim$, and the operations are inherited from (H, V). Note also the same forests are mapped to h by α and by $\beta \circ \alpha$. Therefore, we can use the induction assumption, applied to the homomorphism

$$\beta \circ \alpha : A^\triangle \to (H, V)_{/\sim},$$

to obtain a formula for the set of forests that are mapped to h by α.

C.3 Proof of Lemma 4.5

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

Proof
Note that reachability might not be reflexive, since we V is not necessarily a monoid. Transitivity is obvious. For antisymmetry, we prove that invariance under EF-bisimulation implies property (1). Indeed, since α is surjective, there must be some context p with $\alpha(p) = v$ and some forest t with $\alpha(t) = t$. Since the

forests $pt + t$ and pt are EF-bisimilar, their types must be equal, and hence (1) holds.

Suppose that g is reachable from h, and vice versa. To prove antisymmetry, we need to show that show $g = h$. By assumption there are $v, w \in V$ with $g = wh$ and $h = vg$. Then we have

$$g = wh = wvg \overset{(1)}{=} wvg + vg = g + vg \overset{(1)}{=} vg = h.$$

□

C.4 Profiles determine types

Lemma C.3 All prime trees with the same profile have the same type.

Proof
Let s, t be prime trees. Let G be the first coordinate of the profile, and let $\{s_1, \ldots, s_k\}$ be all the subtrees of s with types in F, likewise for $\{t_1, \ldots, t_n\}$ in t. By property (1) we can assume that $\{\alpha(s_1), \ldots, \alpha(s_k)\} = G = \{\alpha(t_1), \ldots, \alpha(t_n)\}$.

We modify the tree t by replacing every subtree t_i with some tree s_j with the same type ($\alpha(t_i) = \alpha(s_j)$). This operation does not change the type of t. The resulting tree t' is EF-bisimilar to tree s, and therefore has the same type, since α is invariant under EF-bisimulation. □

C.5 Proof of Proposition 4.6

Lemma 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 $\pi \in prof_h$ and $\bot$ otherwise.*

Proof
First consider the case when the profile π is a singleton profile (G, b). The formula φ_π says that the root label is b, all proper subtrees have types in G, and any type in G appears in some proper subtree. Since the types in G all belong to F, we are allowed to use formulas φ_g that define the forests with type g. Therefore, having profile (G, b) is defined by the following formula:

$$b \quad \wedge \quad \mathsf{AG} \bigvee_{g \in G} \varphi_g \quad \wedge \quad \bigwedge_{g \in G} \mathsf{EF}\varphi_g.$$

Note that the formula above satisfies a stronger property than required by the proposition, since it describes exactly the trees with profile (G, b), i.e. we could strengthen the second property in the proposition to "any regular forest satisfying φ_π is a tree of profile (G, b)". This will not, however, be the case in the construction below for connected profiles.

We now consider the case when the profile π is a connected profile (G, B). Let us collect some properties of a prime tree t with this profile. Below we use

the term "type of a node" to refer to the type of the node's subtree. Recall that for $I \subseteq F$, we write φ_I for the disjunction $\bigvee_{f \in I} \varphi_f$. From the definition of a prime tree, a node in a prime tree is in the root component if and only if it has a type outside F. In particular, the tree t has a type outside F:

$$\neg \varphi_F. \tag{12}$$

Since the profile is (B, G), we know that all nodes outside the root component have a type in G, which is stated by the formula:

$$\mathsf{AG}^*(\varphi_F \to \varphi_G). \tag{13}$$

In (13), we could have used AG instead of AG^*, since the root of the tree is guaranteed to have a type outside F.

Again from the definition of the profile, we know that all nodes in the root component have a label in B:

$$\mathsf{AG}^*\Big(\neg \varphi_F \to \bigvee_{b \in B} b\Big). \tag{14}$$

Since the root component is connected, any node in the root component has proper descendant that is still in the root component. In particular, a node in the root component has proper descendants in the root component with all labels from B.

$$\mathsf{AG}^*\Big(\neg \varphi_F \to \bigwedge_{b \in B} \mathsf{EF}(b \wedge \neg \varphi_F)\Big). \tag{15}$$

In the same way, any node in the root component has proper descendants with all possible types from G.

$$\mathsf{AG}^*\Big(\neg \varphi_F \to \bigwedge_{g \in G} \mathsf{EF}\varphi_g\Big). \tag{16}$$

Let $\varphi_{(B,G)}$ be the conjunction of the formulas (12)–(16). From the discussion above we obtain the first property required by the proposition: any prime tree with profile (B, G) satisfies $\varphi_{(B,G)}$. We now proceed to show the second property, namely that any forest satisfying $\varphi_{(B,G)}$ has type h if $(B, G) \in \mathit{prof}_h$ and $\bot$ otherwise.

Let t be a regular forest that satisfies (12)–(16). Let $t_1, \ldots, t_m$ be all the subtrees of t that have types in F, this set is finite since t is regular. Let $b_1, \ldots, b_n$ be all the labels in B. Let us define the following tree:

$$s = (b_1 b_2 \cdots b_n(\Box + t_1 + \cdots + t_m))^\infty.$$

The types of the trees $t_1, \ldots, t_n$ are included in G thanks to (14). These types are actually exactly G, thanks to (16). Therefore, s is a connected prime tree with profile (B, G), and so it has type h if $(B, G) \in \mathit{prof}_h$ and $\bot$ otherwise. We will complete the proof by showing that the trees t and s are EF-bisimilar, even when t is not prime.

Why are s and t EF-bisimilar? Suppose that at the beginning of a round in the EF game, we have a subtree s' of s and a subtree t' of t. Duplicator's strategy is to preserve the following invariant:

> If the type of s' or t' is in F, then the trees s' and t' are equal. Otherwise, the types of both s' and t' are outside F, and the trees have the same root labels.

□

C.6 Proof of Lemma 4.7

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

Proof
By induction on the number of prime components in a forest t, we show that if t satisfies conditions (4) and (6)–(8), its type is not $\bot$. This gives the statement of the lemma, since a forest with a prime subtree cannot have a type in F, so it has type h.

The induction base is when t is prime. The type must be h, since $\bot$ is ruled out by (6).

Consider now the induction step. Suppose first that t is a concatenation of trees $t_1 + \cdots + t_n$. Suppose that the statement holds for $t_1, \ldots, t_n$, which have types $G = \{f_1, \ldots, f_n\} \not\ni \bot$. The type of t is $\sum G$. If $G \subseteq F$, then (4) implies that $\sum G$ is not $\bot$. If $h \in G$, then (7) implies that $\sum G = h$.

It remains to show the lemma when t is a tree. If the root component is a singleton component, a simple argument shows that the type of t is not $\bot$. We are left with the case when t is a tree where the root component is connected. Let $a_1, \ldots, a_m$ be the labels in the root component. Let $t_1, \ldots, t_n$ be all the subtrees of t that have a smaller number of prime components, and let $f_1, \ldots, f_n$ be their types. Since t is not prime, some f_i is h. Since the root component is connected, each node in the root component has a descendant with type h. Therefore, by (8), all the labels $a_1, \ldots, a_m$ belong to C.

It is not difficult to show that t is EF-bisimilar to, and thus has the same type as, the tree

$$s = (a_1 \cdots a_m(\Box + t_1 + \cdots + t_n))^\infty.$$

By (7), we have $h = h + f_1 + \cdots + f_n$. Therefore, the tree s has the same type as

$$(a_1 \cdots a_m(\Box + t_1))^\infty.$$

Let $v = \alpha(a_1 \cdots a_m)$. Our goal is to show that $(v(\Box + h))^\infty = h$. By assumption on $a_1, \ldots, a_m \in C$, we have $vh = h$. In particular, $v^\omega h = h$. Finally

$$(v(\Box + h))^\infty = (v(\Box + v^\omega h))^\infty \overset{(3)}{=} v^\omega h = h.$$

The last equality is the only time we use identity (3) in this paper. □

D Deciding identities

In this section we show how condition (3) can be decided. Actually, we present a general algorithm, which shows how to decide if any given identity is true in the syntactic algebra of a language L. The algorithm is exponential in the state space of a nondeterministic automaton recognizing L.

What is an identity? This is a pair of terms over the signature of ω-forest algebra, which may contain both forest valued variables and context valued variables. This identity is true in a ω-forest algebra if the two sides of the identity are equal for any valuation into that algebra. When writing an identity, we use h, g, f for the forest valued variables, and v, w, u for the context valued variables.

We are interested in properties of finite algebras. In a finite algebra, the vertical monoid has an idempotent power, i.e. a number ω such that v^ω is idempotent. We assume that the operation $v \mapsto v^\omega$ is part of the signature, and therefore can appear in identities. This is the case for identity (3).

An example identity. On an example, we show that identities for infinite forests can be easily misunderstood. Recall the identity (2):

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

This identity seems to say that the forest language is commutative. But what is a commutative language of infinite forests? There are three possible notions. The first notion of commutativity is that the language is closed under rearranging siblings finitely many times. This notion is satisfied by the (regular) language

K = "Finitely many a-labelled nodes with a b-labelled left sibling."

However, the language K does not satisfy (2), as witnessed by the term

$$(a(\Box + x))^\infty,$$

which gives different results depending on whether x is mapped to $a+b$ or $b+a$. Therefore, the second notion of commutativity is languages that satisfy (2). However, this notion is not the same as a third notion, namely languages that are closed under (possibly infinitely many) rearrangings of siblings, as witnessed by the (regular) language:

L = "On each path, finitely many a-labelled nodes with a b-labelled left sibling."

which satisfies the second notion, but not the third.

Deciding identities in ExpTime.

Theorem D.1
The following problem is ExpTime *complete. The input is a nondeterministic parity forest automaton, and an identity. The question is: is the identity true in the syntactic ω-forest algebra of the language recognized by the automaton?*

It is not difficult to show that the problem is ExpTime-hard, even for some fixed identities. Indeed, suppose that the identity is

$$h = g. \tag{17}$$

When does this identity hold in the syntactic ω-forest algebra of a forest language? There are two possibilities: either the language is empty, or full. The question "is L empty or full?" is ExpTime-hard. This is because the question "is L full?" is ExpTime-hard for nondeterministic tree automata, by reduction from emptiness of alternating polynomial space, and the languages used in the reduction are all nonempty.

How do prove the upper bound? The idea is as follows. First, we show how to calculate, given a nondeterministic automaton $\mathcal{A}$, an ω-forest algebra that recognizes all languages recognized by $\mathcal{A}$. This algebra is called the *automaton algebra*. Next, we show how to minimize any algebra.

D.1 The automaton algebra

??

Let us fix a nondeterministic forest automaton $\mathcal{A}$, with states Q, input alphabet A and with parity ranks $\{0, \ldots, k\}$. Below we describe an ω-forest algebra (H, V), which we call the automaton algebra of $\mathcal{A}$, as well as a morphism

$$\alpha : A^{\triangle} \to (H, V),$$

which recognizes all languages recognized by $\mathcal{A}$.

Before describing the algebra itself, we define the morphism α. This morphism should explain what are the intended meanings of H and V.

- To each forest t, the morphism α associates a subset of Q. A state q belongs to $\alpha(t)$ if some run ρ over t has value q.
- To each context p, the morphism α associates a subset of $Q \times \{0, \ldots, k\} \times Q$. A triple (q_1, i, q_2) belongs to $\alpha(p)$ if there exists a forest s and a run ρ over ps such that ρ evaluates ps to q_2, evaluates s to q_1, and the highest rank assigned to nodes that are ancestors of the hole is i.

Therefore, the carriers of the horizontal and vertical monoids are subsets

$$H \subseteq P(Q), \qquad V \subseteq P(Q \times \{0, \ldots, k\} \times Q),$$

which are images of α on forests and contexts, respectively. These might be proper subsets, for instance not every subset of Q need be an image $\alpha(t)$. A forest belongs to L if and only if its image under α contains an accepting state, so α recognizes L.

The operations in the algebra.

Defining the basic operations of forest algebra is straightforward, keeping in mind the intended meaning of the morphism α. We only do the case of forest concatenation and context composition:

$$\begin{aligned} h + g &= \{p + q : p \in h, q \in g\} && \text{for } h, g \in H, \\ vw &= \{(p, \max(i,j), r) : (p,i,q) \in v, (q,j,r) \in w\} && \text{for } v, w \in V. \end{aligned}$$

More work is needed for unfoldings of recursion schemes. Consider a recursion scheme τ, with variables $Z_H \cup Z_V$. Consider also a valuation

$$\eta : (Z_H, Z_V) \to (H, V).$$

We will show how to define the value $\text{unfold}_\tau[\eta]$, which is a set of states, and also calculate which states it contains. The idea is simple: we evaluate a tree automaton over a regular forest. We take any function

$$\gamma : (Z_H, Z_V) \to A^{\triangle} \tag{18}$$

such that $\alpha \circ \gamma = \eta$. Such a function exists by assumption on α being surjective. Inside the free algebra $A^{\triangle}$, we can evaluate $\text{unfold}_\tau[\gamma]$, which is a regular forest. We then define $\text{unfold}_\tau[\eta]$ to be $\text{unfold}_\tau[\alpha \circ \gamma]$. This definition does not depend on the choice of γ, as shown in the following lemma (we use L_q for the set of forests that can be assigned state q by the automaton $\mathcal{A}$).

Lemma D.2 Let t be a forest over $Z_H \cup Z_V$ and γ a valuation as in (18). For $q \in Q$, membership $t[\gamma] \in L_q$ depends only on $\alpha \circ \gamma$, and not on γ.

We also want to show that the value $\text{unfold}_\tau[\eta]$ can be calculated based on τ and η, in time exponential in the size of the automaton. In other words, we want to find the states q such that $\text{unfold}_\tau[\eta] \in L_q$. Consider first the identity valuation

$$id : (Z_H, Z_V) \to (Z_H, Z_V).$$

Then $\text{unfold}_\tau[id]$ is a regular tree over the alphabet $Z_H \cup Z_H$. Now

$$\text{unfold}_\tau[\gamma] \in L_q \qquad \text{iff} \qquad \text{unfold}_\tau[id] \in \gamma^{-1} L_q.$$

Therefore, we have reduced the problem to testing membership of a regular forest in a regular language (regularity of $\gamma^{-1} L_q$ is witnessed by Lemma D.3). This membership can be tested in time polynomial in the size of τ and exponential in the state space of Q (basically, the problem boils down to solving a parity game).

Lemma D.3 For any language L recognized by $\mathcal{A}$, and any γ as in (18), the language $\gamma^{-1} L$ is recognized by an automaton polynomial in $\mathcal{A}$.

Lemma D.4 The function α preserves all operations of ω-forest algebra. In particular, its image (H, V) is an ω-forest algebra.

Note 4. Suppose that parity games can be solved in polynomial time (an open problem). In this case, the operations in the automaton algebra can be calculated in polynomial time (in the size of Q).

D.2 Minimization of ω-forest algebra

In this section we show how, based on the automaton algebra, we can calculate the syntactic algebra and the syntactic morphism.

Note 5. Even if parity games could be solved in polynomial time, calculating the syntactic algebra would still require exponential time. (One could imagine an algorithm, which is polynomial in Q, that computes the syntactic algebra without explicitly constructing the automaton algebra.) This is because testing the identity $h = g$ in the syntactic algebra can be done in constant time: just check if H has at least two elements.

Thanks to Proposition 3.3, the syntactic morphism α_L is obtained from the automaton morphism α as

$$\alpha^L = \gamma \circ \alpha$$

where γ identifies two elements g, h of the automaton algebra whenever some (equivalently, any) forests $s \in \alpha^{-1}(g)$ and $t \in \alpha^{-1}(h)$ are L-equivalent (likewise for contexts). In this section we show how to decide, in time exponential in Q, which elements of the automaton algebra are identified by γ.

This will complete the proof of Theorem D.1. Recall that we want an exponential time algorithm that decides if an identity holds in the syntactic ω-forest algebra. The algorithm simply tries all out all possible valuations into elements of the automaton algebra, and tests (in at most exponential time) if the equality holds after applying γ.

How to decide which elements of the automaton algebra are identified by γ? We only do the construction for H. For $h, g \in H$, we want to check if there exist forests s, t over A such that

$$\alpha(s) = g, \qquad \alpha(t) = h, \qquad g \sim_L h.$$

By unraveling the definition of L-equivalence, we want to know if there is a forest-valued term ϕ of ω-forest algebra over variables $X_H \cup X_V$ and a valuation

$$\eta : (X_H, X_V) \to A^{\triangle}$$

such that for some forest-valued variable $x \in X_H$,

$$\phi[\eta[x \mapsto s]] \in L \qquad \text{iff} \qquad \phi[\eta[x \mapsto t]] \notin L.$$

This is equivalent to asking if there is a regular forest u over the alphabet $A \cup \{x\}$ such that

$$u[x \mapsto s] \in L \qquad \text{iff} \qquad u[x \mapsto t] \notin L.$$

The above can be rephrased as asking if the two inverse images

$$(x \mapsto s)^{-1}(L) \qquad \text{and} \qquad (x \mapsto t)^{-1}(L),$$

which are regular languages over $A \cup \{x\}$ thanks to Lemma D.3, disagree on some regular forest. Since two different regular languages must necessarily disagree on a regular forest, this boils down to checking inequality of two regular forest languages. This inequality can be decided in time exponential in Q. Note also that, thanks to Lemma D.2, the automata for the inverse images do not depend on the particular choice of trees s, t, but only on their images $\alpha(s) = g, \alpha(t) = h \in H$.

E Checking invariance under EF-bisimulation

In this part of the appendix, we give an algorithm that decides if a language is invariant under EF-bisimulation

Theorem E.1
It is decidable, given an automaton $\mathcal{A}$, if there exist two EF-bisimilar regular trees, of which only one is accepted by $\mathcal{A}$.

Unfortunately, our decision procedure uses more than exponential time. We have tried a number of approaches, but in each case we could not get a singly exponential algorithm. These approaches included:

- Modify the algebra so that invariance under EF-bisimulation can be expressed by identities. Our problem was that the modified automaton algebra became doubly exponential.
- For an automaton $\mathcal{A}$, write an automaton $\mathcal{A}'$ that accepts trees that are EF-bisimilar to some tree recognized by $\mathcal{A}$, and then test for language equivalence (this approach works for normal bisimilarity). The problem was that we could only find doubly exponential constructions for $\mathcal{A}'$.

We leave open the exact complexity of invariance under EF-bisimulation.

In Proposition E.2, we present conditions that are necessary and sufficient for a morphism

$$\alpha : A^{\triangle} \to (H, V),$$

to be invariant under EF-bisimulation. Then, we show how these conditions can be decided for the syntactic morphism of the language recognized by $\mathcal{A}$.

It is not difficult to see that conditions (1) and (2) are necessary for invariance. The following condition is also clearly necessary.

$$(v(\Box + (vw)^{\infty}))^{\infty} = (vw)^{\infty}. \tag{19}$$

These conditions can be decided in ExpTime, thanks to Theorem D.1. Unfortunately, these conditions are not sufficient for invariance. We need to add one more condition, which is not given by an identity. The last condition is stated in terms of multicontexts, which we now define.

Multicontexts. An *n-ary multicontext* over variables $x_1, \ldots, x_n$ is a regular forest over alphabet $A \cup \{x_1, \ldots, x_n\}$ where the variables $x_1, \ldots, x_n$ are allowed only in leaves. We allow multiple (possibly infinitely many) occurrences of each variable. Given forests $s_1, \ldots, s_n$ and an n-ary multicontext p, the forest $p(s_1, \ldots, s_n)$ over A is defined in the natural way. Therefore, an n-ary multicontext is a notation for the unfolding of a recursion scheme τ that has free forest-valued variables $x_1, \ldots, x_n$, and some context-valued variables already bound by a valuation:

$$p(s_1, \ldots, s_n) = \text{unfold}_\tau[\eta[x_1 \mapsto s_1, \ldots, x_n \mapsto s_n]]. \tag{20}$$

An n-ary multicontext is called *prime* if, when treated as a forest over the alphabet $A \cup \{x_1, \ldots, x_n\}$, it is a prime tree, and also all of the non-variable nodes are in the same component.

We say that two multicontexts are EF-bisimilar if they are EF-bisimilar when treated as forests over the alphabet $A \cup \{x_1, \ldots, x_n\}$.

By using the morphism α, an n-ary multicontext naturally induces a function $H^n \to H$. Under the notation from (20), this is the function

$$(\alpha(s_1), \ldots, \alpha(s_n)) \qquad \mapsto \qquad \alpha(\text{unfold}_\tau[\eta[x_1 \mapsto s_1, \ldots, x_n \mapsto s_n]]).$$

Since α is a morphism, the above definition does not depend on the choice of $s_1, \ldots, s_n$.

We are now ready to state the necessary and sufficient conditions for invariance under EF-bisimulation.

Proposition E.2 A morphism

$$\alpha : A^{\triangle} \to (H, V)$$

is invariant under EF-bisimulation if and only if its target algebra satisfies identities (1), (2), (19), and the morphism satisfies the following condition

> *Condition* †. Let $a_1, \ldots, a_n \in A$ and let $x_1, \ldots, x_m$ be multicontext variables. Any multicontext that is EF-bisimilar to
>
> $$p = (a_1 \cdots a_n(\Box + x_1 + \cdots + x_m))^\infty$$
>
> induces the same transformation $H^m \to H$.

We prove the above proposition in Section E.1. Then, in Section E.2, we show how to decide if the syntactic morphism of the language recognized by $\mathcal{A}$ satisfies condition †. This completes the proof of Theorem E.1, since identities (1), (2), (19) can be decided thanks to Theorem D.1.

E.1 Proof of Proposition E.2.

The "only if" part of the proof is quite obvious. The rest of this section is devoted to the "if" part.

We want to show that if forests s and t are EF-bisimilar, then they have the same types. The proof is by induction on the number of components in s plus the number of components in t.

Fact E.3 Without loss of generality, we can assume that s and t are trees.

Proof
By using identities (1) and (2). □

The induction base is when both s and t have a single component. If s is finite, then it has a single node a. In this case t also has to be a, since this is the only tree that is EF-bisimilar to a. Suppose now that s and t are infinite. Let $a_1, \ldots, a_n$ be the labels that appear in s (and therefore also in t). It is easy to see that s and t are EF-bisimilar to $u = (a_1 \cdots a_n \Box)^\infty$. All of s, t, u can be treated as prime multicontexts of arity 0. By condition †, both s and t have the same type.

We now do the induction step. Let $s_1, \ldots, s_n$ be all the subtrees of s that have fewer components than s. In other words, there is a prime n-ary multicontext p such that

$$s = p(s_1, \ldots, s_n).$$

Likewise, we distinguish all subtrees $t_1, \ldots, t_k$ inside t that have fewer components than t, and find a prime k-ary multicontext q with

$$t = q(t_1, \ldots, t_k).$$

Since s and t were EF-bisimilar, each tree s_i must be EF-bisimilar to some subtree $\hat{s}_i$ of t. By induction assumption, we know that the trees s_i and $\hat{s}_i$ have the same type. Likewise, each tree t_i has the same type as some subtree $\hat{t}_i$ of s.

By applying (1), we conclude that if either p or q is finite then s and t have the same type. We are left with the case when both p and q are infinite prime contexts. Suppose that

1. For some i, the tree $\hat{s}_i$ has the same number of components as t; and
2. For some j, the tree $\hat{t}_j$ has the same number of components as s.

We use the same notion of reachability on types as was used in Lemma 4.5. From 1 we conclude that the tree $\hat{s}_i$ is in the root component of t, and therefore the type of both s_i and $\hat{s}_i$ is reachable from the type of t. Since s_i is a subtree of s, conclude that the type of s is reachable from the type of t. Reasoning in the same way from 2 we conclude that type of t is reachable from the type of s. Therefore, by Lemma 4.5, the types of s and t are equal (note that Lemma 4.5 used (1)).

Suppose now that one of 1 or 2 does not hold, say 1 does not hold.

Lemma E.4 Without loss of generality, we can assume that $n \leq k$ and

$$s = p(t_1, \ldots, t_n).$$

Proof
Consider the tree $\hat{s} = p(\hat{s}_1, \ldots, \hat{s}_n)$. Since we replaced trees with EF-bisimilar ones, $\hat{s}$ is bisimilar to s. Since we replaced trees with ones of the same type, $\hat{s}$ has the same type as s. So it is enough to prove the result for $\hat{s}$ and t. After possibly renaming variables in p, $\hat{s}$ has the form in the statement of the lemma. □

What about the trees $t_{n+1}, \ldots, t_k$ that do not appear in s? Each of these is EF-bisimilar to one of $s, t_1, \ldots, t_n$. For those that are EF-bisimilar to some $t_i \in \{t_1, \ldots, t_n\}$, we use the tree instead. Therefore, we can without loss of generality assume that

$$t = q(s, t_1, \ldots, t_n).$$

Lemma E.5 Any label $a \in A$ that appears in q also appears in p.

Proof
Let $a \in A$ be a label in q and consider the following strategy for Spoiler in the game over the trees s and t: he picks t and in that tree, some occurrence of a in the root component. Duplicator, in his response, cannot pick a node inside any of the trees $t_1, \ldots, t_n$, since none of these is EF-bisimilar to a tree in the root component of t. Therefore, he must pick a node inside p. □

Let $a_1, \ldots, a_i$ be the labels that appear in q. Thanks to the above lemma, the labels that appear in p are $a_1, \ldots, a_i$ as well as possibly some other labels, say $a_{i+1}, \ldots, a_j$ for some $j \geq i$. Let us define the following two contexts

$$x = (a_1 \cdots a_i(\Box + t_1 + \cdots + t_n))^\infty, \qquad y = (a_{i+1} \cdots a_j \Box).$$

By using †, it is not difficult to show that type of s is the same as the type of $(xy)^\infty$. Again using †, one shows that the type of t is the same as the type of $(x(\Box + s))^\infty$, which is the same as the type of $(x(\Box + (xy)^\infty))^\infty$. Therefore, we conclude that the types of s and t are equal by condition (19).

E.2 Deciding condition †.

Let us fix an automaton $\mathcal{A}$, with states Q and input alphabet A. There are two morphisms that are associated with $\mathcal{A}$.

The first morphism is the automaton morphism

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

as defined in Appendix D. Recall that elements of H are subsets of Q.

The second morphism is the syntactic morphism

$$\alpha_L : A^\triangle \to (H_L, V_L) \ .$$

We will say *type of* t for the value $\alpha_L(t)$. Our goal in this section is to decide if α_L satisfies condition †.

A *counterexample* to † is a sequence $a_1, \ldots, a_n$, types $h_1, \ldots, h_m \in H$ and a multicontext q that is EF-bisimilar to p such that the types

$$p(h_1, \ldots, h_m), \qquad q(h_1, \ldots, h_m).$$

are different under the syntactic congruence in (H, V) induced by the language recognized by $\mathcal{A}$. We can assume that $h_1, \ldots, h_m$ are all distinct, since it does not make sense to use the same type under different variables. We also can assume that $a_1, \ldots, a_n$ are all distinct (otherwise, instead of q we could use the multicontext obtained from p by removing duplicates from $a_1 \cdots a_n$).

We try to find a counterexample for each possible choice of $a_1, \ldots, a_n$ and $h_1, \ldots, h_m$. Enumerating all possible sequences $h_1, \ldots, h_m$ is one of the reason why this method is doubly exponential, since the size of H is exponential in Q.

To simplify the proof, we assume that for each type $h \in H$ there is a label a_h such that the tree a_h has type h. Therefore, searching for a counterexample comes down to checking if there is some forest that is EF-bisimilar to

$$t = (a_1 \ldots a_n(\Box + a_{h_1} + \cdots + a_{h_n}))^\infty.$$

but has a different type than t, under the syntactic congruence. Let L_t be the set of forests that are EF-bisimilar to t. We want to know if all forests from L_t are syntactically equivalent to t. This boils down to testing inclusion of two regular languages, since the language L_t is regular (and even definable in EF), by the following lemma.

Lemma E.6 Let t be a regular tree. The set of trees that are EF-bisimilar to t is definable by an EF formula φ_t.

Proof
The proof is by induction on the number of components in t. Let $t_1, \ldots, t_k$ be the subtrees of t that have fewer components than t. By induction assumption, we already have languages $\varphi_{t_1}, \ldots, \varphi_{t_k}$.

Suppose first that the root of t is in a singleton component, i.e. all proper subtrees of t are among $t_1, \ldots, t_k$. Let a be the root label of t. The following formula defines L_t:

$$a \qquad \wedge \qquad \mathsf{AG}^*\Big(\bigvee_{i \leq k} \varphi_{t_i}\Big) \qquad \wedge \qquad \bigwedge_{i \leq k} \mathsf{EF}\varphi_{t_i}.$$

Suppose now that the root of t is in a connected component. Let $a_1, \ldots, a_n$ be the labels that appear in the root component of t. The formula for L_t is

$$\neg \bigvee_{i \leq k} \varphi_{t_i} \qquad \wedge \qquad \mathsf{AG}^*\Big(\Big(\neg \bigvee_{i \leq k} \varphi_{t_i}\Big) \to \Big(\bigwedge_{i \leq k} \mathsf{EF}\varphi_{t_i} \wedge \bigvee_{i \leq n} a_i\Big)\Big).$$

□