# A Complexity Approach to Tree Algebras: the Polynomial Case

Thomas Colcombet, Arthur Jaquard

# A Complexity Approach to Tree Algebras: the Polynomial Case

**Thomas Colcombet**
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

**Arthur Jaquard**
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

## Abstract

In this paper, we consider infinitely sorted tree algebras recognising regular language of finite trees. We pursue their analysis under the angle of their asymptotic complexity, i.e. the asymptotic size of the sorts as a function of the number of variables involved.

Our main result establishes an equivalence between the languages recognised by algebras of polynomial complexity and the languages that can be described by nominal word automata that parse linearisation of the trees. On the way, we show that for such algebras, having polynomial complexity corresponds to having uniformly boundedly many orbits under permutation of the variables, or having a notion of bounded support (in a sense similar to the one in nominal sets).

We also show that being recognisable by an algebra of polynomial complexity is a decidable property for a regular language of trees.

**2012 ACM Subject Classification** Theory of computation → Tree languages; Theory of computation → Regular languages

**Keywords and phrases** Tree algebra, nominal automata, language theory

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2022.37

**Funding** *Thomas Colcombet*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624) and the DeLTA ANR project (ANR-16-CE40-0007).

## 1 Introduction

Among the many different approaches to language theory, the algebraic one focalises toward the understanding of the expressive power of regular languages based on the properties of algebraic recognizers. The first work in this direction [11] characterized star-free languages, and initiated a very fruitful branch of research. While algebraic theories for word languages (both finite and inifinite) are already well developed, the corresponding picture remains incomplete for e.g. languages of trees (both finite or infinite) or graphs. Finding an effective characterization of the regular languages of trees definable in first order logic remains for instance a long standing open problem.

When designing algebras for tree languages or graph languages, one is naturally inclined to consider infinitely sorted algebras. The case of tree algebras (such as preclones, $\omega$-hyperclones, operads [8, 1]) is typical: plugging a subtree into another one requires a mechanism for identifying the leaf/leaves in which the substitution has to be performed. Notions such as variables, hole types, or colors are used for that. Another example is the one of graphs (HR- and VR- algebras [7]) in which basic operations (a) glue graphs together using a set of colors (sometimes called ports) for identifying the glue-points, or (b) add all possible edges between vertices of fixed given colors. In these examples, the algebras are naturally sliced into infinitely many sorts based on the number of variables/hole types/colors that are used simultaneously.


However, a technical difficulty arises immediately when using such algebras. Even when all sorts are finite (what we call a finite algebra), these algebras are not really finite due to the infinite number of sorts. This forbids, for instance, to explicitely describe the whole algebra in a finite way, which is of course a problem for designing algorithms. This hurdle to handle infinitely sorted algebras can, arguably, be seen as one of the causes of the many years that it took before having a good definition of an algebra for infinite trees [2], or the time that it took before it was possible to characterize logically the expressiveness of recognizable properties of graphs under bounded tree-width hypothesis [4].

**Complexity of algebras.** To cope with this difficulty, the notion of complexity for infinitely sorted algebras was introduced in [5]. In each of the above cases, the sorts are naturally indexed by a natural number parameter: the number of variables, or hole types, or colors. Hence an algebra $\mathcal{A}$ would have a carrier of the form

$$(A_n)_{n\in\mathbb{N}}$$

together with suitable operations that depend on the particular algebra type. Such an algebra is called finite if all the $A_n$ are finite and, in this case, the complexity map $c_{\mathcal{A}} \colon \mathbb{N} \to \mathbb{N}$ of the algebra is defined as:

$$c_{\mathcal{A}}(n) = |A_n| \ , \qquad \text{for all } n \in \mathbb{N}.$$

This approach gives rises to the classification of finite algebras depending on the asymptotic growth of $c_{\mathcal{A}}$: algebras can have bounded complexity, polynomial complexity, etc. It is then possible to study the expressive power of algebras in a prescribed complexity class.

In all of the mentioned examples of algebras, there is a natural operation that performs a renaming of the variables/hole types/colors. This renaming is parameterized by a bijection over variables/hole types/colors, and this permutation acts on the corresponding sort. Thus, there is an action of the symmetric group over $n$ elements, $\mathbf{Sym}(n)$ over $A_n$. The orbit-complexity map $c^{\circ} \colon \mathbb{N} \to \mathbb{N}$ is then naturally defined as:

$$c^{\circ}_{\mathcal{A}}(n) = |A_n/\mathbf{Sym}(n)| \ , \qquad \text{for all } n \in \mathbb{N}.$$

The notions of bounded orbit-complexity, polynomial orbit-complexity, etc. are then defined accordingly.

Along with the definitions of complexity and orbit-complexity, a precise description of the languages recognized by tree algebras of bounded complexity was given in [5][1]. In this article we endeavor to study tree algebras of polynomial complexity.

**Tree algebras.** The notion of tree algebra that we presented above is a bit unpractical, because the variables/hole types/colors are unnamed (we will simply call them variables from now on). We instead consider tree algebras with a carrier of the form

$$(A_X)_{X \text{ finite set of variables}}$$

for which the notions of complexity and orbit-complexity can be easily adapted. Given a finite $X$, a tree is then seen as an element of $A_X$ whenever all the variables on the leaves of the tree are in $X$. This notion of tree algebras may have different flavors:

[1] Let us emphasize that the algebras used in [5] are different, since all variables are furthermore required to appear at least once. This apparently inocuous modification has consequences on the results.

- *Unrestrained tree algebras* in which there is no additional constraint, as in this paper.
- *Affine tree algebras*: all variables must appear on at most one leaf of the tree [2].
- *Relevant tree algebras*: all variables must appear on at least one leaf of the tree [5].
- *Linear tree algebras*: all variables must appear on exactly one leaf of the tree [8, 9].

We could also consider variants in which the variables are ordered. For instance, starting from linear tree algebras and adding the condition that, when read from left to right, the variables are monotone, corresponds to preclones [8]. The expressive power of relevant tree algebras of bounded complexity was precisely described in [5].

**Contributions of the article.** In this article, we study unrestrained tree algebras (or simply tree algebras from now on) of polynomial complexity.

We introduce a way to encode finite trees into data words, and thus to encode languages of trees into languages of data words (we say that the language of trees is described by the language of data words).

We then establish that tree algebras of polynomial complexity and of bounded orbit-complexity have the same expressive power, thus answering a question from [5] (for unrestrained tree algebras). Moreover, the languages that they recognize are exactly those described by regular languages over our data alphabet (in which the notion of regular language is defined using orbit-finite nominal automata [10, 3], a mild generalization of register automata that fit nicely into our algebraic setting).

Our main result is the following Theorem 1. It states the above described equivalences, and add a third item that will be formalised in the body of the paper.

▶ **Theorem 1.** *For a regular language of finite trees, the following properties are equivalent:*

1. *Being recognized by a finite tree algebra of polynomial complexity.*
2. *Being recognized by a finite tree algebra of bounded orbit complexity.*
3. *Being recognized by a finite tree algebra that has a bounded and stable system of supports.*
4. *Being described by a coding automaton.*

Our second theorem, Theorem 22, establishes the decidability of this class.

**Structure of the paper.** In Section 2, we recall some classical definitions, and introduce the notion of algebras. In Section 3, we explain our encoding of trees into data words, and prove Proposition 16 corresponding to the implication from Item 4 to Items 1 and 2 of Theorem 1. In Section 4, we look in detail at the properties of tree algebras of polynomial complexity and bounded orbit-complexity. Doing this, we prove Propositions 19 and 20, that correspond to proving implications from Items 1 and 2 to Item 3, and from Item 3 to Item 4 of Theorem 1. We also address the decidability question an prove Theorem 22. Section 5 concludes.

## 2 Definitions

We denote by $\mathbb{N}$ the set of all non-negative integers. Given $n \in \mathbb{N}$, we write $[n] = \{0, 1, ..., n-1\}$. The symmetric group (resp. alternating group) of a set $X$ is denoted $\mathbf{Sym}(X)$ (resp. $\mathbf{Alt}(X)$). We fix a finite *ranked alphabet* $\Sigma$; the *arity* of a *symbol* $a \in \Sigma$ is denoted $\mathrm{ar}(a)$. It is a *constant* if $\mathrm{ar}(a) = 0$, and is *unary* if $\mathrm{ar}(a) = 1$. For $k \in \mathbb{N}$, we set $\Sigma_k = \{a \in \Sigma \mid \mathrm{ar}(a) = k\}$. $A^*$ is the set of finite words over $A$, and $A^+ = A^* \setminus \{\varepsilon\}$.

## 2.1 Trees

In this section, we introduce notions and notations for trees.

We fix a countable set of *variables* $\mathcal{V}$. Given a finite set of variables $X$, a $\Sigma, X$-tree is, informally, a tree in which nodes are labelled by elements of $\Sigma$ and leaves also possibly by variables of $X$. Formally, a $\Sigma, X$-*tree* is a partial map $t\colon \mathbb{N}^* \to \Sigma \uplus X$ such that $\mathrm{dom}(t)$ is non-empty and prefix-closed, and furthermore, for every $u \in \mathrm{dom}(t)$ there exists $n \in \mathbb{N}$ such that $\{i \mid ui \in \mathrm{dom}(t)\} = [n]$, and

- either $t(u) \in \Sigma_n$ (*symbol node*), or
- $t(u) \in X$ and $n = 0$ (*variable node*). Note that a variable node is always a leaf.

$\Sigma, \emptyset$-trees are simply called $\Sigma$-*trees*. The elements in $\mathrm{dom}(t)$ are called *nodes*. The prefix relation over nodes is called the *ancestor relation*. The node $\varepsilon$ is called the *root* of the tree. The tree $t$ is *finite* if it has finitely many nodes. A *branch* of a tree $t$ is a maximal set of nodes ordered under the ancestor relation. Let $\mathrm{Trees}(\Sigma, X)$ bet the set of finite $\Sigma, X$-trees, for every finite set of variables $X$.

▶ Remark 2. Note that $\Sigma, X$-trees are also $\Sigma, Y$-trees whenever $X \subseteq Y$. This is in contrast with [5] in which all variables were assumed to appear at least once.

**Building trees.** We introduce now some operations on trees. See Fig. 1.

- $\varepsilon_x$, where $x$ is a variable, denotes the $\Sigma, \{x\}$-tree consisting of a single root node labelled $x$.
- $a(x_0, \ldots, x_{n-1})$, for $x_0, \ldots, x_{n-1}$ variables and $a \in \Sigma_n$, denotes the $\Sigma, \{x_0, \ldots, x_{n-1}\}$-tree consisting of a root labelled $a$, and children $0, \ldots, n-1$ labelled with variables $x_0, \ldots, x_{n-1}$ respectively.
- $s \cdot_x t$, for two trees $s \in \mathrm{Trees}(\Sigma, X)$, $t \in \mathrm{Trees}(\Sigma, Y)$ and a variable $x \in \mathcal{V}$, is the $\Sigma, (X \setminus \{x\}) \cup Y$-tree $s$ in which $t$ is substituted for every occurrence of the variable $x$, which may not be present in $s$ at all.
- $\widetilde{\sigma}(t)$, for a tree $t \in \mathrm{Trees}(\Sigma, X)$ and a map $\sigma\colon X \to Y$, is the $\Sigma, Y$-tree obtained as $t$ in which variable $\sigma(x)$ has been substituted to $x$ for every $x \in X$. Note that $\widetilde{\sigma} \circ \widetilde{\tau} = \widetilde{\sigma \circ \tau}$.
- $t[x_0 \leftarrow t_0, \ldots, x_{n-1} \leftarrow t_{n-1}]$ denotes the tree of sort $X \setminus \{x_0, \ldots, x_{n-1}\} \cup \bigcup_i Y_i$ obtained from $t$ by simultaneously substituting the tree $t_i$ for the variable $x_i$ for every $i \in [n]$, where $t$ is a tree of sort $X$, $x_0, \ldots, x_{n-1} \in \mathcal{V}$, and $t_0, \ldots, t_{n-1}$ are trees of sort $Y_i$ for every $i \in [n]$. Note that this operation is equivalent to a combination of the previous ones.
- $a(t_0, ..., t_{n-1})$, for $a \in \Sigma_n$, denotes the tree of root $a$ and children $t_0, \ldots, t_{n-1}$ at respective positions $0, \ldots, n-1$. Again, this operation is equivalent to a combination of the previous ones.

▶ **Example 3.** Throughout this paper, we use the map $\mathrm{create}_x^X : X \to X \cup \{x\}$, acting as the identity, in which $X$ is a finite set of variables and $x \in \mathcal{V}$. $\widetilde{\mathrm{create}_x^X}$ is thus a mapping from $\mathrm{Trees}(\Sigma, X)$ to $\mathrm{Trees}(\Sigma, X \cup \{x\})$ that maps every tree to itself. Most of the time, we will simply write $\mathrm{create}_x(t)$ when $X$ is clear from the context.

▶ **Lemma 4.** *All finite trees can be obtained from the trees of the form $\varepsilon_x$ and $a(x_0, \ldots, x_{n-1})$ using the operations "$\cdot$".*

**Expressions denoting finite trees.** For $X$ a finite set of variables, a *tree-expression of sort $X$ (over the alphabet $\Sigma$)* is an expression built inductively as follows:

- $\varepsilon_x$ is a tree-expression of sort $\{x\}$ for every variable $x$,
- $a(x_0, \ldots, x_{n-1})$ is a tree-expression of sort $\{x_0, \ldots, x_{n-1}\}$ for every symbol $a \in \Sigma_n$,

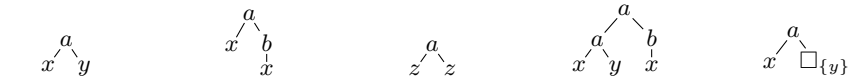


**Figure 1** Trees and contexts with their notations. Here $\sigma(x) = \sigma(y) = z$.

- $S \cdot_x T$ is a tree-expression of sort $X \setminus \{x\} \cup Y$ for all tree-expressions $S$ of sort $X$, all tree-expressions $T$ of sort $Y$, and all variables $x \in \mathcal{V}$ (*substitution*),
- $\tilde{\sigma}(T)$ is a tree-expression of sort $Y$ for all tree-expressions $T$ of sort $X$, and map $\sigma\colon X \to Y$ (*renaming*). Note that $\sigma$ needs not be bijective here.

For a tree-expression $T$ of sort $X$, $[\![T]\!]$ denotes its evaluation into a finite $\Sigma, X$-tree using the operations of substitution and renaming.

**Contexts.** We define now contexts, which are terms with a specific leaf called the hole. Since we work in a multi-sorted algebra, the hole itself has a sort. Essentially, to a hole of sort $X$ will be substituted a term of sort $X$. Formally, for fixed finite set of variables $Y$, a *context of sort $X$ with hole of sort $Y$* (or simply a context) is defined inductively as a tree expression of sort $X$, using the extra construction $\Box_Y$ (the *hole of sort $Y$*) which is a context of sort $Y$ with hole of sort $Y$. This new construction must appear exactly once in a context.

For $C$ a context of sort $X$ with hole of sort $Y$, $[\![C]\!]\colon \mathrm{Trees}(\Sigma, Y) \to \mathrm{Trees}(\Sigma, X)$ is the function which to a tree of sort $Y$ $t$ associates the tree of sort $X$ obtained by evaluating the operations as above, interpreting $\Box_Y$ as $t$.

## 2.2 Finite tree algebras

Our notion of tree algebra is the natural notion associated to finite trees equipped with the above operations. We give here a more formal definition, though the detail of identities is more for reference. What matters is that it is defined such that the free algebra coincides with finite trees. A *tree algebra* $\mathcal{A}$ consists of an infinite collection of carrier sets $A_X$ indexed by finite sets of variables $X$, together with operations:

- $\varepsilon_x^{\mathcal{A}} \in A_{\{x\}}$ for every variable $x$,
- $a(x_0, \dots, x_{n-1})^{\mathcal{A}} \in A_{\{x_0,\dots,x_{n-1}\}}$ for all $a \in \Sigma_n$ and variables $x_0, \dots, x_{n-1}$,
- $\cdot_x^{\mathcal{A}}\colon A_X \times A_Y \to A_{X\setminus\{x\}\cup Y}$ for all finite sets of variables $X, Y$ and $x \in \mathcal{V}$,
- $\sigma^{\mathcal{A}}\colon A_X \to A_Y$ for every renaming $\sigma\colon X \to Y$,

that satisfy the expected identities, i.e. the ones guaranteeing that several ways to describe the same tree yield the same evaluation in the algebra. Formally, for all $s, t, u$ that belong to $A_X$, $A_Y$, $A_Z$ respectively,

- $\varepsilon_x \cdot_x t = t$ for every $x \in \mathcal{V}$,
- $t \cdot_x \varepsilon_x = t$ for every $x \in Y$,
- $(s \cdot_x^{\mathcal{A}} t) \cdot_y^{\mathcal{A}} u = s \cdot_x^{\mathcal{A}} (t \cdot_y^{\mathcal{A}} u)$ for all $x \in X$ and $y \in X \setminus Y$ (horizontal associativity), and
- $(s \cdot_x^{\mathcal{A}} t) \cdot_y^{\mathcal{A}} u = s \cdot_x^{\mathcal{A}} (t \cdot_y^{\mathcal{A}} u)$ for all $x \in X$ and $y \in Y \setminus X \cup \{x\}$ (vertical associativity),

for all $s, t$ that belong to $A_X$, $A_Y$, $x \in X$ and renaming $\sigma\colon X \to Y$ ,

- $\sigma^{\mathcal{A}}(s \cdot_x^{\mathcal{A}} t) = \sigma^{\mathcal{A}}(s) \cdot_x^{\mathcal{A}} t$ if $\sigma^{-1}(\sigma(x)) = \{x\}$ and $\sigma(y) = y$ for every $y \in X \cap Y \setminus \{x\}$,
- $\sigma^{\mathcal{A}}(s \cdot_x^{\mathcal{A}} t) = s \cdot_x \sigma^{\mathcal{A}}(t)$ if $\sigma(y) = y$ for every $y \in X \cap Y \setminus \{x\}$,

for all maps $\sigma\colon X \to Y$ and $\tau\colon Y \to Z$, $(\tau \circ \sigma)^{\mathcal{A}} = \tau^{\mathcal{A}} \circ \sigma^{\mathcal{A}}$, and for all maps $\sigma\colon \{x_0, \dots x_{n-1}\} \to Y$ and $a \in \Sigma_n$, $\sigma^{\mathcal{A}}(a(x_0, \dots, x_{n-1})^{\mathcal{A}}) = a(\sigma(x_0), \dots, \sigma(x_{n-1}))^{\mathcal{A}}$. In practice, we shall not explicitly use these identities, and simply write two elements of the algebra equal as soon as they obviously come from expressions denoting the same trees. A tree algebra is *finite* if the $A_X$'s are all finite.

A *morphism of tree algebras* from $\mathcal{A}$ to $\mathcal{B}$ is a family of maps $\alpha_X \colon A_X \to B_X$ for every finite sets of variables $X$ which preserves all operations, i.e. $\alpha_Y(\sigma^{\mathcal{A}}(s)) = \sigma^{\mathcal{B}}(\alpha_X(s))$ for every map $\sigma \colon X \to Y$, $\alpha(a(x_0, \dots, x_{n-1})^{\mathcal{A}}) = a(x_0, \dots, x_{n-1})^{\mathcal{B}}$, and $\alpha_{X \setminus \{x\} \cup Y}(s \cdot_x^{\mathcal{A}} t) = \alpha_X(s) \cdot_x^{\mathcal{B}} \alpha_Y(t)$ for all $s \in A_X$, $t \in A_Y$ and $x \in \mathcal{V}$.

The $\mathrm{Trees}(\Sigma, X)$ sets equipped with the operations of substitution and renaming form a tree algebra (it is the free tree algebra generated by $\emptyset$). For $\mathcal{A}$ a tree algebra, its associated *evaluation morphism* is the unique morphism from $\mathrm{Trees}(\Sigma)$ to $\mathcal{A}$.

A *congruence* $\sim$ over a tree algebra $\mathcal{A}$ is a family $\sim$ of equivalence relations over the $A_X$'s (each denoted $\sim$) such that, for any $a \sim b \in A_X$, $c \sim d \in A_Y$, $y \in Y$ and $\sigma \colon X \to Y$: $c \cdot_y a \sim c \cdot_y b$; $c \cdot_y a \sim d \cdot_y a$ and $\widetilde{\sigma}(a) \sim \widetilde{\sigma}(b)$. From such a congruence, one can define the *quotient algebra* $\mathcal{A}/\sim$ in the natural way.

## 2.3 Languages and syntactic algebras

A *language of finite $\Sigma$-trees* $L$ is a set of $\Sigma$-trees. It is *recognized by* a tree algebra $\mathcal{A}$ if there is a set $P \subseteq A_\emptyset$ such that $L = \alpha^{-1}(P)$ in which $\alpha$ is the evaluation morphism of $\mathcal{A}$.

The *syntactic congruence $\sim_L$ of a language* $L$ of finite $\Sigma$-trees is defined in the following way $s \sim_L t$ for $s, t$ finite $\Sigma$-trees if, for every context $C$, $[\![C]\!](s) \in L$ if and only if $[\![C]\!](t) \in L$. It is easy to prove that $\sim_L$ is indeed a congruence. The quotient algebra $\mathrm{Trees}(\Sigma)/_{\sim_L}$ is called the *syntactic algebra* of $L$, and this algebra recognizes $L$.

▶ **Example 5.** The language of all finite trees in which the symbol $a$ appears on the leftmost branch has for syntactic tree algebra the algebra with sorts $A_X = \{\bot, \top\} \uplus X$ for every finite set of variables $X$. Let $t$ be a $\Sigma, X$-tree, we define $\alpha$ as follows: $\alpha_X(t) = \top$ if there is an $a$ on the leftmost branch of $t$, $\alpha_X(t) = x$ if the leftmost branch of $t$ ends with an $x$ but contains no $a$, and $\alpha_X(t) = \bot$ otherwise. The operations of the algebra are defined so that $\alpha$ becomes the evaluation morphism.

## 2.4 Complexity

Following [5], we define the notion of complexity and of orbit-complexity of a tree algebra $\mathcal{A}$.

**Complexity.** We start by highlighting the fact that any bijection $\sigma$ between finite sets of variables $X$ and $Y$ induces a bijection $\widetilde{\sigma}$ between $A_X$ and $A_Y$. As such, it is meaningful to define the *complexity map* of the algebra $c_{\mathcal{A}} \colon \mathbb{N} \to \mathbb{N}$ as follows:

$$c_{\mathcal{A}}(|X|) = |A_X| \ , \quad \text{for every finite set of variables } X.$$

A tree algebra $\mathcal{A}$ has *bounded complexity* if $c_{\mathcal{A}}$ is bounded. It has *polynomial complexity* if there is a polynomial $P$ such that $c_{\mathcal{A}}(n) \leqslant P(n)$ for every $n \in \mathbb{N}$.

**Orbit-complexity.** Similarly, we define the *orbit-complexity map* $c^\circ_{\mathcal{A}} \colon \mathbb{N} \to \mathbb{N}$ as:

$$c^\circ_{\mathcal{A}}(|X|) = |A_X/\mathbf{Sym}(X)| \ , \quad \text{for every finite set of variables } X,$$

in which $A_X/\mathbf{Sym}(X)$ is the set of all orbits of $A_X$ under the action of $\mathbf{Sym}(X)$. A tree algebra $\mathcal{A}$ has *bounded orbit-complexity* if $c^\circ_{\mathcal{A}}$ is bounded.

▶ **Example 6.** The tree algebra $\mathcal{A}$ from Example 5 has polynomial complexity and bounded orbit-complexity: $c_{\mathcal{A}}(n) = n + 2$ and $c^\circ_{\mathcal{A}}(n) = 3$ for every $n \in \mathbb{N}$.

To check whether a language of trees is recognized by an algebra with a prescribed complexity, one only needs to look at its syntactic algebra.

▶ **Lemma 7.** *If $L$ is a language of $\Sigma$-trees recognized by a tree algebra $\mathcal{B}$, then its syntactic tree algebra $\mathcal{A}$ has lower complexity:*

$$c_{\mathcal{A}}(n) \leqslant c_{\mathcal{B}}(n) \quad \text{and} \quad c^{\circ}_{\mathcal{A}}(n) \leqslant c^{\circ}_{\mathcal{B}}(n) \ , \ \text{for all } n \in \mathbb{N}.$$

## 2.5 Tree automata

A *tree automaton* $\mathcal{B} = (Q, I, (\delta_a)_{a\in\Sigma})$ over $\Sigma$ has a finite set $Q$ of *states*, a set of *accepting states* $I \subseteq Q$ and a *transition relation* $\delta_a \subseteq Q \times Q^{\mathrm{ar}(a)}$ for every symbol $a \in \Sigma$. A *run of* $\mathcal{B}$ over a finite tree $t$ is a mapping $\rho\colon \mathrm{dom}(t) \to Q$ such that, for any vertex $u \in \mathrm{dom}(t)$ with $t(u) = a \in \Sigma$, $(\rho(u), (\rho(u0), ..., \rho(u(\mathrm{ar}(a) - 1)))) \in \delta_a$. A run is *accepting* if $\rho(\varepsilon) \in I$. A language $L$ of finite trees is called *regular* if it is recognized by a tree automaton $\mathcal{B}$, meaning the trees in $L$ are exactly those for which there is an accepting run in $\mathcal{B}$.

Example 8 below shows the translation from tree automata to tree algebra.

▶ **Example 8** (Automaton algebra)**.** Consider a regular language $L$ of finite trees recognized by the tree automaton $\mathcal{B} = (Q, q_0, (\delta_a)_{a\in\Sigma})$. Consider some finite set of variables $X$. An *$X$-run profile* is a tuple $\tau \in Q \times \mathcal{P}(Q)^X$. For a $\Sigma, X$-tree $t$, $\tau = (p, (U_x)_{x\in X})$ is a *run profile over $t$* if there exists a run $\rho$ of the automaton over $Q$ such that $\rho(\varepsilon) = p$ and for every variables $x \in X$, $U_x$ is the set of states assumed by $\rho$ at leaves labelled $x$. We define a tree algebra $\mathcal{A}$ that has as elements of sort $X$ sets of $X$-run profiles. The definition of the operations is natural, and is such that the image of a $\Sigma, X$-tree $t$ under the evaluation morphism yields the set of run profiles over $t$. It naturally recognizes the language $L$.

Note that this definition yields an algebra of doubly exponential complexity (and hence, this is an upper bound for regular languages). Of course, in practice, one can restrict the algebra to the reachable elements, and this may dramatically reduce the complexity.

The converse translation is also true (it is for instance proved for preclones in [8]), yielding the following result.

▶ **Proposition 9.** *A finite tree language is regular if and only if it is recognized by a finite algebra. Moreover, every regular tree language is recognized by an algebra of doubly exponential complexity.*

## 2.6 Group actions and orbit-finite sets

To conclude this list of definitions, we recall some notions on group actions.

**Group actions.** A (left) group action of a group $G$ on a set $X$ is given by a function $\cdot\colon G \times X \to X$ such that $e \cdot x = x$ and $\sigma \cdot (\tau x) = (\sigma\tau) \cdot x$ for all $x \in X$ and $\sigma, \tau \in G$, where $e$ is the neutral element of $G$. A set $X$ equipped with such an action is called a *$G$-set*.

**Orbits.** For every $x$ in a $G$-set $X$, the set $G \cdot x = \{\sigma \cdot x \mid \sigma \in G\}$ is called the *orbit* of $x$. A $G$-set is partitioned by the orbits of its elements, and it is said to be *orbit-finite* whenever it has only a finite number of different orbits.

**Equivariant subsets and relations.** Given a $G$-set $X$, a subset $Y$ of $X$ is *equivariant* if $\sigma \cdot Y = Y$ for every $\sigma \in G$. Accordingly, a relation $R \subseteq X \times Y$ between $G$-sets $X$ and $Y$ is equivariant if it is an equivariant subset of the $G$-set $X \times Y$ equipped with the point-wise action of $G$. In particular, a function $f\colon X \to Y$ is equivariant exactly when $f(\sigma \cdot x) = \sigma \cdot f(x)$ for all $x \in X$ and $\sigma \in G$.

**Support and nominal sets.** From now on, we will only be looking at the group $G = \mathbf{Sym}(\mathcal{V})$. Consider then a $\mathbf{Sym}(\mathcal{V})$-set $X$. A set $S \subseteq \mathcal{V}$ *supports* an element $x \in X$ if $\sigma \cdot x = x$ for every $\sigma \in \mathbf{Sym}(\mathcal{V} \setminus S)$, where $\mathbf{Sym}(\mathcal{V} \setminus S)$ is seen as a subgroup of $\mathbf{Sym}(V)$. A $\mathbf{Sym}(V)$-set is called a *nominal* set if all of its elements are supported by a finite set.

Given an element $x \in X$ that is finitely supported, it admits a least support. The *least support* of an element $x$ from a nominal set $X$ will be denoted $\mathrm{supp}(x)$.

Whenever $A$ and $B$ are $\mathbf{Sym}(\mathcal{V})$-sets, the set of all functions from $A$ to $B$ may be equipped with the action that maps $f\colon A \to B$ to $\sigma \cdot f$ defined for every $a \in A$ by $(\sigma \cdot f)(a) = \sigma \cdot f(\sigma^{-1} \cdot a)$. When seen in this way as a $\mathbf{Sym}(\mathcal{V})$-set, a function $f\colon A \to B$ is supported by $X \subseteq \mathcal{V}$ whenever $f(\sigma \cdot x) = \sigma \cdot f(x)$ for every $\sigma \in \mathbf{Sym}(\mathcal{V} \setminus X)$.

# 3 Nominal word automata for tree languages

We start our presentation by showing how nominal word automata can be used to describe regular languages of finite trees. To this end, we first explain in Section 3.1, how to encode trees into data words. In Section 3.2, we exploit this to interpret data languages as tree languages, using the notion of coding automata. In Section 3.3 we establish Proposition 16 stating that languages of trees that are described by coding automata are recognized by tree algebras of polynomial complexity and bounded orbit complexity (thus proving the implication from Item 4 to Items 1 and 2 of Theorem 1).

## 3.1 Coding languages

In this section, we show how to encode finite trees into data words.

**Coding alphabet.** We begin by defining the nominal alphabet used for this encoding:

- Let $C_{\mathcal{V}}$ be the set of elements $[x]$ for $x \in \mathcal{V}$.
- Let $C_{\mathcal{V},\Sigma}$ be the set of elements $[\cdot_x a(x_0, ..., x_{n-1})]$ for $a \in \Sigma_n$ and $x, x_0, ..., x_{n-1} \in \mathcal{V}$.
- Let $C = C_{\mathcal{V}} \uplus C_{\mathcal{V},\Sigma}$. It is called the *coding alphabet*.

The coding alphabet is naturally made into a $\mathbf{Sym}(V)$-set, by defining, for every $\sigma \in \mathbf{Sym}(\mathcal{V})$, $\sigma[x] = [\sigma(x)]$ and $\sigma[\cdot_x a(x_0, ..., x_{n-1})] = [\cdot_{\sigma(x)} a(\sigma(x_0), ..., \sigma(x_{n-1}))]$ for all $[x]$ and $[\cdot_x a(x_0, ..., x_{n-1})]$ in the coding alphabet. It is obviously both orbit-finite and nominal.

**Tree coding.** A *coding* is a word in $\mathrm{Codings} = C_{\mathcal{V}} C_{\mathcal{V},\Sigma}^*$. We shall describe now how codings can be evaluated to trees. This is natural since, forgetting the bracket notation, codings can be seen as tree expressions. The *evaluation* $T(c)$ of a coding $c$ is defined as follows: $T([x]) = x$, where $[x] \in C_{\mathcal{V}}$ and $T(c[\cdot_x a(x_0, ..., x_{n-1})]) = \mathrm{create}_x(T(c)) \cdot_x a(x_0, ..., x_{n-1})$, where $c$ is a tree coding and $[\cdot_x a(x_0, ..., x_{n-1})] \in C_{\mathcal{V},\Sigma}$.

▶ **Example 10.** According to the definition of evaluation, $T([x][\cdot_x a(x, y)]) = x \cdot_x a(x, y) = a(x, y)$. Similarly, $T([x][\cdot_x a(x, y)][\cdot_x c][\cdot_x d]) = ((x \cdot_x a(x, y)) \cdot_x c) \cdot_y d = a(c, d)$.

We will be particularly interested in codings $c$ that evaluate to trees without variables (meaning $T(c) \in \mathrm{Trees}(\Sigma, \emptyset)$). Let $\mathrm{Codings}_\emptyset$ be the set of these codings that evaluate to variable-less trees.

**Describing languages of trees.** A language of codings $K$ is said to *describe* a language $L$ of trees without free variables if, for every coding $c \in \mathrm{Codings}_\emptyset$, $c \in K$ if and only if $T(c) \in L$. The crucial point in this definition is that the language $K$ may also contain codings that do not evaluate to a tree without free variables.

▶ **Example 11.** Let $L$ be a language of trees without free variables, and let $K$ be the language of all codings that evaluate to trees in $L$. Then both $K$ and $K' = K \cup \{[x][\cdot_x a(x,y)]\}$ describe $L$. That is because $[x][\cdot_x a(x,y)] \notin \mathrm{Codings}_\emptyset$.

However, not all languages of codings describe tree languages.

▶ **Example 12.** Let $\Sigma = \Sigma_0 \cup \Sigma_2$ with $\Sigma_0 = \{c\}$ and $\Sigma_2 = \{a, b\}$, and consider the language $K$ of codings in which the third letter is $[\cdot_x c]$ for any choice of $x \in \mathcal{V}$.

Let $c = [x][\cdot_x a(x,y)][\cdot_y c][\cdot_x a(y,y)][\cdot_y c]$ and $c' = [x][\cdot_x a(x,y)][\cdot_x a(y,y)][\cdot_y c]$, then $c$ is in $K$ but not $c'$. Note however that $T(c) = T(c') = a(a(c,c),c)$, and thus $K$ does not describe a language of trees.

## 3.2 Coding automata

Let us now look at acceptance of coding languages by automata. We start by recalling the notion of nominal automata [10], which is used to recognize data languages over orbit-finite nominal alphabets.

**Nominal automaton.** A *deterministic $G$-automaton* is given by
- an orbit-finite $G$-set $A$ (the alphabet),
- a $G$-set $Q$ (the states),
- an empty supported $q_0 \in Q$ called the *initial state*,
- an equivariant subset $F \subseteq Q$ of *final states*,
- and an equivariant function $\delta \colon Q \times A \to Q$ called the *transition function*.

*Acceptance* of a word $w \in A^*$ is then defined in the standard way. A deterministic $G$-automaton is called *orbit-finite* whenever $Q$ is. It is called *nominal* when $G = \mathbf{Sym}(V)$ and when $A$ and $Q$ are both nominal sets. In this paper, we only consider *orbit-finite nominal automata*.

**Coding automata.** An orbit-finite nominal automaton $\mathcal{A}$ over the coding alphabet is called a *coding automaton* if it recognizes a language that describes a tree language. We also assume that there is no transition toward the initial state. The *tree language described* by a coding automaton is the language $L$ of trees without free variables described by the language $K$ recognized by $\mathcal{A}$. In other words, it is the language

$$L = \{T(c) \mid c \in \mathrm{Codings}_\emptyset,\ \mathcal{A} \text{ accepts } c\}\ .$$

▶ **Example 13.** Let $\Sigma = \Sigma_0 \cup \Sigma_2$ with $\Sigma_0 = \{c\}$ and $\Sigma_2 = \{a, b\}$, and consider the language $L$ of trees such that the total number of nodes labelled $a$ appearing on the leftmost branch and the rightmost branch is even. We give a coding automaton that describes $L$. Its set of states is $Q = \{q_0\} \uplus \{\varepsilon(x) \mid x \in \mathcal{V}\} \uplus \{i(x,y) \mid i \in [2], x, y \in \mathcal{V} \cup \{*\}\}$, and the action of $\mathbf{Sym}(V)$ on $Q$ is the one naturally obtained by permuting the variables. Its transitions are defined in the natural way so that:

- $q_0$ is the initial state,
- $\varepsilon(x)$ is reached from $q_0$ by reading $[x]$,
- $\delta(q_0, c) = i(x, y)$, in which $i$ is the total number of nodes labelled $a$ on the leftmost and rightmost branches of $t$ modulo 2, and $x$ (resp. $y$) is the variable on the leaf of the leftmost (resp. rightmost) branch ($*$ if there is no such variable), for $t = T(c)$.

Setting the only accepting state to be $0(*, *)$, this automaton describes $L$.

A coding automaton is thus a device that takes as input a top-down description of a tree (a coding), and that decides its belonging to a language while remembering only boundedly many variables. Intuitively, such a device can only remember what happens along boundedly many branches. The subtlety of the model is that it must always yield the same result for a given tree, nonwithstanding the actual coding that was provided.

**Minimization.** Coding automata can be minimized, in the sense that a coding automaton can be effectively turned into another one that describes the same tree language and is minimal. This property turns out to be key to prove Theorem 1. The construction, though similar to the classical one for minimizing nominal automata recognizing a word language, is not the same. One subtlety is that in our case, there are states in the automaton that may accept both codings in $\text{Codings}_\emptyset$ and codings outside of $\text{Codings}_\emptyset$. Standard constructions are not able to cope with such a phenomenon.

We start by defining the *Myhill-Nerode relation* (or *congruence*) of a tree language $L$ over codings by $c \equiv_L c'$, for codings $c, c'$, when

$$T(cv) \in L \Leftrightarrow T(c'v) \in L \quad \text{for all } v \in {C_{\mathcal{V},\Sigma}}^* \text{ such that } cv \in \text{Codings}_\emptyset \text{ and } c'v \in \text{Codings}_\emptyset \ .$$

Note here that the trees coded by $c$ and $c'$ may have a different set of free variables. This is the only subtlety in the proof of the following expected statement.

▶ **Lemma 14.** *The Myhill-Nerode relation of a tree language $L$ is an equivariant congruence in the sense that it is an equivalence and $cv \equiv_L c'v$ for all codings $c, c'$ such that $c \equiv_L c'$, and every $v \in {C_{\mathcal{V},\Sigma}}^*$.*

It is now standard (see e.g. [10]) to define the *minimal automaton* $\text{Min}_L = (Q, q_0, F, \delta)$ of a tree language $L$ as follows:

- $Q = \{q_0\} \uplus \text{Codings}/_{\equiv_L}$, in which $q_0$ is the (fresh) initial state,
- $F = \{[c]_{\equiv_L} \mid [c]_{\equiv_L} \subseteq L\}$,
- the transition function $\delta$ is given by $\delta(q_0, [x]) = [\,[x]\,]_{\equiv_L}$ and $\delta([c]_{\equiv_L}, v) = [cv]_{\equiv_L}$,

where $[c]_{\equiv_L}$ is the $\equiv_L$-class of $c \in \text{Codings}$.

▶ **Lemma 15.** *For $L$ a a tree language described by a coding automaton, $\text{Min}_L$ is effectively a coding automaton which describes $L$.*

## 3.3 From coding automata to tree algebras

Our next result is Proposition 16 below, which corresponds to the implication from Item 4 to Items 1 and 2 of Theorem 1.

▶ **Proposition 16.** *Every tree language $L$ described by a coding automaton is also recognized by a tree algebra that has both polynomial complexity and bounded orbit-complexity.*

This is proved by transforming the minimal automaton $\mathrm{Min}_L = (Q, q_0, F, \delta)$ that describes a language $L$, into a tree algebra $\mathcal{A}$ that recognizes the same language and has both polynomial complexity and bounded orbit-complexity. We only outline this construction, which is similar to the monoid of transitions of a word automaton.

Let $Q_-$ be $Q \setminus \{q_0\}$. Given a state $q \in Q_-$, a variable $x$ and a tree $t$ possibly with free variables, we define:

$$\delta_t(q, x) := \delta(q, v)$$

in which $v \in C_{\mathcal{V},\Sigma}^*$ is such that $T([x]v) = t$ and does not contain a letter of the form $[\cdot_y a(z_0, ..., z_{\mathrm{ar}(a)-1})]$ in which $y \in \mathrm{supp}(q) \setminus \{x\}$. This definition is shown to be meaningful, in the sense that it does not depend on the choice of $v$.

The elements of $\mathcal{A}$ are then defined to be the $\delta_t$'s, and the operations of the algebra are defined so that the image of a tree $t$ under the evaluation morphism is $\delta_t$. This algebra recognizes $L$ and we show, this is the difficult part of the proof, that it has both polynomial complexity and bounded orbit-complexity.

# 4 Tree algebras

We now tackle the study of tree algebras of polynomial complexity and bounded orbit-complexity, and prove the remaining implications of Theorem 1. We first define in Section 4.1 the notion of system of supports of a tree algebra that appears in Theorem 1. In Section 4.2, we make use of this notion to prove Theorem 1. Finally we state Theorem 22, our decidability result, in Section 4.3.

## 4.1 Syntactic tree algebras and systems of supports

We start this section by defining systems of supports of a tree algebra. We then prove Lemma 17 which states that a syntactic tree algebra always has a minimal system of supports, and that it is stable (this property is mentionned in Item 3 of Theorem 1). Finally, we prove Proposition 19, corresponding to implications from Items 1 and 2 to Item 3 in Theorem 1.

**System of supports.** We now introduce the notion of system of supports. It is a way to transport the notion of support from $\mathbf{Sym}(V)$-sets to tree algebras, which are a collection of $\mathbf{Sym}(X)$-sets for $X$ finite.

For every finite set of variables $X$, a subset $S_a$ of $X$ is a *support* of $a \in A_X$ if $\sigma(a) = a$ for every $\sigma \in \mathbf{Sym}(X \setminus S_a)$. A *system of supports* for an algebra $\mathcal{A}$ is a family $(S_a)_{a \in \mathcal{A}}$ such that $S_a$ is a support of $a$ for all finite $X$ and $a \in A_X$.

**Stability.** The notion of system of supports is however lacking to describe properties of tree algebras, as there is no relation between the supports of the different elements, we introduce different notions of stability to cope with this issue.

We define the following properties for a system of supports $(S_a)_{a \in \mathcal{A}}$:

- Being *stable under renamings* if $S_{\sigma(a)} \subseteq \sigma(S_a)$ for $a \in A_X$, $X$ finite, and renaming $\sigma$.
- Being *stable under internal substitutions* if $S_{a \cdot_x b} \subseteq (S_a \setminus \{x\}) \cup S_b$ for all $a \in A_X$, $x \in S_a$ and $b \in A_Y$, for finite $X$ and $Y$.
- Being *stable under external substitutions* $S_{a \cdot_x b} \subseteq S_a$ for all $a \in A_X$, $x \in X \setminus S_a$ and all $b \in A_Y$, for finite $X$ and $Y$.

- Being *stable under substitutions* if it is both stable under internal and external substitutions.

Finally, a system of supports is *stable* if it is both stable under renamings and substitutions, and it is *bounded* if there is a bound $K$ such that $|S_a| \leq K$ for every $a \in \mathcal{A}$.

Our main result concerning systems of supports, Lemma 17, states the existence of a canonical support for syntactic tree algebras. Moreover, this system of supports $(S_a)_{a\in\mathcal{A}}$ is *minimal*, meaning that $S_a \subseteq T_a$, for every $a \in \mathcal{A}$ and every $(T_a)_{a\in\mathcal{A}}$ with the same stability properties.

▶ **Lemma 17.** *Let $\mathcal{A}$ be a syntactic tree algebra. Then $\mathcal{A}$ has a minimal stable system of supports.*

The proof of this result is lengthy and relies on the fact that a $\Sigma, X$-tree $t$ can be seen as a $\Sigma, Y$-tree for all $X \subseteq Y$, allowing one to use standard techniques from nominal set theory. The system of supports introduced in Lemma 17 is the *canonical system of supports* of $\mathcal{A}$.

▶ **Example 18.** We once again take a look at the tree algebra introduced in Example 5. Its canonical system of supports is given, for all finite $X$, by:

$$S_\top = S_\bot = \emptyset \ , \qquad S_x = \{x\} \text{ for all } x \in X \ .$$

We now establish Proposition 19 below, corresponding to implications from Item 1 to Item 3 and from Item 2 to Item 3 of Theorem 1. According to Lemma 7, it is enough to prove the results for syntactic tree algebras, meaning we only need to prove that the canonical system of supports is bounded.

▶ **Proposition 19.** *Let $\mathcal{A}$ be a finite syntactic tree algebra that has either polynomial complexity or bounded orbit complexity. Then $\mathcal{A}$ has a bounded and stable system of supports.*

## 4.2 From tree algebras to coding automata

Our next result is Proposition 20 below, corresponding to implication from Item 3 to Item 4 of Theorem 1.

▶ **Proposition 20.** *Let $L$ be recognized by a finite tree algebra with a bounded and stable system of supports. Then there is a coding automaton that describes $L$.*

The following Lemma 21 shows that we only need to consider syntactic tree algebras.

▶ **Lemma 21.** *Let $L$ be a language of trees. If $L$ is recognized by a tree algebra that has a bounded and stable system of supports, then the syntactic tree algebra of $L$ can also be equipped with such a system of supports.*

**From tree algebra to coding automata.** Fix a language of trees $L$, let $\mathcal{A}$ be its syntactic tree algebra, and let $(S_a)_{a\in\mathcal{A}}$ be its canonical system of supports, which is bounded (by say $K$) and stable. In order to prove Proposition 20, we extract from $\mathcal{A}$ a coding automaton $\mathrm{Auto}_\mathcal{A}$ that describes $L$. Intuitively, because $(S_a)_{a\in\mathcal{A}}$ is stable and bounded, $\mathcal{A}$ is uniquely determined by the $A_X$'s for $|X| \leq K$. This is used to defined an appropriate automaton, whose set of states is orbit-finite.

This concludes the proof of Proposition 20 and thus the proof of Theorem 1.

## 4.3 Decidability

Our last result is the following.

▶ **Theorem 22.** *There is an algorithm which, given a regular tree language, decides whether it is recognizable by a tree algebra of polynomial complexity.*

The proof informally consists in constructing the expected coding automaton, dropping when possible the variables that are irrelevant for deciding the membership to the language (in the sense that, whatever tree is plugged in it, it does not change the membership to the language). The crucial point is to prove that a bounded number of variables suffices. Regular cost functions over finite trees provide a straightforward technique for solving this boundedness question [6].

# 5 Conclusion

We analyzed the expressive power of unrestrained tree algebras of polynomial complexity. Theorem 1 shows a link between combinatorial properties (being recognized by tree algebras of polynomial complexity, or of bounded orbit-complexity) and an algebraic one (being described by a coding automaton). Doing so, it gives a crisp description of the expressive power of this class of tree algebras, which we also proved to be decidable.

**Other types of tree algebras.** This theory is easily ported to affine tree algebras, for which we get the same equivalence and decidability results. In the case of relevant and linear tree algebras, however, the breaking point is the existence of a canonical system of supports. At the cost of a much more technical proof, it remains possible to prove the equivalence of Items 1, 3, and 4 in Theorem 1. We conjecture Theorem 1 to still hold.

**Future work.** A natural extension to this work is to study the expressive power of tree algebras of exponential complexity, which we conjecture to be the same as the one of tree algebras of polynomial orbit-complexity. Such tree algebras are much more expressive than those studied in this article. It is for instance easy to check that both of these classes subsume the one of languages recognized by top-down deterministic tree automata.

## References


1 Achim Blumensath. Recognisability for algebras of infinite trees. *Theoretical Computer Science*, 412(29):3463–3486, 2011. doi:10.1016/j.tcs.2011.02.037.

2 Achim Blumensath. Regular Tree Algebras. *Logical Methods in Computer Science*, Volume 16, Issue 1, February 2020. doi:10.23638/LMCS-16(1:16)2020.

3 Mikołaj Bojanczyk. Slightly infinite sets, 2016. A draft of a book available at https://www.mimuw.edu.pl/~bojan/paper/atom-book.

4 Mikolaj Bojanczyk and Michal Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'16, New York, NY, USA, July 5-8, 2016*, pages 407–416. ACM, 2016. doi:10.1145/2933575.2934508.

5 Thomas Colcombet and Arthur Jaquard. A complexity approach to tree algebras: the bounded case. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

6 Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 70–79. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.36.

7 Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press, 2012.

8 Zoltán Ésik and Pascal Weil. Algebraic recognizability of regular tree languages. *Theor. Comput. Sci.*, 340(1):291–321, 2005. doi:10.1016/j.tcs.2005.03.038.

9 Zoltán Ésik and Pascal Weil. Algebraic characterization of logically defined tree languages. *Int. J. Algebra Comput.*, 20(2):195–239, 2010. doi:10.1142/S0218196710005595.

10 Sławomir Lasota, Bartek Klin, and Mikołaj Bojańczyk. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10, 2014.

11 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.