Algebra for Trees

Mikołaj Bojańczyk[*]

University of Warsaw
email: bojan@mimuw.edu.pl

2010 Mathematics Subject Classification: 68Q70

Key words: Tree automata, Forest algebra

Abstract. This chapter presents several algebraic approaches to tree languages. The idea is to design a notion for trees that resembles semigroups or monoids for words. The focus is on the connection between the structure of an algebra recognizing a tree language, and the kind of logic needed to define the tree language. Four algebraic approaches are described in this chapter: trees as terms of universal algebra, preclones, forest algebra, and seminearrings. Each approach is illustrated with an application to logic on trees.

1 Introduction

This chapter presents several algebraic approaches to regular tree languages. [1] An algebra is a powerful tool for studying the structure of a regular tree language, often more powerful than tree automata. This makes algebra a natural choice for proving lower bounds. Another area which uses algebra a lot is the search for effective characterizations. Since this is the guiding motivation for this chapter, we begin with a discussion on effective characterizations.

Effective characterizations. Let $\mathscr{L}$ be a class of regular languages (for words or trees). We say that $\mathscr{L}$ has an *effective characterization* if there is an algorithm, which inputs a representation of a regular language, and says if the language belongs to $\mathscr{L}$. We are mainly interested in decidability, so we do not pay too much attention to the format in which the input language is represented, it could be e.g. an automaton or a formula of monadic second-order logic.

On the surface, finding an effective characterization looks like a strange problem. Its

[*] Author supported by ERC Starting Grant "Sosna"

[1] I would like to thank my colleagues for their helpful comments, especially Tomasz Idziaszek, Luc Segoufin, Howard Straubing, Igor Walukiewicz, Pascal Weil and Thomas Wilke.

practical applications seem limited. And yet for the last several decades, people have intensively searched for effective characterizations of language classes, originally for word languages, and recently also for tree languages. Why? The reason is that each time someone proved an effective characterization of some class of languages, the proof would be accompanied by a deeper insight into the structure of the class. One can say that "give an effective characterization" is a synonym for "understand". In this sense, we have still not understood numerous tree logics, including first-order logic with descendant, chain logic, PDL, CTL* and CTL.

A classical example is first-order logic for words. It is one thing to prove that first-order logic cannot define some language, such as $(aa)^*$. It is another thing to prove the theorem of Schützenberger-McNaughton-Papert, which says that a word language can be defined in first-order logic if and only if its syntactic monoid is group-free. The theorem does not just give one example of a language the cannot be defined in first-order logic, but it describes *all* such examples, in this case the languages whose syntactic monoids contain a group. Also, the theorem establishes a beautiful connection between formal language theory and algebra.

In recent years, many researchers have tried to extend effective characterizations from words languages to tree languages. This chapter describes some of the known effective characterizations for tree languages, and it gives references to the others. Nevertheless, as mentioned above, many important logics for trees are missing effective characterizations.

Of course, algebra has played an important role in the research on tree languages. The goal of this chapter is to describe the algebraic structures that have been developed. However, part of the focus is always on the applications to logic. Therefore, this chapter is as much about logics for trees as it is about algebras for trees. This chapter is exclusively about finite trees.

For word languages, the algebraic approach is to use monoids or semigroups. For tree languages, there is much more diversity. This chapter describes four different algebraic approaches, and gives a recipe to define any number of new ones. One reason for this diversity is that trees are more complicated than words, and there are more parameters to control, such as: are trees ranked, or unranked? are trees sibling-ordered or not? Another reason is that the algebraic theory of tree languages is in a state of flux. We still do not know which of the competing algebraic approaches will prove more successful in the long run. Instead of trying to choose the right algebra, we take a more pragmatic approach. Every algebraic approach is illustrated with some results on tree logics that can be proved using the approach, preferably examples which would require more work using the other approaches.

There is always the question: what is an algebra? What is the difference between an algebra and an automaton? Two differences are mentioned below, but we make no attempt to answer this interesting question definitively.

The first difference is that, simply put, an algebra has more elements than an automaton. For instance, in the word case, a deterministic automaton assigns meaningful information to every prefix of a word, while a homomorphism into a finite monoid assigns meaningful information to every infix. This richer structure is often technically useful. For instance, in monoids, on can define a transitive relation on elements, which considers an infix s to be simpler than its extension ust. This relation is used as an induction parameter in numerous proofs about monoids. The relation does not make sense for

automata.

The second difference is that in algebra, unlike in automata, the recognizing device is usually split into two parts: a homomorphism and a target algebra. Consider, as an example, the case of words and monoids. If we just know the target monoid and not the homomorphism, it is impossible to tell if the identity of the monoid is the image of only the empty word, or also of some other words. For reasons that are unclear, this loss of information seems to actually make proofs easier and not harder. At any rate, separating the recognizing device into two parts is a method of abstraction that is distinctive of the algebraic approach.

Potthof example. Before we begin our discussion of the various algebras, we present a beautiful example due to Andreas Potthoff, see Lemma 5.1.8 in [23]. This example shows how intuitions gained from studying words can fail for trees.

Consider an alphabet $\{a, b\}$. This alphabet is ranked: we only consider trees where nodes with letter a have two children, and nodes with letter b are leaves. Let P be the set of such trees where every leaf is at even depth. For instance, $b \notin P$ and $a(b, b) \in P$. Intuition suggests that P cannot be defined in first-order logic, for the same reasons that first-order logic on words cannot define $(aa)^*$. If we consider first-order logic with the descendant predicate, then this intuition is correct. Consider the balanced tree t_n of depth n, defined by $t_0 = b$ and $t_{n+1} = a(t_n, t_n)$. An Ehrenfeucht-Fraissé argument shows that every formula of size n will give the same results for all balanced trees of depth greater than 2^n.

What if, apart from the descendant order, we also allow formulas to use sibling order? Sibling order is the relation $x \preceq y$ which holds when x is a sibling of y, and x is to the left of y. For the alphabet in question sibling order could be replaced by a unary "left child" predicate, but we use sibling order, since it works well for unranked trees.

We first show that a formula with descendant and sibling orders can distinguish binary complete trees of even and odd depth. The idea is to look at the *zigzag path*, which begins in the root, goes to the left child, then the right child, then the left child and so on until it reaches a leaf. We say a tree satisfies the *zigzag property* if the zigzag path has even length, which is the same as saying that the unique leaf on the zigzag path is a left child. Consequently, a balanced binary tree of depth n satisfies the zigzag property if and only if n is even. The zigzag property can be defined in first-order logic, using the descendant and sibling orders: one says that there exists a leaf x which is a left child, and such that for every ancestor y of x that has parent z, one of the nodes y, z is a left child, and the other is a right child or the root.

What is more, the zigzag property can be used to actually define the language P. A tree *does not* have all leaves at even depth if and only if either: a) the zigzag property is not satisfied; or b) for some two siblings x, y the zigzag property is satisfied by the subtree of x, but not the subtree of y. These conditions can be formalized in first-order logic; and therefore the language P can be defined in first-order logic with descendant and sibling orders.

This example can be used to disprove some intuitions. For instance, the language P is order invariant (i.e. invariant under swapping sibling subtrees). It follows that first-order logic with descendant order only is strictly weaker than order invariant first-order logic with descendant and sibling orders.

2 Trees as ground terms

The first type of algebra that we talk about works for ranked trees. Ranked trees are built using a ranked alphabet, where each letter is assigned a number, called the letter's *arity*. A tree over a ranked alphabet is a tree where the number of children of each node is the same as the arity of its label. We write t, s for trees. In particular leaves are letters of arity zero, also called nullary letters. Since we are considering finite trees, it only makes sense to consider alphabets with at least one nullary letter.

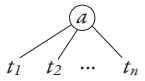
The algebraic approach is to see trees as terms, in an algebra whose signature is given by the ranked alphabet. (More exactly, trees are ground terms, i.e. terms that do not use any variables.) The free algebra corresponds to the set of all trees. A finite algebra corresponds to a (deterministic bottom-up) tree automaton, the domain of the algebra is the state space. The original paper [27] on regular tree languages by Thatcher and Wright talks about trees and tree automata this way.

Here is the setup. A ranked alphabet is treated as a signature in the sense of universal algebra. Each letter of the ranked alphabet is a function symbol of the same arity. In particular, the nullary letters, or leaves, are constants. We study algebras over this signature. We call them *A-algebras* when the alphabet is A. Following universal algebra, an A-algebra $\mathcal{A}$ is defined by giving a domain H, and an interpretation of each n-ary letter a in the ranked alphabet as a function $a^{\mathcal{A}} : H^n \to H$.

A *morphism* from one A-algebra to another is a mapping from the domain of the first algebra to the domain of the second algebra that preserves the operations in the usual sense. We are only interested in algebras that are accessible, which means that every element of the domain can be obtained by evaluating some expression built out of constants and function symbols. This makes morphisms uninteresting, since there is exactly one morphism between any two accessible A-algebras.

We use two types of A-algebra: the free algebra, and algebras with finite domain. The free algebra will correspond to trees, and the finite algebras will correspond to automata.

The free algebra. The domain of the *free A-algebra* is the set of all trees over the ranked alphabet A, which we denote $trees(A)$. Each n-ary letter a is interpreted as an n-ary operation which takes trees $t_1, \ldots, t_n$ and returns the tree $a(t_1, \ldots, t_n)$ shown below.



This algebra is free in the following sense. For every A-algebra $\mathcal{A}$, there is a unique morphism α from the algebra to $\mathcal{A}$. If $\mathcal{A}$ is an A-algebra and t is a tree (an element of the free A-algebra), we write $t^{\mathcal{A}}$ for the image $\alpha(t)$ under this unique morphism. (Unlike for monoids or semigroups, the alphabet is interpreted in the signature, and not as generators. In this sense, the set of generators for the free A-algebra is empty, since the trees are built out of constants, or nullary letters. This will change for the other algebras in this chapter.)

Recognizing languages. A tree language L over alphabet A is said to be *recognized* by an algebra $\mathcal{A}$ if membership $t \in L$ depends only on $t^{\mathcal{A}}$. When $\mathcal{A}$ is finite, it can be viewed as a deterministic bottom-up tree automaton: the state $t^{\mathcal{A}}$ assigned to a tree depends only

on the root label of t and the states assigned to the children. Consequently, a tree language over a ranked alphabet is regular if and only if it is recognized by some finite algebra.

Syntactic algebra. We now define the syntactic algebra of a tree language, which plays the same role as the syntactic (or minimal) deterministic automaton for a word language. The definition uses a Myhill-Nerode style congruence, which talks about putting trees in different contexts. Here, a context over alphabet A is defined as a tree over an extended alphabet $A \cup \{\Box\}$, which includes an additional nullary hole symbol $\Box$. A context must use the hole symbol exactly once. The hole plays the role of a variable, but we use the name hole and the symbol $\Box$ for consistency with the other parts of this chapter. We write p, q, r for contexts. If p is a context and s is a tree, we write ps for the tree obtained by replacing the hole of p by s.

We now define the syntactic A-algebra of a tree language L over an alphabet A. A non-regular language also has a syntactic A-algebra, but it is infinite. We say that two trees s and t are L-equivalent if there is no context that distinguishes them, i.e. no context p such that exactly one of the trees ps and pt is in L. This equivalence relation is a congruence in the free A-algebra, so it makes sense to consider the quotient of the free A-algebra with respect to L-congruence. This quotient is called the *syntactic A-algebra of L*. One can show that A-algebra is a morphic image of any other A-algebra that recognizes L. A consequence is that a tree language is regular if and only if its syntactic A-algebra is finite.

Limitations of A-algebras. An advantage of the approach described above is that it uses very simple concepts to describe regular tree languages. Arguably, the definition is simpler than the algebraic approach to word languages via semigroups. But is it fair to use the name algebra for an A-algebra? Or is it an automaton? Below we describe some benefits from using algebra (semigroups and monoids) in the word case which do not work well for A-algebras.

An important theme in the algebraic approach to word languages is that properties of word languages, such as "the word language can be defined in first-order logic", correspond to properties of syntactic semigroups, such as "the semigroup is group-free". Also, important properties of semigroups can be stated by identities, such as the identity

$$s^\omega = s^{\omega+1},$$

which says that a semigroup is group-free, or the identities

$$st = ts \qquad \text{and} \qquad ss = s,$$

which say that a semigroup is commutative and idempotent. Unfortunately, we run into problems when we try to do this for A-algebras.

The first problem is that the signature (i.e. the set of operations) in an A-algebra depends on the ranked alphabet A. Suppose that we want to talk about the class of tree languages that are invariant under reordering siblings. This property can be expressed using identities, but in a cumbersome way: for each letter a of the alphabet, of arity n, we need identities which imply that the children can be reordered, e.g.

$$a(x_1, \ldots, x_n) = a(x_j, x_2, \ldots, x_{j-1}, x_1, x_{j+1}, \ldots, x_{n-1}, x_n) \qquad \text{for } j = 2, \ldots, n.$$

A second, and more important, problem is that the set of objects is not rich enough. Consider the group-free identity $s^\omega = s^{\omega+1}$. What makes this identity so powerful is that it says that the left side can be replaced by the right side in any environment. In terms of words, this means that for sufficiently large n, an infix w^n can be replaced by an infix w^{n+1}. In A-algebras, elements of the algebra correspond to subtrees, and any identity will only allow to replace one subtree with another. For words, this would be like using identities to describe suffixes, and not infixes. This means that very few important properties of tree languages can be described using identities in A-algebras.

Terms. As we remarked above, talking about trees and subtrees may be insufficient. Sometimes, we want to talk about contexts, or contexts with several holes, which we call multicontexts. Formally speaking, a *multicontext* over alphabet A is a tree over alphabet $A \cup \{\Box\}$, where $\Box$ is a nullary letter. In a multicontext, there is no restriction on the number of times (possibly zero) the hole symbol $\Box$ is used, this number is called the arity of the multicontext. We number the holes from left to right, beginning with 1 and ending with the arity.

There are two kinds of substitution for multicontexts. Suppose that p is an n-ary multicontext, and q is an m-ary multicontext. The first kind of substitution places q in one hole. For any $i \in \{1, \ldots, n\}$, we can replace the i-th hole of p by q, the resulting multicontext is denoted $p \cdot_i q$, and its arity is $n + m - 1$. The second kind of substitution places q in all holes simultaneously; the resulting multicontext is denoted $p \cdot q$, and its arity is $m \cdot n$. When talking about ranked trees, we will only use the first kind of substitution.

Suppose that $\mathcal{A}$ is an A-algebra, with domain H. Every k-ary multicontext p over alphabet A can be interpreted, in a natural way, as a function

$$p^{\mathcal{A}} : H^k \to H.$$

For technical reasons, we assume that p is not the empty multicontext $\Box$. We use the name k-ary *$\mathcal{A}$-term* for any such function. Nullary $\mathcal{A}$-terms can be identified with the domain H. When $\mathcal{A}$ is the free A-algebra, $\mathcal{A}$-terms can be identified with the set of k-ary multicontexts over A. There is a natural definition of substitution for $\mathcal{A}$-terms which mirrors substitution on multicontexts, defined by

$$p^{\mathcal{A}} \cdot_i q^{\mathcal{A}} = (p \cdot_i q)^{\mathcal{A}}.$$

2.1 Definite Languages

To illustrate the power of A-algebras, but also the difficulties of trees, we will use A-algebras to study definite languages. This is a class of languages, which in the case of words has a simple and elegant algebraic characterization.

Definite word languages. A word language L is called *definite* if there is a threshold $n \in \mathbb{N}$, such that membership $w \in L$ depends only on the first n letters of w. Stated differently, a definite word language over alphabet A is a finite boolean combination of languages of the form wA^* for $w \in A^*$.

There is simple algebraic characterization of definite word languages. Suppose that

$L \subseteq A^*$ is a word language, whose syntactic semigroup morphism is $\alpha : A^+ \to S$. Then L is definite if and only if the identity

$$s^\omega t = s^\omega u \tag{2.1}$$

holds for any two elements $s, t, u \in S$. The idea is that s^ω represents a long word, and anything after it is not important[2]. We prove this characterization below.

We say that elements s, t of the syntactic semigroup S *have arbitrarily long common prefixes* if for any $n \in \mathbb{N}$, there are words $w, v \in A^+$, which are mapped by α to s, t respectively, and which have the same labels up to position n. This can be stated without referring to the morphism α as

$$\forall n \in \mathbb{N} \quad \exists u_1, \ldots, u_n \in S \qquad s, t \in u_1 \cdots u_n S.$$

It is easy to see that a language is definite if and only if any two elements of S that have arbitrarily long common prefixes are equal. We now explain how the latter property is captured by the identity (2.1). If n is sufficiently large, then a Ramsey argument can be used to show that for any element $u_1, \ldots, u_n$, there exist $1 < i < j < n \in \{1, \ldots, n\}$ such that $u_i \cdots u_j$ is idempotent. Therefore, a condition necessary for s, t having arbitrarily long common prefixes is

$$s, t \in xy^\omega z S \qquad \text{for some } x, y, z \in S. \tag{2.2}$$

It suffices to take $x = u_1 \cdots u_{i-1}$, $y = u_i \cdots u_j$ and $z = u_{j+1} \cdots u_n$. It is not difficult to see that the above condition is also sufficient, since y^ω is of the form $u_1 \cdots u_n$ for arbitrarily large n, e.g. by taking $u_1 = \cdots = u_n = y$. It follows that L is definite if and only if its syntactic semigroup S satisfies the identity

$$xy^\omega z s' = xy^\omega z t'.$$

One can show that the above identity is equivalent to (2.1).

Definite tree languages. We now try to generalize the ideas above from words to trees. As long as we are only interested in testing if a tree language is definite, then the above approach works. On the other hand, if we want to know which trees have arbitrarily long prefixes, maybe in a language that is not definite, then the above approach stops working. We explain this in more detail below.

Consider an A-algebra $\mathcal{A}$. We say that $g, h \in \mathcal{A}$ *have arbitrarily deep common prefixes* if for any $n \in \mathbb{N}$, there are trees s, t from the free A-algebra, with $t^{\mathcal{A}} = g$ and $s^{\mathcal{A}} = h$, which have the same nodes and labels up to depth n.

We would like to give an alternative definition, which does not mention trees, which are elements of the infinite free A-algebra. Preferably, the alternative definition would give an effective criterion to decide which elements have arbitrarily deep common prefixes. A simple tree analogue of (2.2) would be

$$g, h \in xy^\omega z\mathcal{A} = \{(xy^\omega z)(f) : f \in \mathcal{A}\} \qquad \text{for some unary } \mathcal{A}\text{-terms } x, y, z. \tag{2.3}$$

[2]It is important that α is the semigroup morphism, which represents nonempty words, and not the syntactic monoid morphism, which also represents the empty word. Otherwise, we would have to restrict the identity (2.1) so that s represents at least one nonempty word.

(In the notation $xy^\omega z$ we treat unary $\mathcal{A}$-terms as elements of a finite semigroup.) Unfortunately, the condition above is not the same as saying that g, h have arbitrarily deep common prefixes. The condition is sufficient (for sufficiency, it is important that our definition of $\mathcal{A}$-term does not allow the empty context $\Box$), but not necessary, as demonstrated by the following example, which is due to Igor Walukiewicz.

Example 2.1. The alphabet has a binary letter a and nullary letters b, c. Consider the language "all leaves have the same label", and its syntactic algebra, which has three elements:

$$h_b = \text{all leaves have label } b, \qquad h_c = \text{all leaves have label } c, \qquad \bot = \text{ the rest.}$$

All three elements of the syntactic algebra have arbitrarily deep common prefixes, but the elements h_b and h_c cannot be presented as

$$h_b = xy^\omega z g_b, \qquad h_c = xy^\omega z g_c$$

for any choice of g_b, g_c. The only possible choice would be $g_b = h_b$ and $g_c = h_c$. The problem is that each context x, y, z comes with its own leaves (recall that the symbol a is binary). The first equality requires the contexts x, y, z to have all leaves with label b, and the second equality requires all leaves to have label c.

Tree prefix game. The above example indicates that idempotents in the semigroup of unary $\mathcal{A}$-terms are not the right tool to determine which trees have arbitrarily deep common prefixes. Then what is the right tool?

We propose a game, called the *tree prefix game*. The game is played by two players, Spoiler and Duplicator. It is played in rounds, and may have infinite duration. At the beginning of each round, there are two elements f_1, f_2 of the algebra, which are initially $f_1 = g$ and $f_2 = h$. A round is played as follows. First player Duplicator chooses a letter a of the alphabet, say of arity n, and $2n$ elements of the algebra

$$f_{11}, \ldots, f_{1n}, \quad f_{21}, \ldots, f_{2n} \qquad \text{with } f_1 = a^{\mathcal{A}}(f_{11}, \ldots, f_{1n}) \text{ and } f_2 = a^{\mathcal{A}}(f_{21}, \ldots, f_{2n}).$$

If Duplicator cannot find such elements the game is terminated, and Spoiler wins. Otherwise, Spoiler chooses some $i \in \{1, \ldots, n\}$ and the game proceeds to the next round, with the elements f_{1i} and f_{2i}. If $n = 0$ (which implies $f_1 = f_2$), then the game is terminated, and Duplicator wins. If the game continues forever, then Duplicator wins.

Theorem 2.1. *Two elements of an algebra have arbitrarily deep prefixes if and only if Duplicator wins the tree prefix game.*

Note that the above theorem gives a polynomial time algorithm to decide if two elements of a finite algebra have arbitrarily deep common prefixes, since the tree prefix game is a safety game with a polynomial size arena, which can be solved in polynomial time.

Definite tree languages, again. We have seen before that idempotents are not the right tool to describe trees with arbitrarily deep common prefixes. The solution we proposed was the tree prefix game. This game provides an algorithm that decides if a tree language is definite: try every pair of distinct elements in the syntactic algebra, and see if Duplicator

can win the game. If there is a pair where Duplicator wins, then the language is not definite. If there is no such pair, then the language is definite.

However, if we are only interested in arbitrarily deep common prefixes as a tool to check if a tree language is definite, then idempotents are enough, as shown by the following theorem. (The language in Example 2.1 is not definite.)

Theorem 2.2. *Let L be a tree language whose syntactic algebra is $\mathcal{A}$. Then L is definite if and only if every unary $\mathcal{A}$-term u and every elements $f, g \in \mathcal{A}$ satisfy*

$$u^\omega f = u^\omega g.$$

Proof. It is easy to see the "only if" direction. We prove the "if" direction.

Let α be the syntactic morphism. Since unary $\mathcal{A}$-terms form a finite semigroup, there must be a number n such that for any unary $\mathcal{A}$-terms $u_1, \ldots, u_n$, there is a decomposition $u_1 \cdots u_n = xy^\omega z$ for some unary $\mathcal{A}$-terms x, y, z. Let a be some nullary (leaf) letter in the alphabet. We claim that every tree s over alphabet A has the same image under α as the tree $\hat{s}$ obtained from s by replacing all nodes at depth n by a leaf with label a. This implies that L is definite, since α gives the same result for any two trees that have the same nodes up to depth n.

Consider then a tree s. We prove the claim that $\alpha(s) = \alpha(\hat{s})$ by induction on the number of nodes in s that have depth n and are not a leaf with label a. The base case, when all nodes at depth n have label a, is immediate, since it implies $s = \hat{s}$. Consider the induction step. Let v be a node at depth n inside s that is not a leaf with label a. Let p be the context obtained from s by putting the hole in v, and let t be the subtree of v; we have $s = pt$. By choice of v and n, there exist unary $\mathcal{A}$-terms x, y, z with $\alpha(p) = xy^\omega z$. We use the identity from the statement of the theorem to prove that pt and pa have the same image under α:

$$\alpha(pt) = \alpha(p)\alpha(t) = xy^\omega z\alpha(t) = xy^\omega z\alpha(a) = \alpha(p)\alpha(a) = \alpha(pa).$$

The tree pa has more nodes at depth n with label a than the tree s, so we can use the induction assumption to conclude that pa has the same image under α as $\widehat{pa} = \hat{s}$. □

2.2 First-order logic with child relations

In this section, we state one of the more advanced results connecting logic and algebra. The result talks about a variant of first-order logic that is allowed to use the child predicate, but not the descendant predicate.

Fix a ranked alphabet A. We now define a logic that is used to describe trees over alphabet A. For each label $a \in A$, there is a unary predicate $a(x)$, which says that node x has label a. Let n be the maximal arity of a symbol from A. For any $i \in \{1, \ldots, n\}$ we have a predicate $child_i(x, y)$, which says that y is the i-th child of x. Importantly, we *do not* have a predicate $x \leqslant y$ for the descendant relation. In this section, we talk about first-order logic with these predicates, which we call *first-order logic with child relations*.

Which tree languages can be defined in first-order logic with child relations? We begin with the straightforward observation that the logic can only define "local" properties.

Then we state the main result, Theorem 2.3, which characterizes the logic in terms of two identities.

Suppose that p is a multicontext of arity k. We say that p appears in node x of a tree t, if the subtree of t in node x can be decomposed as $p(t_1, \ldots, t_k)$ for some trees $t_1, \ldots, t_k$. A *local formula* is a statement of the form "multicontext p appears in at least m nodes of the tree", or a statement of the form "multicontext p appears in the root of the tree". Of course every local formula can be expressed in first-order logic with child relations. The Hanf locality theorem gives the converse: any formula of first-orer logic with child relations is equivalent, over trees, to a boolean combination of local formulas.

This normal form using local formulas explains what can and what cannot be expressed in first-order logic with child relations. We give an illustration below.

Example 2.2. The alphabet has a binary letter a, a unary letter b and nullary letters c, d. Consider the language L that consists of trees where the root has label a, and some descendant of the root's left child has label c. We will show that L cannot be described by a boolean combination of local formulas, and therefore L cannot be defined in first-order logic with child relations. For $n \in \mathbb{N}$, consider the two trees

$$a(b^n(c), b^n(d)) \in L \qquad \text{and} \qquad a(b^n(d), b^n(c)) \notin L\ .$$

Consider any multicontext p. If all holes in p are at depth at most n, then p appears in the same number of nodes in both trees above. Consequently, the two trees cannot be distinguished by any local formula that uses a multicontext with all holes at depth at most n. It follows that any boolean combination of local formulas will confuse the two trees, for sufficiently large n.

In the example above, any local formula would be confused by swapping two subtrees $b^n(c)$ and $b^n(d)$ for sufficiently large n. The reason is that the two subtrees agree on nodes up to depth n. This leads us back to the notion of trees that have arbitrarily deep common prefixes, which was discussed in Section 2.1. This notion will be key to the following Theorem 2.3, which characterizes the tree languages that can be defined in first-order logic with child relations.

To state the theorem, we extend the notion of having arbitrarily deep common prefixes from elements of $\mathcal{A}$ to $\mathcal{A}$-terms. We say two $\mathcal{A}$-terms u, v have arbitrarily deep common prefixes if for any $n \in \mathbb{N}$, one can find multicontexts p, q that have the same nodes and labels up to depth n, and such that $u = p^{\mathcal{A}}$ and $v = q^{\mathcal{A}}$.

Theorem 2.3. *A tree language is definable in first-order logic with child relations if and only if its syntactic algebra $\mathcal{A}$ satisfies the following two conditions.*

- *Vertical swap. Suppose that u_1, u_2 are unary $\mathcal{A}$-terms with arbitrarily deep common prefixes, likewise for v_1, v_2. Then*

$$v_1 u_1 v_2 u_2 = v_2 u_1 v_1 u_2.$$

- *Horizontal swap. Suppose that $h_1, h_2 \in \mathcal{A}$ have arbitrarily deep common prefixes, and w is a binary $\mathcal{A}$-term. Then*

$$w(h_1, h_2) = w(h_2, h_1).$$

References. The algebraic approach presented in this section dates from the first paper on regular tree languages [27]. Variety theory for tree languages seen as term algebras was developed in [26]. Theorem 2.2, one of the first effective characterizations of logics for trees, was first proved in [15]. The idea to study languages via the monoid of contexts is from [28]. The generalization of classical concepts, such as aperiodicity, star-freeness, and definability in logics such as first-order logic, chain logic or anti-chain logic was studied [28, 16, 17, 25, 24] Theorem 2.3 was proved in [1]. Effective characterizations for some temporal logics on ranked trees were given in [9, 13, 21], while [22] gives an effective characterization of locally testable tree languages.

3 A recipe for designing an algebra

Here are some disadvantages of the A-algebras discussed in Section 2.

- The principal disadvantage is that the set of objects described by an A-algebra, namely trees, is not rich enough. Almost any nontrivial analysis of a tree language requires talking about contexts (terms with one hole), or even terms with more than one hole. Why not make these part of the algebra?
- The set of operations depends on the ranked alphabet A. One consequence of this is that one cannot define any class of tree languages by a single set of identities; since identities need to refer to a common set of operations. Of course a quick fix is to give separate identities for each alphabet.
- The trees are ranked. Unranked trees, where there is no limit on the number of children of a node, are important in computer science, especially in XML. In the unranked case, an alphabet provides just names for the letters, without specifying their arities.
- From the point of view of many logics, fixing the number of children for each node is artificial. Consider modal logic, which accesses the tree structure via operators "in some child φ" and "in all children φ". A property of trees defined in modal logic should be closed under reordering and duplicating children. Reordering is not a problem, but duplicating is disallowed by the syntax of ranked trees.

In the rest of this chapter, we present some algebras that try to solve these problems. However, a problem with designing an algebra for trees is that there are so many parameters to control. Are the trees ranked or unranked? Does the algebra represent only trees? Or does it also represent contexts? Or maybe also terms of arbitrary arity? Is it legal for a context to have the hole in the root? When studying unranked trees, it makes sense to study trees, contexts and terms which have many roots – which leads to a whole new set of parameters.

For each choice of parameters there is an algebra. Due to a lack of space and interest, we will not enumerate all these algebras.

One solution for controlling the parameters is the framework of $\mathcal{C}$-varieties from [20], which also works for trees.

We choose a different solution. We give a general recipe for designing an algebra, and then uses it to design some algebras for trees. The recipe requires three steps. Each step is described by a question.

(1) **What are the objects?** In the first step, we choose what objects will be represented. Some possible choices:
   (a) Multicontexts of arities $\{0, 1, \ldots\}$, which may have several roots.
   (b) Multicontexts as above, but where none of the holes is in a root.
   (c) Multicontexts of arity at most one, which may have several roots.

   If there are different kinds of objects, the algebra might require several sorts. For instance, in the last case we would have two sorts: for arities zero and one.

(2) **What are the operations?** In the second step, we design the operations. The operations should not depend on the alphabet. These are designed so that if we take an unranked alphabet A, and start with contexts of the form $a\Box$ (a node with label a, with a single child that is a hole), then all the other objects can be generated using the operations. There are other ways of interpreting letters as generators, but we stay with $a\Box$ in the interest of reducing the already large number of models. Note that the objects represented in the algebra, as chosen in the first step, must include at least the generator contexts.

(3) **What are the axioms?** In the first two steps, we have basically designed the free algebra. In the last step, we provide the axioms. These should be chosen so that the free algebra designed in the first two steps is free in the sense of universal algebra. In other words, if we take all possible expressions that can be constructed from the generators $a\Box$ and the operations designed in the second step; and quotient these expressions by the least congruence including the axioms, then we get the objects designed in the first step.

We use the recipe to design three algebras: preclones in Section 4, seminearrings in Section 6, and forest algebra in Section 5. The reader can use the recipe to design other algebras.

An important algebra not included in this chapter is the *tree algebra* of Thomas Wilke, see [29]. The algebraic approach to tree languages, as described in the recipe above, was pioneered by this tree algebra. Also, tree algebra was used to give one of the first nontrivial effective characterizations of a tree logic, namely an effective characterization of frontier testable languages, again see [29]. Nevertheless, tree algebra is omitted from this chapter, mainly due to its close similarity with the forest algebra, which is described in Section 5, and which was inspired by tree algebra.

4 Preclones

As we saw in the study of definite tree languages in the previous section, in some cases it is convenient to extend an A-algebra with terms of arities $1, 2, 3$ and so on. So why not include all these objects in the algebra? This is the idea behind preclones.

What are the objects? The objects represented by a preclone are all multicontexts. For each arity, there is a separate sort. Consequently, there are infinitely many sorts.

What are the operations? Suppose that A is a ranked alphabet. Each letter of arity k can be treated as an element of the sort for arity k. We want to design the operations so

that from the letters, all possible multicontexts can be built. All we need is substitution: for an m-ary multicontext, an n-ary multicontext, and a hole number $i \in \{1, \ldots, m\}$, return the multicontext $p \cdot_i q$ of arity $m + n - 1$, obtained by replacing the i-th hole of p with q. Formally speaking, if the sorts are $\{T_m\}_{m \in \mathbb{N}}$, then we have an infinite set of operations

$$(u \in T_m, v \in T_n) \quad \mapsto \quad u \cdot_i v \in T_{m+n-1} \qquad \text{for } m, n \in \mathbb{N} \text{ and } i \in \{1, \ldots, m\}.$$

What are the axioms? A preclone should satisfy the following associativity axiom for any arities k, n, m and terms u, v, w of these arities, respectively.

$$(u \cdot_i v) \cdot_j w = \begin{cases} (u \cdot_j w) \cdot_{i+m-1} v & \text{for } j \in \{1, \ldots, i-1\} \\ u \cdot_i (v \cdot_{j-i+1} w) & \text{for } j \in \{i, \ldots, i+n-1\} \\ (u \cdot_{j-n+1} w) \cdot_i v & \text{for } j \in \{i+n, \ldots, k+n-1\} \end{cases}$$

The axioms are justified below, where we prove that the free algebra indeed consists of multicontext, as we postulated in the first step of the design process.

Preclones from an algebra. Each algebra, as described in Section 2, can be extended to a preclone. Consider a ranked alphabet A. The *preclone of an A-algebra $\mathcal{A}$* is defined by using k-ary $\mathcal{A}$-terms as the k-th sort, and the natural notion of substitution.

Free preclone. Suppose that A is a ranked alphabet. Consider the preclone where the sort of arity k consists of k-ary multicontexts over alphabet A, and the substitution operations are defined in the natural way. Call this preclone the *free preclone over alphabet A*. It is isomorphic to the preclone of the free A-algebra.

The notion of preclone morphism is inherited from universal algebra. The only point of interest is that preclones have multiple sorts. A *preclone morphism* between two preclones $\mathcal{T}$ and $\mathcal{U}$ is a function, which maps alements of the k-th sort of $\mathcal{T}$ to elements of the k-th sort of $\mathcal{U}$, and preserves the substitution operations in the natural way.

This preclone is free in the following sense, which justifies our choice of axioms in the design process. Let $\mathcal{T}$ be any preclone, and consider a function which maps each letter $a \in A$ to an element of $\mathcal{T}$ of the same arity. Then this function can be extended in a unique way to a preclone morphism from the free preclone to $\mathcal{T}$.

Notation. We are only interested in preclones that are either finitary (each sort is finite) or free. We use two different notations for elements of such preclones. For nullary elements in a finite preclone, we use letters f, g, h. For elements of arity one or more in a finite preclone, we use letters u, v, w. For nullary elements in the free preclone, which are trees, we use letters s, t, u. For elements of arity one or more in a free preclone, which are multicontexts, we use letters p, q, r.

4.1 An analogue of idempotent for binary terms

The notion of idempotent plays a very important role in the study of finite semigroups. An idempotent is used in algebraic versions of pumping arguments, where it serves as

a placeholder for "very long word". We have seen this in Section 2.1 on definite word languages.

What about trees? Is there a notion of idempotent?

One notion of idempotent is a context, which is idempotent in the semigroup of contexts. We tried this notion, with limited success, in Section 2.1. Despite its limitations, this notion plays an important role in algebras for tree languages.

However, there is another, less obvious notion, which we present in this section. This notion talks about binary terms. Here is the result that we generalize: every finite semigroup has a subsemigroup with just one element. For semigroups, the proof is straightforward. We start with any element s of the semigroup, and take the idempotent power s^ω of s. By idempotency, $\{s^\omega\}$ is a subsemigroup.

The generalization for preclones is stated below. It talks about finitary preclones. Recall that a preclone is called finitary if for every k, the sort of arity k is finite.

Theorem 4.1. *Consider a finitary preclone with terms of all arities. There is a sub-preclone where there is only one nullary, one unary, and one binary term.*

The theorem does not generalize to include ternary terms. The assumption on having terms with all arities is not as strong as it looks: a preclone has terms of all arities if and only if it has a nullary term and some term of arity at least two. The rest of Section 4.1 is devoted to showing the theorem. We start with a simple lemma.

Lemma 4.2. *Consider a finitary preclone with terms of all arities. There is a sub-preclone where there are terms of all arities, but only one nullary term.*

Proof. Fix a term u of arity at least $k \geqslant 2$ in the clone, and a nullary term h. For $i \in \mathbb{N}$, consider the term u_i of arity k^i defined by

$$u_1 = u, \qquad u_{i+1} = u(u_i, \ldots, u_i).$$

Let h_i be the nullary term obtained from u_i by substituting h in all holes. Since the clone is finitary, there must be some $i < j$ such that $h_i = h_j$. By induction on expression size, one shows that any expression built out of u^{j-i} and h_i that evaluates to a nullary term has value h_i. Therefore, the sub-preclone generated by u^{j-i} and h_i has only one nullary term, namely h_i. □

We will use a lemma on finite semigroups, which is stated below. A proof can be found in [4]. An alternative proof would use Green's relations.

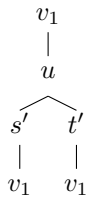
Lemma 4.3. *Let s, t be elements of a finite semigroup S. For some $s', t' \in S$ we have*

$$\hat{s} = \hat{s}\hat{s} = \hat{s}\hat{t} \qquad \hat{t} = \hat{t}\hat{t} = \hat{t}\hat{s} \qquad \text{for } \hat{s} = ss', \hat{t} = tt'.$$

Proof of Theorem 4.1. Let the finitary clone be $\mathcal{T}$. Thanks to Lemma 4.2, we may assume that $\mathcal{T}$ has only one nullary term, call it h. Let u be some binary term in $\mathcal{T}$. Our goal is to find a sub-preclone $\mathcal{U}$, which has only one unary term v_1, and one binary term v_2. Define two unary terms:

$$s = u \cdot_2 h \qquad t = u \cdot_1 h.$$

Consider the semigroup S of unary terms generated by s, t, and apply Lemma 4.3, resulting in unary terms $\hat{s}, \hat{t}$. The term $\hat{s}$ will be the unique unary term v_1 in the sub-preclone $\mathcal{U}$ that we are defining. The unique binary term v_2 is defined as



Let $\mathcal{U}$ be the sub-preclone of $\mathcal{T}$ generated by h, v_1 and v_2. We claim that v_1 is the only unary term in $\mathcal{U}$ and that v_2 is the only binary term in $\mathcal{U}$. To prove the claim, we use two sets of identities.

The first set of identities says that extending any term from $\mathcal{U}$ with v_1, either at the root or in some hole, does not affect that term:

$$v_1 \cdot v = v \quad \text{and} \quad v \cdot_i v_1 = v \qquad \text{for any } k\text{-ary term } v \text{ in } \mathcal{U}, \text{ and } i \in \{1, \ldots, k\}.$$

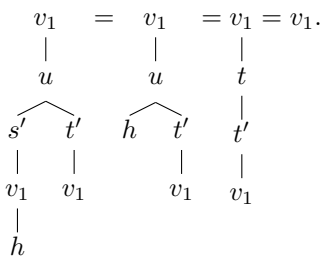
(In $v_1 \cdot v$, we use the operation $\cdot$ which substitutes v for all holes of v_1. Since v_1 is a unary term, this is the same as $v_1 \cdot_1 v$.) The identities hold when $k = 0$, since there is only one nullary term. When $k \geqslant 1$, then the root and all holes of v are padded by v_1, which is idempotent, since it was obtained from Lemma 4.3.

The second set of identities says that plugging either hole in v_2 with the unique nullary term h gives v_1:

$$v_2 \cdot_1 h = v_1 \qquad \text{and} \qquad v_2 \cdot_2 h = v_1$$

We only prove the first identity, the second one is shown the same way.

$$v_2 \cdot_1 h =$$



$$=$$

$$=$$

$$= v_1 = v_1.$$

The first equality is by definition of v_2. The second equality is because h is the only nullary term. The third equality is by definition of $s = u \cdot_1 h$. The last equality is by the properties of $v_1 = \hat{s}$ from Lemma 4.3.

Using the two sets of identities, one shows that if an expression built from h, v_1 and v_2 evaluates to a term of arity at most two, then that term is one of h, v_1, v_2. □

4.2 An application to logic

In this section we present an application of Theorem 4.1 to logic.

Consider words, and first-order logic with the order relation $\leqslant$ on word positions.

Every sentence (a formula without free variables) is logically equivalent to a sentence that uses only three variables. This follows, for instance from Kamp's theorem [18] on the equivalence of first-order logic and LTL, or from the McNaughton-Papert theorem [19] on the equivalence of first-order logic and star-free expressions. The reason is that any LTL formula, or any star-free expression, can be translated into a sentence of first-order logic with at most three variables.

The three variable theorem fails on trees if the signature has the descendant order, but does not have access to sibling order. The counterexample is very simple. Consider an alphabet with a letter a of arity $2n+1$, and nullary letters b and c. A sentence with n variables, which uses the descendant order and labels, cannot distinguish the two trees

$$a(\overbrace{b,\ldots,b}^{n \text{ times}},\ \overbrace{c,\ldots,c}^{n+1 \text{ times}}) \qquad \text{and} \qquad a(\overbrace{b,\ldots,b}^{n+1 \text{ times}},\overbrace{c,\ldots,c}^{n \text{ times}}).$$

A sentence with $n+1$ variables can distinguish the two trees. For $n=3$, the example above shows that three variables are not sufficient to capture all first-order logic.

The counterexample above no longer works if we allow sibling order. (Sibling order is the partial order which orders siblings, and does not order node pairs that are not siblings. Equivalently, one can use the lexicographic linear order on nodes. This is because in the presence of the descendant order, the sibling and lexicographic orders can be defined in terms of each other.) Actually, under the signature which has the descendant and sibling orders, every sentence can be expressed using only three variables.

The picture becomes more interesting if we allow free variables in formulas. Of course, if a formula has more than three free variables, then some special statement of the three variable theorem is needed. Here is a solution, which works for words: any formula with free variables $x_1,\ldots,x_n$ is equivalent, over words, to a boolean combination of formulas $\theta(x_i,x_j)$ that have two free variables and use three variables. What about trees? The following theorem shows that the result fails.

Theorem 4.4. *Consider first-order logic with the descendant and sibling orders. The following formula with free variables x,y,z*

$$\forall u \quad (u \leqslant x \wedge u \leqslant y) \Rightarrow (u \leqslant z)$$

is not equivalent to any boolean combination of formulas with two free variables.

A corollary is that the formula from the theorem is not equivalent to any formula which uses only three variables, including the bound variables. Suppose that the equivalent formula is $\varphi(x,y,z)$. By stripping φ to the first quantifiers, we see that φ is a boolean combination of two-variable formulas, which is impossible by Theorem 4.4.

The rest of this section is devoted to proving Theorem 4.4. The proof uses preclones and Theorem 4.1. We first show how preclones can describe formulas with free variables.

Let $\bar{x} = x_1,\ldots,x_n$ be a tuple of nodes in a tree t. The *order type* of $\bar{x}$ in t consists of information about how the nodes in the tuple are related with respect to the descendant and lexicographic order. Define x_0 to be the root. The $\bar{x}$-decomposition of t is a tuple $(p_0,p_1,\ldots,p_n)$, where p_i is the multicontext obtained from t by setting the root in x_i and placing holes in all the minimal proper descendants of x_i from $\{x_1,\ldots,x_n\}$.

Lemma 4.5. *Let θ be a formula with free variables, over a ranked alphabet A. There is a morphism α from the free preclone over A into a finitary preclone $\mathcal{T}$ such that the answer of θ for a tuple of nodes $\bar{x}$ in a tree t depends only on the order type of $\bar{x}$ and the image under α of the $\bar{x}$-decomposition of t.*

The lemma above actually holds even if θ is defined in a stronger logic, namely MSO. Since the lemma is our only interface to the logic in the proof below, we see that a stronger version of Theorem 4.4 holds: the formula from the theorem is not equivalent to any boolean combination of MSO formulas with two free variables.

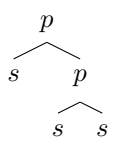
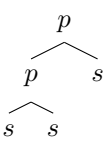
Proof of Theorem 4.4. Suppose that $\psi(x, y, z)$ is equivalent to a boolean combination of formulas with two free variables. These formulas can be of the form $\theta(x, y)$, $\theta(y, z)$ or $\theta(x, z)$. Let Γ be the set of these formulas θ. Toward a contradiction, we will construct trees t_1, t_2 and tuples of nodes $(x_1, y_1, z_1), (x_2, y_2, z_2)$ such that

$$t_1 \models \psi(x_1, y_1, z_1) \qquad \text{and} \qquad t_2 \not\models \psi(x_2, y_2, z_2) \tag{4.1}$$

but for every formula $\theta \in \Gamma$, we have

$$\begin{aligned} t_1 \models \theta(x_1, y_1) \quad &\iff \quad t_2 \models \theta(x_2, y_2), \\ t_1 \models \theta(y_1, z_1) \quad &\iff \quad t_2 \models \theta(y_2, z_2), \\ t_1 \models \theta(x_1, z_1) \quad &\iff \quad t_2 \models \theta(x_2, z_2). \end{aligned}$$

For each $\theta \in \Gamma$, apply Lemma 4.5, resulting in a morphism α_θ from the free preclone over alphabet A into a finitary preclone $\mathcal{T}_\theta$. Let $\mathcal{T}$ be the product of these preclones, and let α be the corresponding product morphism. Apply Theorem 4.1 to the preclone $\mathcal{T}$, obtaining a sub-preclone $\mathcal{U}$. Let s be some tree that is mapped by α to the unique nullary term v_0 in $\mathcal{U}$, and let p be some binary multicontext that is mapped by α to the unique binary term v_2 in $\mathcal{U}$. Consider the two trees

$t_1 =$  $t_2 =$ 

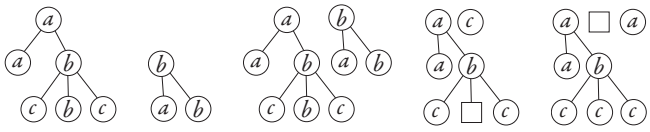
Define x_1, y_1, y_1 to be the roots of the three s trees in t_1, from left to right. Likewise for x_2, y_2, z_2. It is not difficult to see that (4.1) holds. We now show that each binary query $\theta \in \Delta$ gives the same answer for (x_1, y_1) in t_1 as it does for (x_2, y_2) in t_2. The same argument works for the other two combinations of variables. By Lemma 4.5, it suffices to show that the order type is the same for (x_1, y_1) is the same as for (x_2, y_2), which it is, and that the images under α are the same for the (x_1, y_1)-decomposition of t_1 and for the (x_2, y_2)-decomposition of t_2. But these images are necessarily the same, since they belong to the pre-clone $\mathcal{U}$, which has only one term in the nullary and binary sorts.

References. Preclones where introduced in [14]. One of the themes studied in [14] was the connection of first-order logic to the block product for preclones; we will come back to such questions in Section 7. Theorem 4.1 was proved in [4], although not in the formalism of preclones. Theorem 4.4 was suggested by Balder ten Cate.

5 Forest Algebra

We now present the second algebraic structure designed according to the recipe from Section 3, which is called forest algebra [10]. Forest algebra is defined for unranked trees. In an unranked tree, the alphabet A does not give arities for the letters. A tree over an unranked alphabet has no restriction on the number of children, apart from finiteness.

What are the objects? We work with ordered sequences of unranked trees, which we call *forests*. We adapt the definition of contexts to the unranked setting in the natural way, with the added difference that we allow several roots. More formally, a context over an unranked alphabet A is an ordered sequence of trees over alphabet $A \cup \{\Box\}$, where the symbol $\Box$ appears in exactly one leaf. We allow the hole to appear in a root, and also a context $\Box$ that consists exclusively of the hole. (Multicontexts, which have several holes, are not considered in forest algebra. They will appear in seminearrings, an algebra described in Section 6). Here are some examples.



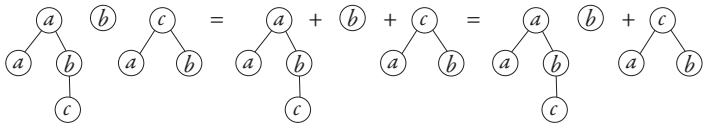
a tree a tree a forest a context a context

In a forest algebra, we choose our objects to be forests and contexts. These live in two separate sorts. These sorts are denoted H (as in horizontal), for the forest sort, and V (as in vertical) for the context sort.

What are the operations? We interpret each letter a of an unranked alphabet as the following context, which is also denoted $a\Box$.

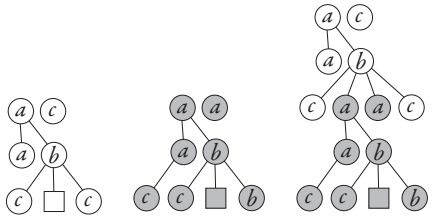
The operations below are designed so that for any alphabet, starting with contexts above, we can build all forests and contexts.

- Two constants: an empty forest 0, and an identity context $\Box$.
- Concatenating forests s and t, written $s + t$. This operation is illustrated below.



We can also concatenate a context p and a forest t, the result is a context $p + t$; likewise we can get $t + p$. We cannot concatenate two contexts, since the result would have arity two. Formally speaking, there are three concatenation operations, of types forest-forest, context-forest and forest-context, which formally should be denoted $+_{HH}$, $+_{VH}$ and $+_{HV}$. In most cases, we will skip these subscripts, which are determined by the sorts of the arguments.

- Composing two contexts p and q, written $p \cdot q$. This operation is illustrated below.



We can also substitute a forest t into the hole of a context p, the result is written $p \cdot t$. (Again, we have two different types of $\cdot$ operation, which should formally be distinguished by subscripts $\cdot_{VV}$ and $\cdot_{VH}$.) As usual with multiplicative notation, we sometimes skip the dot and write pq instead of $p \cdot q$. We also assume that $\cdot$ has precedence over $+$, so $pq + s$ means $(p \cdot q) + s$ and not $p \cdot (q + s)$.

It is not difficult to see that any forest or context over alphabet A can be constructed using the above operations from the contexts 0, $\Box$ and $\{a\Box\}_{a \in A}$. The construction is by induction on the size of the forest or context, and corresponds to a bottom up-pass.

What are the axioms? So far, we know that a forest algebra is presented by giving two sorts: forests H and contexts V, along with operations:

$$0 \in H \qquad \Box \in V$$
$$+_{HH} : H \times H \to H \qquad +_{HV} : H \times V \to V \qquad +_{VH} : V \times H \to V$$
$$\cdot_{VV} : V \times V \to V \qquad \cdot_{VH} : V \times H \to H$$

Of course, there are some axioms that need to satisfied, if we want the objects represented by the algebra to be forests and contexts.

(1) $(H, +_{HH}, 0)$ is a monoid (called the horizontal monoid).
(2) $(V, \cdot_{VV}, \Box)$ is a monoid (called the vertical monoid).
(3) The operations

$$+_{HV} : H \times V \to V \qquad +_{VH} : V \times H \to V \qquad \cdot_{VH} : V \times H \to H$$

are, respectively, a left monoidal action of H on V, a right monoidal action of H on V, and a left monoidal action of V on H. In other words, the following hold for any $g, h \in H$ and $v, w \in V$.

$$(h +_{HH} g) +_{HV} v = h +_{HV} (g +_{HV} v) \qquad 0 +_{HV} v = v$$
$$v +_{VH} (h + g) = (v +_{VH} h) +_{VH} g \qquad v +_{VH} 0 = v$$
$$(v \cdot_{VV} w) \cdot_{VH} h = v \cdot_{VH} (w \cdot_{VH} h) \qquad \Box \cdot_{VH} h = h$$
$$(h +_{HV} v) +_{VH} g = h +_{HV} (v +_{VH} g)$$

This completes the design process. Below, when talking about the free forest algebra, we will justify why the axioms above are the right ones. First though, we define morphisms.

Free forest algebra. The notion of forest algebra morphism is inherited from universal algebra. A forest algebra morphism between two forest algebras is a function which maps the horizontal sort of the first algebra into the horizontal sort of the second algebra, and which maps the vertical sort of the first algebra into the vertical sort of the second algebra. We write a morphism α from a forest algebra (H, V) into a forest algebra (G, W) as

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

For an unranked alphabet A, we define *free forest algebra over an alphabet* A, which is denoted by (H_A, V_A). The elements of the horizontal sort H_A are all forests over A, and the elements of the vertical sort are all contexts over A. This is indeed a free object in the category of forest algebras, as stated by the following result. Let (H, V) be a forest algebra, and consider any function $f : A \to V$. There exists a unique forest algebra morphism $\alpha : (H_A, V_A) \to (H, V)$ which extends the function f in the sense that $\alpha(a\Box) = f(a)$ for all $a \in A$.

Recognizing languages. A forest language L over alphabet A is said to be *recognized* by a morphism α from the free forest algebra over A into a forest algebra (H, V) if membership $t \in L$ depends only on $\alpha(t)$. A language is recognized by a forest algebra if it is recognized by a forest algebra morphism into that algebra. One can show finite forest algebras recognize exactly the regular forest languages, in any one of the many equivalent definitions of regularity for forest languages (such as hedge automata [] or MSO).

Syntactic forest algebra. Forest algebra also has a notion of syntactic object. Consider a forest language L over an alphabet A, which is not necessarily regular. We define a Myhill-Nerode equivalence relation on the free forest algebra as follows.

$s \sim_L t$ holds for $s, t \in H_A$ if $ps \in L \iff pt \in L$ for all $p \in V_A$,

$p \sim_L q$ holds for $p, q \in V_A$ if $rps \in L \iff rqs \in L$ for all $r \in V_A$, $s \in H_A$.

This two-sorted equivalence relation is a congruence for all operations of forest-algebra, hence it makes sense to consider a quotient of the free forest algebra with respect to the equivalence. This quotient is called the *syntactic forest algebra* of L, and it is denoted by (H_L, V_L). The morphism which maps each forest or context to its equivalence class is called the *syntactic forest algebra morphism*, and is denoted by α_L. As is typical for syntactic morphisms, any (surjective) forest algebra morphism that recognizes L can be uniquely extended to α_L. Consequently, a language is regular if and only if its syntactic forest algebra is finite.

A simple example: label testable languages A forest language is called *label testable* if membership of a forest in the language depends only on the set of labels that appear in the forest. In other words, this is a boolean combinations of languages of the form "forests that contain some node with label a".

Theorem 5.1. *A forest language L is label testable if and only if its syntactic forest algebra (H_L, V_L) satisfies the identities:*

$$vv = v, \qquad vw = wv \qquad \textit{for } v, w \in V_L$$

Proof. The "only if" part is straightforward, we only concentrate on the "if" part. Suppose that identity in the statement of the theorem is satisfied. We will show that for every forest t, its image $\alpha_L(t)$ under the syntactic forest algebra morphism α_L depends only on the set of labels appearing in t.

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

$$h + h = (h + \Box)(h + \Box)0 = (h + \Box)0 = h \; .$$

The second is the commutativity of the horizontal monoid:

$$h + g = (h + \Box)(g + \Box)0 = (g + \Box)(h + \Box)0 = g + h \; .$$

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

$$vh = h + v0 \; .$$

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

$$vh = v(h + \Box)0 = (h + \Box)v0 = h + v0 \; .$$

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

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

5.1 A more difficult example: the logic EF

In this section we use forest algebras to give an effective characterization of a temporal logic called EF. Even thought the proof is a bit (but not too much) involved, we present it in full, because it illustrates how concepts familiar from semigroup theory appear in forest algebra, in a suitably generalized form. These concepts include ideals and Green's relations.

The logic EF. We begin by defining the logic. Because forest algebras are better suited to studying forests, rather than trees, we provide a slightly unusual definition of the logic EF, which allows formulas to express properties of forests.

There are two kinds of EF formulas: tree formulas, which define tree languages, and forest formulas, which define forest languages. The most basic formula is a, for any letter of the alphabet, this is a tree formula that is satisfied by trees with a in the root. If φ is a tree formula, then $\mathsf{EF}\varphi$ is a forest formula, which defines the set of forests t where some subtree satisfies φ. (If $t_1, \ldots, t_n$ are trees, then a subtree of the forest $t_1 + \cdots + t_n$

is a subtree of any of the trees $t_1, \ldots, t_n$, possibly one of these trees.) If φ is a forest formula, then $[\varphi]$ is a tree formula that is satisfied by trees of the form at, where t is a forest satisfying φ, and a is any label. Finally, both tree and forest formulas allow boolean operations $\vee$, $\wedge$ and $\neg$.

As far as properties of trees are concerned, our definition of EF is equivalent to the standard definition – just use $[\mathsf{EF}\varphi]$ instead of $\mathsf{EF}\varphi$.

Theorem 5.2. *A forest language is definable by a forest formula of* EF *if and only if its syntactic forest algebra satisfies the following identities, called the* EF *identities:*

$$g + h = h + g \tag{5.1}$$

$$vh = h + vh\ . \tag{5.2}$$

This theorem is stated for the nonstandard forest variant of EF. However, it can be used to characterize the tree variant. One can show that a tree language L can be defined by a tree formula of EF if and only if for every label a, the forest language $\{t : at \in L\}$ can be defined by a forest formulas of EF.

We begin the proof with the "only if" implication in the theorem. Fix a forest formula φ of EF. For a forest t, consider the set of tree subformulas of φ that are true in some subtree of t. We say that two forests are φ-equivalent if these sets coincide for them. Observe that if forests s, t are φ-equivalent, then so are ps, pt for any context p. Consider the syntactic forest algebra morphism of the forest language defined by φ. By definition of the syntactic morphism, it follows that φ-equivalent forests have the same image under thee syntactic forest algebra morphism. It is also easy to see that $s + t$ is φ-equivalent to $t + s$ for any forests s, t. Likewise, pt is φ-equivalent to $t + pt$. It follows that the EF identities must be satisfied by the syntactic forest algebra of a forest language defined by φ.

The rest of Section 5.1 is devoted to the more interesting "if" implication in Theorem 5.2. Consider a forest algebra morphism α from a free forest algebra (H_A, V_A) into a finite forest algebra (H, V) that satisfies the two EF identities. We will show that any forest language recognized by α can be defined by a forest formula of EF. This gives the "if" implication in the case when α is the syntactic forest algebra morphism.

The proof is by induction on the size of H. The induction base, when H has one element, is immediate. A forest algebra with one element in H can only recognize two forest languages: all forests, or no forests. Both are definable by forest formulas of EF.

The key to the proof is a relation on H, which we call reachability. We say that $h \in H$ is *reachable* from $g \in H$ if there is some $v \in V$ such that $h = vg$. Another definition is that h is reachable from g if $Vh \subseteq Vg$. That is why we write $h \leqslant g$ when h is reachable from g. This relation is similar in spirit to Green's relations. (Note that both H and V are monoids, so they have their own Green's relations in the classical sense.)

Lemma 5.3. *If the* EF *identities are satisfied, then reachability is a partial order.*

Proof. The reachability relation is transitive and reflexive in any forest algebra. To prove that it is antisymmetric, we use the EF identities. Suppose that $g, h \in H$ are reachable

from each other, i.e. $g = vh$ and $h = wg$ for some $v, w \in V$. Then they must be equal:

$$h = wvh \overset{(5.2)}{=} vh + wvh = vh + h \overset{(5.1)}{=} h + vh \overset{(5.2)}{=} vh = g.$$

□

The reachability order has smallest and greatest elements, which we describe below. Every forest is reachable from the empty forest 0, which is the greatest element. The smallest element, call it $h_\perp$, is obtained by concatenating all the elements of H into a single forest $h_1 + \cdots + h_n$. This element is reachable from all elements of H, by

$$h_\perp = (h_1 + \cdots + h_{i-1} + \Box + h_{i+1} + \cdots + h_n)h_i.$$

For $h \in H$, we define $H_h \subseteq H$ to be the set of elements from which h is *not* reachable. This set is an ideal of forest algebra in the sense that the following inclusions hold.

$$H_h + H \subseteq H_h \qquad H + H_h \subseteq H_h \qquad VH_h \subseteq H_h$$

Consider the equivalence $\sim_h$, which identifies all elements of H_h into one equivalence class, and leaves all other elements distinct. We can extend this equivalence to V by keeping all elements of V distinct. Because H_h is an ideal, the resulting two-sorted equivalence is a congruence for all operations of forest algebra. Therefore, it makes sense to consider the quotient forest algebra morphism α_h from (H, V) into the quotient forest algebra $(H, V)_{/\sim_h}$.

Our strategy for the rest of the proof is as follows. For every element $h \in H$, we will prove that the inverse image $\alpha^{-1}(h)$ can be defined by a forest formula of EF, call it φ_h. Consequently, every language recognized by α is definable by a forest formula of EF, as finite disjunction of formulas φ_h.

In our analysis, we pay special attention to subminimal elements. An element $h \in H$ is called *subminimal* if there is exactly one element $g < h$, the smallest element $h_\perp$.

Consider an element h that is neither subminimal nor the smallest element. Recall the congruence $\sim_h$ and the resulting forest algebra morphism α_h. By definition of α_h, for any forest t with h reachable from $\alpha(t)$, the images of t under α_h and α are the same. It follows that $\alpha^{-1}(h)$ is recognized by the morphism α_h. Since H_h contains at least two elements (a subminimal element and the smallest element) then $\sim_h$ is a nontrivial congruence. Consequently, $\alpha^{-1}(h)$ can be defined by a forest formula of EF, using the induction assumption.

Consider an element h that is subminimal. Then H_h contains the smallest element $h_\perp$ and all the subminimal elements different than h. (Here we use the assumption that $\leqslant$ is a partial order.) Therefore, if there are at least two subminimal elements, then $\alpha^{-1}(h)$ is definable by a forest formula of EF, call it φ_h, for any element $h \neq h_\perp$. It remains to give a formula for $h_\perp$. This formula says that $h_\perp$ correponds to all other forests: $\neg \bigvee_{h \neq h_\perp} \varphi_h$.

We are left with the case where there is exactly one subminimal element, call it h_*. By the reasoning above we have formulas for all elements except the smallest $h_\perp$ and the unique subminimal h_*. It is therefore enough to give a formula that distinguishes between the two. We do this below.

We begin by defining some tree formulas. Consider an element h that is neither $h_\perp$

nor h_*. Below we define a *tree* formula ψ_h of EF that describes trees mapped by α to h.

$$\psi_h = \bigvee_{\substack{a \in A, g \geqslant h \\ \alpha(a)g = h}} a \wedge [\varphi_g].$$

Thus far, for each element h other than $h_\perp$ and h_* we have a forest formula φ_h and a tree formula ψ_h. We would like similar formulas for h_*. However, we will have to deal with a certain loss of precision: instead of saying that a tree (or forest) is mapped to h_*, we will say that it is mapped to either h_* or $h_\perp$. This is accomplished by the formulas

$$\varphi_{h_*} = \bigwedge_{h \neq h_\perp, h_*} \varphi_h \qquad \text{and} \qquad \psi_{h_*} = \bigwedge_{h \neq h_\perp, h_*} \psi_h.$$

We now conclude the proof, by giving a forest formula of EF that describes $\alpha^{-1}(h_\perp)$. When is a forest t mapped by α to $h_\perp$? One possibility is that t has a subtree mapped to $h_\perp$. If that subtree is taken minimal, then it is of the form as, with $\alpha(s) \neq h_\perp$ and $\alpha(a)\alpha(s) = h_\perp$. Another possibility is that t has no subtrees with mapped to h_0, in which case t is a concatenation of trees $t = t_1 + \cdots + t_n$ with

$$\alpha(t_1), \ldots, \alpha(t_n) \in H - \{h_\perp\}, \qquad \alpha(t_1) + \cdots + \alpha(t_n) = h_\perp.$$

By expressing the above analysis in a formula, we see that any forest mapped by α to $h_\perp$ satisfies the following formula.

$$\bigvee_{\substack{a \in A, h \neq h_\perp \\ \alpha(a)h = h_\perp}} a \wedge [\varphi_h] \qquad \vee \qquad \bigvee_{\substack{h_1, \ldots, h_n \in H - \{h_\perp\} \\ h_1 + \cdots + h_n = h_\perp}} \mathsf{EF}\psi_{h_1} \wedge \cdots \wedge \mathsf{EF}\psi_{h_n}.$$

We also prove the converse implication: any forest that satisfies the above formula is mapped by α to $h_\perp$.

Suppose that a forest satisfies the first disjunct, for some a and h with $\alpha(a)h = h_\perp$. This means that the forest has a subtree as, where s satisfies φ_h. If h is not h_*, then we know that s is mapped to h by α, and therefore as is mapped to $h_\perp$ by α. Any forest that has a subtree mapped to $h_\perp$ must necessarily be mapped to $h_\perp$ itself, by definition of the reachability order.

Suppose that a forest t satisfies the second disjunct, for some choice of $h_1, \ldots, h_n$. Let $t_1, \ldots, t_n$ be the subtrees of t that satisfy the formulas $\psi_{h_1}, \ldots, \psi_{h_n}$. One possibility is that some h_i is h_*, and the tree t_i is mapped to $h_\perp$ (recall the loss of precision in the formula ψ_{h_*}). In this case we are done since also t is mapped to $h_\perp$. Otherwise, every tree t_i is mapped to h_i. Consider now any of the trees t_i. Since it is a subtree of t, there is a context p_i with $t = p_i t_i$. Therefore, we have

$$\alpha(t) = \alpha(p_i t_i) \stackrel{(5.2)}{=} \alpha(t_i) + \alpha(p_i t_i) = h_i + \alpha(t).$$

By applying the above to all the trees t_i, we get

$$\alpha(t) = h_1 + \cdots + h_n + \alpha(t) = h_\perp + \alpha(t) = h_\perp.$$

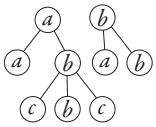
References Forest algebra was introduced in [10]. The characterization of EF also comes from [10], although it is based on a characterization for ranked trees from [9]. Forest algebra was used to obtain algebraic characterizations of several tree logics: a vari-

ant of EF with past modalities [3]; boolean combinations of purely existential first-order formulas [7]; languages that correspond to level Δ_2 of the quantifier alternation hierarchy [6]. A variant of forest algebra for infinite trees was proposed in [5].

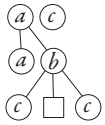
6 Seminearring

In this section we present the third and final algebraic structure that is designed using the recipe in Section 3. The algebraic structure is called a seminearring; it is like a semiring, but with some missing axioms.

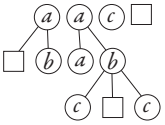
What are the objects? We want to represent all multicontexts, which are unranked forests with any number of holes. More formally, a *multicontext* over an unranked alphabet A is an ordered sequence of unranked trees over alphabet $A \cup \{\Box\}$, where the symbol $\Box$ appears only in leaves. We allow the hole to appear in a root, and also multicontexts that consist exclusively of holes. Here are some examples.



a forest, which is a multicontext of arity 0



a context, which is a multicontext of arity 1

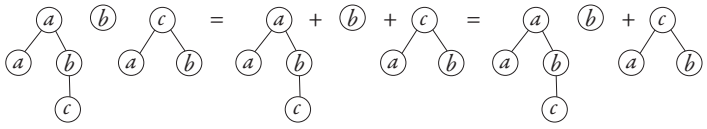


a multicontext of arity 3

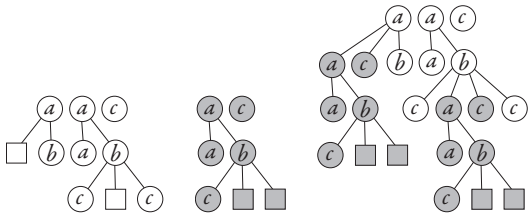
In a seminearring, we choose our objects to be all multicontexts. A seminearring has only one sort; all multicontexts live in the same sort.

What are the operations? We have to design the operations so that for any alphabet A, starting with contexts $a\Box$, we can build all other multicontexts.

- Two constants: a multicontext 0 with no holes and no nodes; and a multicontext $\Box$ with a single hole.
- Concatenating two multicontexts p and q. The result, denoted $p + q$ has an arity which is sum of arities of p and q. This operation is illustrated below.



- Composing two multicontexts p and q. The result, denoted $p \cdot q$ or pq, is obtained from p by substituting each hole by q. The arity of pq is the product of arities of p and q.

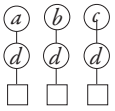


Any multicontext p over alphabet A can be constructed using concatenation and composition from the multicontexts 0, $\Box$ and $\{a\Box\}_{a\in A}$. The construction is by induction on the size of p, and corresponds to a bottom up-pass. This completes the second state of the design process.

What are the axioms? Consider two expressions that use the operations above, e.g.

$$\big(a + (b + c)\big) \cdot d \qquad \text{and} \qquad \big((a \cdot d) + (b \cdot d)\big) + (c \cdot d).$$

These expressions should be equal, since they describe the same multicontext, namely



(As usual, we treat each letter a as describing the multicontext $a\Box$.) We need to design the axioms so that they imply equality of the two expressions given above, and other equalities like it. We use the following axioms, which can be stated as identities.

(1) Concatenation $+$ is associative, with neutral element 0.
(2) Composition $\cdot$ is associative, with neutral element $\Box$. [3]
(3) For any $p, q, r \in N$ we have left-distributivity: $(p + q) \cdot r = p \cdot r + q \cdot r$.
(4) In the composition monoid, 0 annihilates to the left: $0 \cdot p = 0$.

The axioms above complete the design process. The algebraic object described above is called a *seminearring*. It is like a semiring, but some axioms are missing. In Section 6.2, we will say what properties of trees can be described by those seminearrings which are semirings.

We will write $\mathcal{U}, \mathcal{V}, \mathcal{W}$ for seminearrings, and u, v, w for their elements.

Free seminearring. If A is an alphabet, we write $\mathcal{V}_A$ for the seminearring whose domain consists of all multicontexts over alphabet A. This seminearring is free in the following sense. Let A be an alphabet and $\mathcal{V}$ be a seminearring. Any function from $\{a\Box : a \in A\}$ to the domain of $\mathcal{V}$ extends uniquely to a morphism $\alpha : \mathcal{V}_A \to \mathcal{V}$.

Recognizing languages. We say that a forest language L is *recognized* by a seminearring morphism $\alpha : \mathcal{V}_A \to \mathcal{V}$ if for every forest t, treated as a nullary multicontext, membership $t \in L$ depends only on the image $\alpha(t)$.

[3] For consistency with the other algebras in this chapter, we use $\Box$ instead of the more common 1.

Syntactic seminearring. A syntactic seminearring can be defined, like for forest algebra, using a Myhill-Nerode equivalence relation on the free seminearring as follows.

$p \simeq_L q$ holds for $p, q \in \mathcal{V}_A$ if $r_1 p r_2 0 \in L \iff r_1 q r_2 0 \in L$ for all $r_1, r_2 \in \mathcal{V}_A$.

Since elements of $\mathcal{V}_A$ are multicontexts, and elements of the form $r_2 0$ are all forests, the condition for $p \simeq_L q$ can be restated as

$$rpt \in L \iff rqt \qquad \text{for every forest } t \text{ and multicontext } r.$$

This is a congruence for all operations of seminearring, and hence it makes sense to define the syntactic seminearring and the syntactic seminearring morphism using the quotient under this relation.

6.1 From a seminearring to a forest algebra and back again

Suppose that (H, V) is a forest algebra. There is a natural way to construct a seminearring out of this forest algebra. Consider the least family $\mathcal{V}$ of functions $v : H \to H$ which:

- Contains the constant function $g \mapsto h$ for every $h \in H$.
- Contains the function $h \mapsto vh$ for every $v \in V$.
- Is closed under concatenation: if v, w belong $\mathcal{V}$, then so does $h \mapsto vh + wh$.
- Is closed under composition: if v, w belong $\mathcal{V}$, then so does $h \mapsto w(vh)$.

When equipped with the seminearring operations in the natural way, this set forms a seminearring, which we call the seminearring *induced* by the forest algebra (H, V).

Lemma 6.1. *The syntactic seminearring of a forest language is isomorphic to the seminearring induced by its syntactic forest algebra.*

From the above lemma it follows that two languages have the same syntactic forest algebra, then they have the same syntactic seminearring. In general, the converse fails. For the forest sort, there is no problem: use elements of the form $v0$. However, there is a problem for the context sort, since there is no way of telling which elements of a seminearring correspond to multicontexts of arity 1. This is illustrated in the following example.

Example 6.1. We present two languages, which have the same syntactic seminearring, but different syntactic forest algebras.

- The alphabet is $\{a, b\}$. The language is "there is an a". The syntactic seminearring has three elements $0, \Box, \infty$. The syntactic morphism is described below.
  0 is the image of multicontexts of arity 0 without an a;
  $\Box$ is the image of multicontexts of arity at least 1 without an a;
  ∞ is the image of multicontexts with an a.
  The operations in the seminearring are defined by the axioms and by

  $$\infty + v = v + \infty = \infty \cdot v = v \cdot \infty = \infty.$$

  The syntactic forest algebra of this language has two elements in the context sort, these elements correspond to $\Box$ and ∞. There is no element corresponding to 0, since every context must have a hole.

- The alphabet is $\{a, b, c\}$. The language is "some node has label a, but no ancestor with label b". The syntactic seminearring is isomorphic to the one above, only the syntactic morphism is different, as described below.
  - 0 is the image of multicontexts that have no a without b ancestors, and where every hole (if it exists) has a b ancestor.
  - $\Box$ is the image of multicontexts that have no a without b ancestors, but where some hole has no b ancestors.
  - ∞ is the image of multicontexts that have an a without b ancestors.

  The syntactic forest algebra of this language has three elements in the context sort, these elements correspond to $0, \Box$ and ∞.

Note that the first language in the example can be defined in the logic EF. The second example cannot be defined in EF, since violates identity (5.2), e.g. the forests ba and $ba + a$ have different images under the syntactic forest algebra morphism. This shows that there is no way of telling if a forest language can be defined in EF just by looking at its syntactic seminearring.

This is a general theme in the algebraic theory of tree languages. When we use an algebra that represents a richer set of objects, we lose information in the syntactic object. The classic example is that when we look at the syntactic monoid instead of the syntactic semigroup, we do not know if the identity element in the syntactic monoid represents some nonempty words in addition to the empty word. Of course a solution is to recover the lost information by taking into account the syntactic morphism. This solution is illustrated by the following theorem, which characterizes EF in terms of the syntactic seminearring morphism.

Theorem 6.2. *A forest language can be defined by a forest formula of EF if and only if its syntactic seminearring morphism $\alpha_L : \mathcal{V}_\mathcal{A} \to \mathcal{V}_L$ satisfies the identity*

$$v + w = w + v \tag{6.1}$$

for any elements $v, w \in \mathcal{V}_\mathcal{A}$ and the identity

$$v = v + \Box \tag{6.2}$$

for any element $v \in \mathcal{V}_\mathcal{A}$ that is the image under α_L of a multicontext of arity at least one.

One way to prove the theorem above is to use Theorem 5.2. One shows that the conditions (6.1) and (6.2) on the syntactic seminearring morphism above are equivalent to the conditions (5.1) and (5.2) on the syntactic forest algebra. Next, one applies Theorem 5.2.

Is there another way? What if one tries to prove Theorem 6.2 directly, using seminearrings? If, like in the forest algebra version, one tries to take the quotient under an ideal, the exposition becomes cumbersome, since one must take care to distinguish elements that are images of multicontexts of arity at least one.

The problems indicated above suggest the following heuristic: when studying a class of languages, use the algebraic structure which has the richest possible set of objects, and still allows to describe the class without referring to the syntactic morphism.

Example 6.2. Consider first-order logic with descendant and sibling orders. This example shows that seminearrings might not be the right formalism, since just by looking at the syntactic seminearring one cannot tell if a language can be defined in the logic.

- The alphabet is $A = \{a\}$. The language, call it L, contains trees where each leaf is at even depth; and each node has either zero or two children. As we have shown in the introduction, this language can be defined in first-order logic.
- The alphabet is $B = \{a, b\}$. The language, call it K, contains trees where each leaf is at even depth; each node with label a has zero or two children; each node with label b has zero or one child. This language cannot be defined in first-order logic, since it requires distinguishing between b^n and b^{n+1} for arbitrarily large n.

We claim the two languages have the same syntactic seminearring. Consider the two seminearring morphisms

$$\alpha : \mathcal{V}_A \to \mathcal{V}_B \qquad \beta : \mathcal{V}_B \to \mathcal{V}_A$$

which preserve multicontexts over alphabet $\{a\}$, and such that $\beta(b\Box) = a\Box + a\Box$. One can show that

$$L = \alpha^{-1}(K) \qquad K = \beta^{-1}(L)\ .$$

Consequently, L is recognized by the syntactic seminearring of K, and K is recognized by the syntactic seminearring of L. It follows that these seminearrings are isomorphic. On the other hand, one can show that the syntactic forest algebra of a language uniquely determines if the language can be defined in first-order logic (although we still do not know an algorithm which does this).

6.2 Which languages are recognized by semirings?

The axioms of a seminearring look very similar to those of a semiring, but some semiring axioms are missing. First, a semiring requires addition to be commutative.

$$w + v = v + w. \tag{6.3}$$

This is the least important difference, since in many examples addition will actually be commutative. The second difference with semirings is that we do not require $+$ to distribute over $\cdot$ to the right, i.e. we do not require

$$u \cdot (v + w) = u \cdot v + u \cdot w. \tag{6.4}$$

As we will see below, adding the above axiom corresponds to restricting all regular tree languages to a natural subclass, called the *path testable languages*. Often, one requires a semiring to satisfy also the following axiom:

$$v \cdot 0 = 0. \tag{6.5}$$

Unlike right distributivity (6.4), adding the above axiom does not seem to make any sense for multicontext algebra. We use the term *semiring* for a seminearring that satisfies the axioms (6.3) and (6.4). We *do not* require 0 to annihilate on the right in composition, i.e. we do not require (6.5).

Which forest languages are recognized by semirings? It turns out that these are languages which are determined by the set words labeling paths in a forest. These are described in more detail below.

Path testable languages. Let x be a node in a forest over an alphabet A. The path leading to x is the set of ancestors of x, including x. The *labeling* of a path is defined to be the word in A^* obtained by reading the labels of the nodes in the path, beginning in the unique root in the path, and ending in x. For any word language $L \subseteq A^*$, we define a forest language $\mathsf{E}L$ as follows. A forest belongs to $\mathsf{E}L$ if the labelings of some path (possibly a path that does not end in a leaf) belongs to L. A path testable language is a boolean combination of languages of the form $\mathsf{E}L$ (there can be different languages L involved in the boolean combination).

Theorem 6.3. *A regular forest language is path testable if and only if its syntactic seminearring is a semiring where addition (i.e. concatenation) is idempotent.*

Proof. The only if implication is straightforward, because the syntactic seminearring of a path testable language satisfies the two identities. Note that when $w = 0$, right distributivity says

$$u \cdot (v + 0) = u \cdot v + u \cdot 0.$$

which explains why the paths in path testable languages need not end in a leaf. For instance, the language "some leaf has label a" is not path testable.

Consider now the converse implication. We will prove that if $\mathcal{V}$ is a semiring with idempotent addition, then any forest language recognized by a seminearring morphism $\alpha : \mathcal{V}_A \to \mathcal{V}$ is path testable. Consider a forest s. For a path π in s, we write $tree_s(\pi)$ for the tree which consists exclusively of the nodes from s that appear in π (all nodes in this tree, except the last one, have one child).

The key observation is that any forest is equivalent to the concatenation of its paths:

$$\alpha(s) = \sum_{\pi \text{ a path in } s} \alpha(tree_s(\pi)). \tag{6.6}$$

Note that the $\sum$ symbol can be used without indicating an order on paths, since concatenation is commutative. The statement above is proved by induction on the size of s, using right distributivity. For any $v \in \mathcal{V}$, let L_v be the set of forests s where some path π satisfies $\alpha(tree_s(\pi)) = v$. This language is clearly path testable. Consequently, for any $W \subseteq \mathcal{V}$, the language L_W, defined as

$$\bigcap_{w \in W} L_w \quad - \quad \bigcup_{w \notin W} L_w$$

is also path testable; this is the set of forests where paths are mapped exactly to the elements of W. By (6.6) and idempotency of addition, a forest s is mapped by α to an element $w \in \mathcal{V}$ if and only if there is some set $W \subseteq \mathcal{V}$ such that $s \in L_W$ and $\sum W = w$. Therefore, the forests that are mapped by α to w form a path testable language, as a finite union of path testable languages L_W. Likewise for the language L itself. □

References. To the author's best knowledge, this chapter is the first time that seminearrings are explicitly used to recognize forest languages. The results on path testable languages in Section 6.2 are adapted from the forest algebra characterization of path testable languages in [10].

7 Nesting algebras

In this section we define an operation on algebras that simulates nesting of formulas. This type of operation can be defined for all the algebraic structures described in this chapter. We show it for seminearrings.

Wreath product of seminearrings. Consider two seminearrings $\mathcal{U}, \mathcal{V}$. We distinguish the operations of these seminearrings by subscripts, e.g. $+_{\mathcal{U}}$ is the concatenation in $\mathcal{U}$ and $0_{\mathcal{V}}$ is the additive neutral element in $\mathcal{V}$. Below we define a new seminearring $\mathcal{U} \circ \mathcal{V}$, which is called the *wreath product*. In the definition, we use the set $\mathcal{V}_0 = \{v \cdot_{\mathcal{V}} 0_{\mathcal{V}} : v \in \mathcal{V}\}$, whose elements we denote by letters f, g, h. The carrier of the wreath product consists of pairs (τ, v), where the first coordinate is a function $\tau : \mathcal{V}_0 \to \mathcal{U}$ and the second coordinate is an element $v \in \mathcal{V}$. The carrier can be viewed as a cartesian product of $\mathcal{V}_0$ copies of $\mathcal{U}$ and a single copy of $\mathcal{V}$. The cartesian product interpretation explains the constants $0, \Box$ and concatenation operation in the wreath product, which are defined coordinate-wise. The composition operation is not defined coordinate-wise, it is defined by

$$(\tau, u) \cdot (\sigma, v) = (f \mapsto \tau(v \cdot_{\mathcal{V}} f) \cdot \sigma(f), u \cdot_{\mathcal{V}} v) \qquad \text{for } \tau, \sigma : \mathcal{V}_0 \to \mathcal{U} \text{ and } u, v \in \mathcal{V}.$$

Lemma 7.1. *The wreath product of two seminearrings is also a seminearring.*

Proof. The additive structure is a monoid, as a cartesian product of monoids. It is not difficult to see that $\Box$ is an identity for composition, and that 0 from the additive monoid is a left zero for composition. For associativity of the composition operation, one notices that wreath product of seminearrings is a special case of wreath product of transformation semigroups, and the latter preserves associativity. It remains to show left distributivity:

$$((\tau_1, v_1) + (\tau_2, v_2))(\tau, v) \stackrel{?}{=} ((\tau_1, v_1) \cdot (\tau, v)) + ((\tau_2, v_2) \cdot (\tau, v)) \tag{7.1}$$

Let us inspect the first coordinate of the left side of the equation:

$$f \mapsto \sigma(v \cdot_{\mathcal{V}} f) \cdot_{\mathcal{U}} \tau(f) \qquad \text{where } \sigma \text{ is the function } f \mapsto \tau_1(f) +_{\mathcal{U}} \tau_2(f)$$

Therefore, the above becomes

$$f \qquad \mapsto \qquad (\tau_1(v \cdot_{\mathcal{V}} f) +_{\mathcal{U}} \tau_2(v \cdot_{\mathcal{V}} f)) \cdot_{\mathcal{U}} \tau(f)$$

By right distributivity of $\mathcal{U}$, the above is the same as

$$f \qquad \mapsto \qquad \big((\tau_1(v \cdot_{\mathcal{V}} f) \cdot_{\mathcal{U}} \tau(f)\big) +_{\mathcal{U}} \big(\tau_2(v \cdot_{\mathcal{V}} f) \cdot_{\mathcal{U}} \tau(f))\big).$$

which is the first coordinate of the right side of (7.1). The second coordinates agree by assumption on $\mathcal{V}$. $\Box$

Nesting forest languages. To explain the connection between wreath product and nesting of logical formulas, we provide a general notion of temporal logic, where the operators are given by regular forest languages.

Fix an alphabet A, and let $L_1, \ldots, L_k$ be forest languages that partition all forests over A. We can treat this partition as an alphabet B, with one letter per block L_i of the partition. The partition and alphabet are used to define a relabeling, which maps a forest

t over alphabet A to a forest $t[L_1, \ldots, L_k]$ over alphabet $A \times B$. The set of nodes in $t[L_1, \ldots, L_k]$ is the same as in t, except that each node x gets a label (a, L_i), where the first coordinate a is the old label of x in t, while the second coordinate L_i is the unique language L_i that contains the subforest of x in t. If L is a forest language over alphabet $A \times B$, we define $L\{L_1, \ldots, L_k\}$ to be the set of all forests t over alphabet A for which $t[L_1, \ldots, L_k] \in L$.

The operation of language composition is similar to formula composition. The definition below uses this intuition, in order to define a "temporal logic" based on operators given as forest languages. First however, we comment on a technical detail concerning alphabets. In the discussion below, a forest language is given by two pieces of information: the forests it contains, and the input alphabet. For instance, we distinguish between the set L_1 of all forests over alphabet $\{a\}$, and the set L_2 of all forests over the alphabet $\{a, b\}$ where b does not appear. The idea is that sometimes it is relevant to consider a language class $\mathscr{L}$ that contains L_1 but does not contain L_2 (although such classes will not appear in this particular paper). This distinction will be captured by our notion of language class: a language class is actually a mapping $\mathscr{L}$, which associates to each finite alphabet a class of languages over this alphabet.

Let $\mathscr{L}$ be a class of forest languages, which will be called the *language base*. The temporal logic with language base $\mathscr{L}$ is defined to be the smallest class $\mathsf{TL}[\mathscr{L}]$ of forest languages that contains $\mathscr{L}$, is closed under boolean operations and under language composition, i.e.

$$L_1, \ldots, L_k, L \in \mathsf{TL}[\mathscr{L}] \quad \Rightarrow \quad L[L_1, \ldots, L_k] \in \mathsf{TL}[\mathscr{L}].$$

Below we give examples of how the above definition can be used to describe some common temporal logics. The proofs in the examples are straightforward inductions.

- Consider the class $\mathscr{L}_{\mathsf{EF}}$ of languages "forests over alphabet A that contain some a", for every alphabet A and letter $a \in A$. Then $\mathsf{TL}[\mathscr{L}_{\mathsf{EF}}]$ is exactly the class of forest languages that can be defined by a forest formula of EF, as defined in Section 5.1.
- Consider the class $\mathscr{L}_{PT}$ of path testable languages, as defined in Section 6.2. Then $\mathsf{TL}[\mathscr{L}_{PT}]$ is exactly the class of forest languages that can be defined by a formula of the temporal logic PDL.
- Consider the subclass $\mathscr{L}_{LTL}$ of path testable languages, where the word languages $\mathsf{E}L$ in the definition of path testable languages are restricted LTL definable word languages. Then $\mathsf{TL}[\mathscr{L}_{LTL}]$ is exactly the class of forest languages that can be defined by a formula of the temporal logic CTL*.

Wreath product and nesting temporal formulas. We can now state the connection between wreath product of seminearrings and nesting of languages.

Theorem 7.2. *Let $\mathscr{U}$ be a class of seminearrings, and $\mathscr{L}$ the forest languages recognized by seminearrings in $\mathscr{U}$. Then $\mathsf{TL}[\mathscr{L}]$ is exactly the class of languages recognized by iterated wreath products of $\mathscr{U}$, i.e. by seminearrings in $\{\mathcal{U}_1 \circ \cdots \circ \mathcal{U}_n : \mathcal{U}_1, \ldots, \mathcal{U}_n \in \mathscr{U}\}$.*

Corollary 7.3. *A forest language is definable in PDL if and only if it is recognized by an iterated wreath product of additively idempotent semirings. Likewise for CTL*, with the additional requirement that the semirings are multiplicatively aperiodic.*

The good thing about Corollary 7.3 is that it connects three concepts from different areas: temporal logic, wreath products, and semirings. The bad thing is that it does not really give any insight into the structure of the *syntactic* seminearrings of languages from PDL and CTL*. All we know is that these syntactic seminearrings are quotients of iterated wreath products; but the whole structure of wreath product gets lost in a quotient.

References. In this section, we talked about wreath products of seminearrings. Similar operations have been studied for all the other algebraic structures. For A-algebras, one can consider cascade product, as studied in [12, 2]. For preclones, the more powerful block product was studied in [14], one can also use cascade/wreath product [12]. For forest algebras, the natural product seems to be wreath product, as studied in [8]. In all cases, there is a strong connection with nesting of temporal formulas (an exception is the block product, which is better suited to simulating quantifiers in first-order logic).

The definition of language nesting is based on notions introduced by Ésik in [11], and identical to the definition in [8].

References

[1] M. Benedikt and L. Segoufin. Regular tree languages definable in FO. In *STACS*, pages 327–339, 2005.

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

[3] M. Bojanczyk. Two-way unary temporal logic over trees. In *LICS*, pages 121–130, 2007.

[4] M. Bojanczyk and T. Colcombet. Tree-walking automata cannot be determinized. *Theor. Comput. Sci.*, 350(2-3):164–173, 2006.

[5] M. Bojanczyk and T. Idziaszek. Algebra for infinite forests with an application to the temporal logic EF. In *CONCUR*, pages 131–145, 2009.

[6] M. Bojanczyk and L. Segoufin. Tree languages defined in first-order logic with one quantifier alternation. In *ICALP (2)*, pages 233–245, 2008.

[7] M. Bojanczyk, L. Segoufin, and H. Straubing. Piecewise testable tree languages. In *LICS*, pages 442–451, 2008.

[8] M. Bojanczyk, H. Straubing, and I. Walukiewicz. Wreath products of forest algebras, with applications to tree logics. In *LICS*, pages 255–263, 2009.

[9] M. Bojanczyk and I. Walukiewicz. Characterizing EF and EX tree logics. *Theor. Comput. Sci.*, 358(2-3):255–272, 2006.

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

[11] Z. Ésik. Characterizing CTL-like logics on finite trees. *Theor. Comput. Sci.*, 356(1-2):136–152, 2006.

[12] Z. Ésik and S. Iván. Products of tree automata with an application to temporal logic. *Fundam. Inform.*, 82(1-2):61–78, 2008.

[13] Z. Ésik and S. Iván. Some varieties of finite tree automata related to restricted temporal logics. *Fundam. Inform.*, 82(1-2):79–103, 2008.

[14] Z. Ésik and P. Weil. Algebraic recognizability of regular tree languages. *Theor. Comput. Sci.*, 340(1):291–321, 2005.

[15] U. Heuter. Definite tree languages. *Bulletin of the EATCS*, 35:137–142, 1988.

[16] U. Heuter. *Zur Klassifizierung regulaerer Baumsprachen*. PhD thesis, RWTH Aachen, 1989.

[17] U. Heuter. First-order properties of trees, star-free expressions, and aperiodicity. *ITA*, 25:125–146, 1991.

[18] J. A. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Univ. of California, Los Angeles, 1968.

[19] R. McNaughton and S. A. Papert. *Counter-Free Automata (M.I.T. research monograph no. 65)*. The MIT Press, 1971.

[20] J.-E. Pin and H. Straubing. Some results on $\mathcal{C}$-varieties. *ITA*, 39(1):239–262, 2005.

[21] T. Place. Characterization of logics over ranked tree languages. In *CSL*, pages 401–415, 2008.

[22] T. Place and L. Segoufin. A decidable characterization of locally testable tree languages. In *ICALP (2)*, pages 285–296, 2009.

[23] A. Potthoff. *Logische Klassifizierung regulärer Baumsprachen*. PhD thesis, Institut für Informatik und Praktische Mathematik, Universität Kiel, 1994. Bericht Nr.9410.

[24] A. Potthoff. Modulo-counting quantifiers over finite trees. *Theor. Comput. Sci.*, 126(1):97–112, 1994.

[25] A. Potthoff and W. Thomas. Regular tree languages without unary symbols are star-free. In *FCT*, pages 396–405, 1993.

[26] M. Steinby. A theory of tree language varieties. In *Tree Automata and Languages*, pages 57–82. 1992.

[27] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[28] W. Thomas. Logical aspects in the study of tree languages. In *CAAP*, pages 31–50, 1984.

[29] T. Wilke. An algebraic characterization of frontier testable tree languages. *Theor. Comput. Sci.*, 154(1):85–106, 1996.