A Complexity Approach to Tree Algebras: the Polynomial Case

Arthur Jaquard

joint work with Thomas Colcombet

Université de Paris, CNRS, IRIF

Automata Seminar | March 11, 2021

INSTITUT
DE RECHERCHE
EN INFORMATIQUE
FONDAMENTALE

Université de Paris

Infinitely sorted tree algebras

Let Σ be a ranked alphabet and $\mathcal{V}$ be a countably infinite set of variables. The free tree algebra has as carrier sets the $(T_X)_{X \subseteq \mathcal{V} \text{ finite}}$.

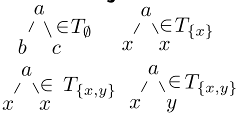
$$T_X = \{\text{trees in which all the variables on the leaves are in } X\}$$

Infinitely sorted tree algebras

Let Σ be a ranked alphabet and $\mathcal{V}$ be a countably infinite set of variables. The free tree algebra has as carrier sets the $(T_X)_{X \subseteq \mathcal{V} \text{ finite}}$.

$$T_X = \{\text{trees in which all the variables on the leaves are in } X\}$$

Objects

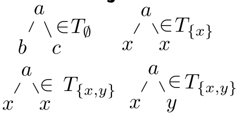


Infinitely sorted tree algebras

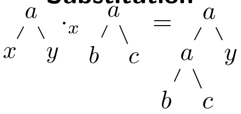
Let Σ be a ranked alphabet and $\mathcal{V}$ be a countably infinite set of variables. The free tree algebra has as carrier sets the $(T_X)_{X \subseteq \mathcal{V} \text{ finite}}$.

$$T_X = \{\text{trees in which all the variables on the leaves are in } X\}$$

Objects



Substitution

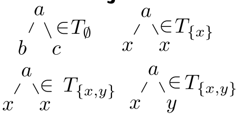


Infinitely sorted tree algebras

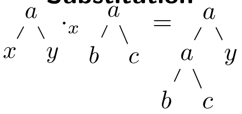
Let Σ be a ranked alphabet and $\mathcal{V}$ be a countably infinite set of variables. The free tree algebra has as carrier sets the $(T_X)_{X \subseteq \mathcal{V} \text{ finite}}$.

$$T_X = \{\text{trees in which all the variables on the leaves are in } X\}$$

Objects

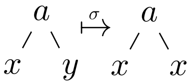


Substitution



Renaming

$$\sigma(x) = \sigma(y) = x$$

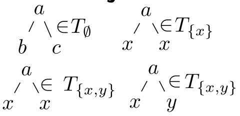


Infinitely sorted tree algebras

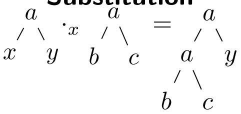
Let Σ be a ranked alphabet and $\mathcal{V}$ be a countably infinite set of variables. The free tree algebra has as carrier sets the $(T_X)_{X \subseteq \mathcal{V} \text{ finite}}$.

$$T_X = \{\text{trees in which all the variables on the leaves are in } X\}$$

Objects

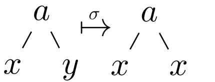


Substitution



Renaming

$$\sigma(x) = \sigma(y) = x$$



Definition (Finite Tree algebras)

A finite tree algebra $\mathcal{A}$ consists of an infinite series of finite carrier sets A_X indexed by finite sets of variables X, together with operations:

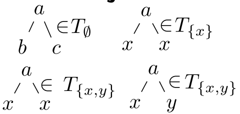
- **Constants.** $a(x_0, \ldots, x_{n-1})^{\mathcal{A}} \in A_{\{x_0,\ldots,x_{n-1}\}}$ for all $a \in \Sigma_n$ and variables x_i,
- **Substitution.** $\cdot_x^{\mathcal{A}} \colon A_X \times A_Y \to A_{X\setminus\{x\}\cup Y}$ for all finite X, Y and variable x,
- **Renaming.** $\sigma^{\mathcal{A}} \colon A_X \to A_Y$ for all maps $\sigma \colon X \to Y$.

Infinitely sorted tree algebras

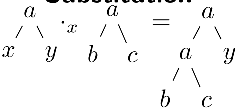
Let Σ be a ranked alphabet and $\mathcal{V}$ be a countably infinite set of variables. The free tree algebra has as carrier sets the $(T_X)_{X \subseteq \mathcal{V} \text{ finite}}$.

$$T_X = \{\text{trees in which all the variables on the leaves are in } X\}$$

Objects

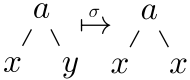


Substitution



Renaming

$$\sigma(x) = \sigma(y) = x$$



Definition (Finite Tree algebras)

A finite tree algebra $\mathcal{A}$ consists of an infinite series of finite carrier sets A_X indexed by finite sets of variables X, together with operations:

- **Constants.** $a(x_0, \ldots, x_{n-1})^{\mathcal{A}} \in A_{\{x_0, \ldots, x_{n-1}\}}$ for all $a \in \Sigma_n$ and variables x_i,
- **Substitution.** $\cdot_x^{\mathcal{A}} : A_X \times A_Y \to A_{X \setminus \{x\} \cup Y}$ for all finite X, Y and variable x,
- **Renaming.** $\sigma^{\mathcal{A}} : A_X \to A_Y$ for all maps $\sigma : X \to Y$.

Identities? $a(x, y) \cdot_y b$ $a(x, z) \cdot_z b$

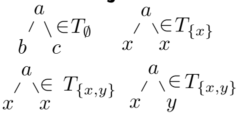
We also define morphisms, congruences...

Infinitely sorted tree algebras

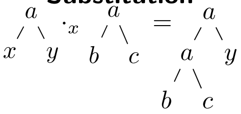
Let Σ be a ranked alphabet and $\mathcal{V}$ be a countably infinite set of variables. The free tree algebra has as carrier sets the $(T_X)_{X \subseteq \mathcal{V} \text{ finite}}$.

$$T_X = \{\text{trees in which all the variables on the leaves are in } X\}$$

Objects

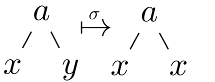


Substitution



Renaming

$$\sigma(x) = \sigma(y) = x$$



Definition (Finite Tree algebras)

A finite tree algebra $\mathcal{A}$ consists of an infinite series of finite carrier sets A_X indexed by finite sets of variables X, together with operations:

- **Constants.** $a(x_0, \ldots, x_{n-1})^{\mathcal{A}} \in A_{\{x_0,\ldots,x_{n-1}\}}$ for all $a \in \Sigma_n$ and variables x_i,
- **Substitution.** $\cdot_x^{\mathcal{A}} \colon A_X \times A_Y \to A_{X \setminus \{x\} \cup Y}$ for all finite X, Y and variable x,
- **Renaming.** $\sigma^{\mathcal{A}} \colon A_X \to A_Y$ for all maps $\sigma \colon X \to Y$.

Given a finite tree algebra $\mathcal{A}$, there is a unique morphism from the free algebra to $\mathcal{A}$. It is called the evaluation morphism of $\mathcal{A}$.

Languages and the size of the algebra

Definition (Language recognized by an algebra)

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

Languages and the size of the algebra

Definition (Language recognized by an algebra)

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

Finite tree algebras exactly recognize the regular languages.

Languages and the size of the algebra

Definition (Language recognized by an algebra)

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

Finite tree algebras exactly recognize the regular languages.

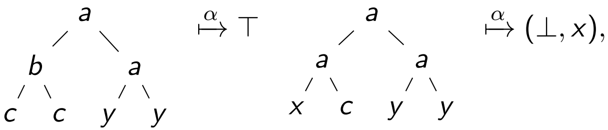
Example $L =$ trees with a b on the leftmost branch

Definition (Language recognized by an algebra)

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

Finite tree algebras exactly recognize the regular languages.

Example $L =$ trees with a b on the leftmost branch



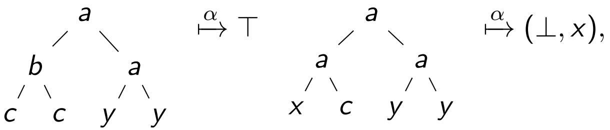
Languages and the size of the algebra

Definition (Language recognized by an algebra)

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

Finite tree algebras exactly recognize the regular languages.

Example $L =$ trees with a b on the leftmost branch



$A_X = \{\top, \bot\} \uplus (\{\top, \bot\} \times X)$ $\qquad |A_X| = 2 + 2|X|$ is linear in $|X|$.

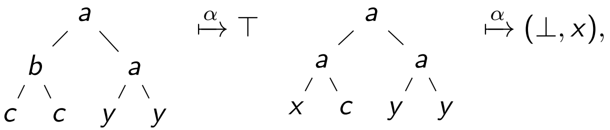
Languages and the size of the algebra

Definition (Language recognized by an algebra)

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

Finite tree algebras exactly recognize the regular languages.

Example $L =$ trees with a b on the leftmost branch



$A_X = \{\top, \bot\} \uplus (\{\top, \bot\} \times X)$ $\qquad$ $|A_X| = 2 + 2|X|$ is linear in $|X|$.

This algebra has linear complexity.

Complexity

Definition (Complexity of an algebra)

The complexity of a finite algebra $\mathcal{A}$ is the asymptotic size of $|A_X|$ as a function of $|X|$.

Complexity

Definition (Complexity of an algebra)

The complexity of a finite algebra $\mathcal{A}$ is the asymptotic size of $|A_X|$ as a function of $|X|$.

A bounded hierarchy of classes

All regular languages are recognized by algebras of doubly-exponential complexity.

Complexity

Definition (Complexity of an algebra)

The complexity of a finite algebra $\mathcal{A}$ is the asymptotic size of $|A_X|$ as a function of $|X|$.

A bounded hierarchy of classes

All regular languages are recognized by algebras of doubly-exponential complexity.

Describe the languages recognized by algebras of bounded / polynomial / exponential complexity.

Complexity

Definition (Complexity of an algebra)

The complexity of a finite algebra $\mathcal{A}$ is the asymptotic size of $|A_X|$ as a function of $|X|$.

A bounded hierarchy of classes

All regular languages are recognized by algebras of doubly-exponential complexity.

Describe the languages recognized by algebras of bounded / polynomial / exponential complexity.

Bounded complexity	[Colcombet, J, 2021]
Polynomial complexity	This talk
Exponential complexity	-
Doubly-exponential complexity	All regular languages

Complexity

Definition (Complexity of an algebra)

The complexity of a finite algebra $\mathcal{A}$ is the asymptotic size of $|A_X|$ as a function of $|X|$.

A bounded hierarchy of classes

All regular languages are recognized by algebras of doubly-exponential complexity.

Describe the languages recognized by algebras of bounded / polynomial / exponential complexity.

Bounded complexity	[Colcombet, J, 2021]
Polynomial complexity	This talk
Exponential complexity	-
Doubly-exponential complexity	All regular languages

The objective is to identify new classes of languages and to gain a better understanding of tree algebras.

Another example

$L =$ trees whose leftmost branch ends with $a(c, c)$, where
$\Sigma = \{(c, 0), (d, 0), (a, 2)\}$

Another example

$L =$ trees whose leftmost branch ends with $a(c, c)$, where
$\Sigma = \{(c, 0), (d, 0), (a, 2)\}$

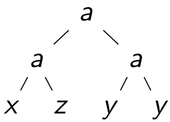
$$A_X = \{c, d\} \cup \{a(x, y) \mid x, y \in X \cup \{c, *\}\}$$

Another example

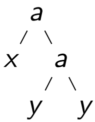
$L =$ trees whose leftmost branch ends with $a(c, c)$, where
$\Sigma = \{(c, 0), (d, 0), (a, 2)\}$

$$A_X = \{c, d\} \cup \{a(x, y) \mid x, y \in X \cup \{c, *\}\}$$

$d \overset{\alpha}{\mapsto} d$



$\overset{\alpha}{\mapsto} a(x, z)$



$\overset{\alpha}{\mapsto} a(x, *)$

Another example

$L =$ trees whose leftmost branch ends with $a(c, c)$, where
$\Sigma = \{(c, 0), (d, 0), (a, 2)\}$

$$A_X = \{c, d\} \cup \{a(x, y) \mid x, y \in X \cup \{c, *\}\}$$

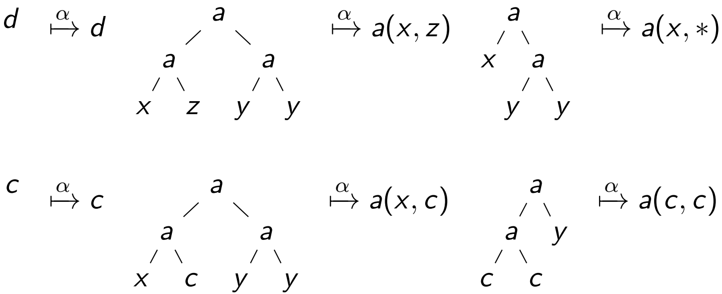
$$d \overset{\alpha}{\mapsto} d \qquad a(a(x, z), a(y, y)) \overset{\alpha}{\mapsto} a(x, z) \qquad a(x, a(y, y)) \overset{\alpha}{\mapsto} a(x, *)$$

$$c \overset{\alpha}{\mapsto} c \qquad a(a(x, c), a(y, y)) \overset{\alpha}{\mapsto} a(x, c) \qquad a(a(c, c), y) \overset{\alpha}{\mapsto} a(c, c)$$

Another example

$L =$ trees whose leftmost branch ends with $a(c,c)$, where
$\Sigma = \{(c,0),(d,0),(a,2)\}$

$$A_X = \{c,d\} \cup \{a(x,y) \mid x,y \in X \cup \{c,*\}\}$$

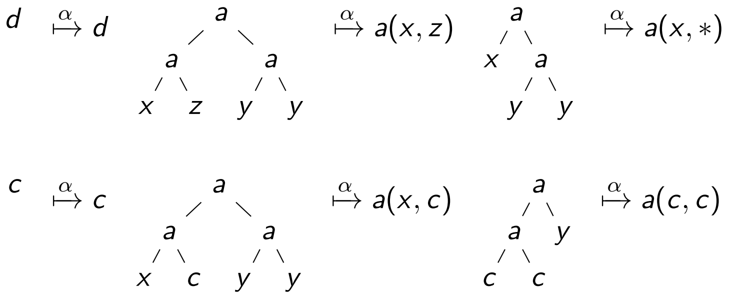


Orbits: c, d, $a(x,y)$, $a(x,x)$, $a(x,c)$, $a(c,x)$, $a(x,*)$, $a(*,x)$, $a(c,c)$, $a(*,*)$

Another example

L = trees whose leftmost branch ends with $a(c,c)$, where $\Sigma = \{(c,0),(d,0),(a,2)\}$

$$A_X = \{c, d\} \cup \{a(x,y) \mid x, y \in X \cup \{c, *\}\}$$



Orbits: c, d, $a(x,y)$, $a(x,x)$, $a(x,c)$, $a(c,x)$, $a(x,*)$, $a(*,x)$, $a(c,c)$, $a(*,*)$

This algebra has quadratic complexity and bounded orbit complexity.

Orbit complexity

Let $|A_X/\mathbf{Sym}(X)|$ be the number of orbits of A_X under the action of $\mathbf{Sym}(X)$ induced by renamings.

Definition (Orbit complexity of an algebra)

The orbit complexity of a finite algebra $\mathcal{A}$ is the asymptotic size of $|A_X/\mathbf{Sym}(X)|$ as a function of $|X|$.

Orbit complexity

Let $|A_X/\mathbf{Sym}(X)|$ be the number of orbits of A_X under the action of $\mathbf{Sym}(X)$ induced by renamings.

Definition (Orbit complexity of an algebra)

The orbit complexity of a finite algebra $\mathcal{A}$ is the asymptotic size of $|A_X/\mathbf{Sym}(X)|$ as a function of $|X|$.

Another bounded hierarchy of classes

All regular languages are recognized by algebras of exponential orbit complexity.

Orbit complexity

Let $|A_X/\mathbf{Sym}(X)|$ be the number of orbits of A_X under the action of $\mathbf{Sym}(X)$ induced by renamings.

Definition (Orbit complexity of an algebra)

The orbit complexity of a finite algebra $\mathcal{A}$ is the asymptotic size of $|A_X/\mathbf{Sym}(X)|$ as a function of $|X|$.

Another bounded hierarchy of classes

All regular languages are recognized by algebras of exponential orbit complexity.

Another hierarchy of classes:

Bounded orbit complexity	-
Polynomial orbit complexity	-
Exponential orbit complexity	All regular languages

What complexity means

Complexity is a tool to quantify what the algebra remembers about the variables:

Bounded complexity
The algebra does not remember anything about the variables.
$A_X \rightsquigarrow$ the variables that appear in the tree are in X.

What complexity means

Complexity is a tool to quantify what the algebra remembers about the variables:

Bounded complexity
The algebra does not remember anything about the variables.
$A_X \rightsquigarrow$ the variables that appear in the tree are in X.

Polynomial complexity
$A_X = X^k \rightsquigarrow k$ variables (e.g. k branches)

What complexity means

Complexity is a tool to quantify what the algebra remembers about the variables:

Bounded complexity
The algebra does not remember anything about the variables.
$A_X \rightsquigarrow$ the variables that appear in the tree are in X.

Polynomial complexity
$A_X = X^k \rightsquigarrow k$ variables (e.g. k branches)

Exponential complexity
$A_X = k^X \rightsquigarrow$ a function from X to k (e.g. a set of variables when $k = 2$, or modulo counting if $k = \mathbb{Z}/q\mathbb{Z}$)

What complexity means

Complexity is a tool to quantify what the algebra remembers about the variables:

Bounded complexity
The algebra does not remember anything about the variables.
$A_X \rightsquigarrow$ the variables that appear in the tree are in X.

Polynomial complexity
$A_X = X^k \rightsquigarrow k$ variables (e.g. k branches)

Exponential complexity
$A_X = k^X \rightsquigarrow$ a function from X to k (e.g. a set of variables when $k = 2$, or modulo counting if $k = \mathbb{Z}/q\mathbb{Z}$)

Doubly exponential complexity
All regular languages.

Polynomial complexity

What are the languages recognized by algebras of polynomial complexity?

Polynomial complexity

What are the languages recognized by algebras of polynomial complexity?

- $L =$ trees with a b on the leftmost branch,
- $L =$ trees with some fixed branch in a fixed regular language,
- Boolean combinations of such languages.

Polynomial complexity

What are the languages recognized by algebras of polynomial complexity?

- $L =$ trees with a b on the leftmost branch,
- $L =$ trees with some fixed branch in a fixed regular language,
- Boolean combinations of such languages.
- $L =$ trees whose leftmost branch ends with $a(c, c)$.

Polynomial complexity

What are the languages recognized by algebras of polynomial complexity?

- $L =$ trees with a b on the leftmost branch,
- $L =$ trees with some fixed branch in a fixed regular language,
- Boolean combinations of such languages.
- $L =$ trees whose leftmost branch ends with $a(c, c)$.

Common property: at all times, these algebras only keep in memory a bounded number of branches.

Polynomial complexity

What are the languages recognized by algebras of polynomial complexity?

- $L =$ trees with a b on the leftmost branch,
- $L =$ trees with some fixed branch in a fixed regular language,
- Boolean combinations of such languages.
- $L =$ trees whose leftmost branch ends with $a(c, c)$.

Common property: at all times, these algebras only keep in memory a bounded number of branches.

Equivalence theorem

For a regular language of finite trees, the following properties are equivalent:

a. Being recognized by a finite tree algebra of polynomial complexity.

Polynomial complexity

What are the languages recognized by algebras of polynomial complexity?

- $L =$ trees with a b on the leftmost branch,
- $L =$ trees with some fixed branch in a fixed regular language,
- Boolean combinations of such languages.
- $L =$ trees whose leftmost branch ends with $a(c, c)$.

Common property: at all times, these algebras only keep in memory a bounded number of branches.

Equivalence theorem

For a regular language of finite trees, the following properties are equivalent:

a. Being recognized by a finite tree algebra of polynomial complexity.

b. Being recognized by a finite tree algebra of bounded orbit complexity.

Polynomial complexity

What are the languages recognized by algebras of polynomial complexity?

- $L =$ trees with a b on the leftmost branch,
- $L =$ trees with some fixed branch in a fixed regular language,
- Boolean combinations of such languages.
- $L =$ trees whose leftmost branch ends with $a(c, c)$.

Common property: at all times, these algebras only keep in memory a bounded number of branches.

Equivalence theorem

For a regular language of finite trees, the following properties are equivalent:

a. Being recognized by a finite tree algebra of polynomial complexity.

b. Being recognized by a finite tree algebra of bounded orbit complexity.

Equivalence between a. and b. is not obvious.

Polynomial complexity

What are the languages recognized by algebras of polynomial complexity?

- $L =$ trees with a b on the leftmost branch,
- $L =$ trees with some fixed branch in a fixed regular language,
- Boolean combinations of such languages.
- $L =$ trees whose leftmost branch ends with $a(c, c)$.

Common property: at all times, these algebras only keep in memory a bounded number of branches.

Equivalence theorem

For a regular language of finite trees, the following properties are equivalent:

a. Being recognized by a finite tree algebra of polynomial complexity.
b. Being recognized by a finite tree algebra of bounded orbit complexity.
c. Being described by a coding automaton.

Equivalence between a. and b. is not obvious.

Nominal automata

Let $\mathbf{Sym}(\mathcal{V})$ act upon sets X and Y.

- X is called orbit-finite if the group action has finitely many orbits.

Nominal automata

Let $\mathbf{Sym}(\mathcal{V})$ act upon sets X and Y.

- X is called orbit-finite if the group action has finitely many orbits.
- $x \in X$ is called finitely supported if there exists $S \subseteq \mathcal{V}$ finite such that, for every $\sigma \in \mathbf{Sym}(\mathcal{V})$, $\sigma(x) = x$ whenever $\sigma(s) = s$ for every $s \in S$.

Nominal automata

Let $\mathbf{Sym}(\mathcal{V})$ act upon sets X and Y.

- X is called orbit-finite if the group action has finitely many orbits.
- $x \in X$ is called finitely supported if there exists $S \subseteq \mathcal{V}$ finite such that, for every $\sigma \in \mathbf{Sym}(\mathcal{V})$, $\sigma(x) = x$ whenever $\sigma(s) = s$ for every $s \in S$.
- X is called nominal if its elements are finitely supported.

Nominal automata

Let $\mathbf{Sym}(\mathcal{V})$ act upon sets X and Y.

- X is called orbit-finite if the group action has finitely many orbits.
- $x \in X$ is called finitely supported if there exists $S \subseteq \mathcal{V}$ finite such that, for every $\sigma \in \mathbf{Sym}(\mathcal{V})$, $\sigma(x) = x$ whenever $\sigma(s) = s$ for every $s \in S$.
- X is called nominal if its elements are finitely supported.
- $f \colon X \to Y$ is supported by $S \subseteq \mathcal{V}$ if $f(\sigma(x)) = \sigma(f(x))$, for all $x \in X$, $\sigma \in \mathbf{Sym}(\mathcal{V} \setminus S)$.

Nominal automata

Let $\mathbf{Sym}(\mathcal{V})$ act upon sets X and Y.

- X is called orbit-finite if the group action has finitely many orbits.
- $x \in X$ is called finitely supported if there exists $S \subseteq \mathcal{V}$ finite such that, for every $\sigma \in \mathbf{Sym}(\mathcal{V})$, $\sigma(x) = x$ whenever $\sigma(s) = s$ for every $s \in S$.
- X is called nominal if its elements are finitely supported.
- $f : X \to Y$ is supported by $S \subseteq \mathcal{V}$ if $f(\sigma(x)) = \sigma(f(x))$, for all $x \in X$, $\sigma \in \mathbf{Sym}(\mathcal{V} \setminus S)$.
- X (resp. f) is called equivariant if it is supported by the empty set.

Nominal automata

Let $\mathbf{Sym}(\mathcal{V})$ act upon sets X and Y.

- X is called orbit-finite if the group action has finitely many orbits.
- $x \in X$ is called finitely supported if there exists $S \subseteq \mathcal{V}$ finite such that, for every $\sigma \in \mathbf{Sym}(\mathcal{V})$, $\sigma(x) = x$ whenever $\sigma(s) = s$ for every $s \in S$.
- X is called nominal if its elements are finitely supported.
- $f : X \to Y$ is supported by $S \subseteq \mathcal{V}$ if $f(\sigma(x)) = \sigma(f(x))$, for all $x \in X$, $\sigma \in \mathbf{Sym}(\mathcal{V} \setminus S)$.
- X (resp. f) is called equivariant if it is supported by the empty set.

A deterministic orbit-finite nominal automaton is given by

- an orbit-finite nominal set A (the alphabet),
- an orbit-finite nominal set Q (the states),
- equivariant subsets $\{q_I\}$ and F of Q (the initial state and the final states),
- and an equivariant transition function $\delta : Q \times A \to Q$.

Nominal automata

Let $\mathbf{Sym}(\mathcal{V})$ act upon sets X and Y.

- X is called orbit-finite if the group action has finitely many orbits.
- $x \in X$ is called finitely supported if there exists $S \subseteq \mathcal{V}$ finite such that, for every $\sigma \in \mathbf{Sym}(\mathcal{V})$, $\sigma(x) = x$ whenever $\sigma(s) = s$ for every $s \in S$.
- X is called nominal if its elements are finitely supported.
- $f \colon X \to Y$ is supported by $S \subseteq \mathcal{V}$ if $f(\sigma(x)) = \sigma(f(x))$, for all $x \in X$, $\sigma \in \mathbf{Sym}(\mathcal{V} \setminus S)$.
- X (resp. f) is called equivariant if it is supported by the empty set.

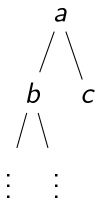
A deterministic orbit-finite nominal automaton is given by

- an orbit-finite nominal set A (the alphabet),
- an orbit-finite nominal set Q (the states),
- equivariant subsets $\{q_I\}$ and F of Q (the initial state and the final states),
- and an equivariant transition function $\delta \colon Q \times A \to Q$.

Example: a deterministic register automaton can be seen as a deterministic orbit-finite nominal automaton.

Coding of trees

How to build the following tree ?



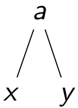
Coding of trees

How to build the following tree ?

x $[x]$

Coding of trees

How to build the following tree ?

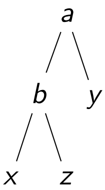


$[x]$

$[\cdot_x a(x, y)]$

Coding of trees

How to build the following tree ?



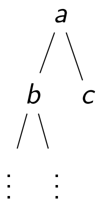
$[x]$

$[\cdot_x a(x, y)]$

$[\cdot_x b(x, z)]$

Coding of trees

How to build the following tree ?



$[x]$

$[\cdot_x a(x,y)]$

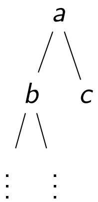
$[\cdot_x b(x,z)]$

$[\cdot_y c]$

$\dots$

Coding of trees

How to build the following tree ?



$[x]$

$[\cdot_x a(x, y)]$

$[\cdot_x b(x, z)]$

$[\cdot_y c]$

...

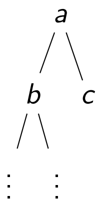
$$C_{\mathcal{V}} = \{[x] \mid x \in \mathcal{V}\}$$

$$C_{\mathcal{V},\Sigma} = \{[\cdot_x a(x_0, \ldots, x_{n-1})] \mid a \in \Sigma_n, x, x_0, \ldots, x_{n-1} \in \mathcal{V}\}$$

The alphabet $C_{\mathcal{V}} \cup C_{\mathcal{V},\Sigma}$ is called the coding alphabet. It is a nominal orbit-finite alphabet.

Coding of trees

How to build the following tree ?



$[x]$

$[\cdot_x a(x, y)]$

$[\cdot_x b(x, z)]$

$[\cdot_y c]$

...

$$C_{\mathcal{V}} = \{[x] \mid x \in \mathcal{V}\}$$

$$C_{\mathcal{V},\Sigma} = \{[\cdot_x a(x_0, ..., x_{n-1})] \mid a \in \Sigma_n, x, x_0, ..., x_{n-1} \in \mathcal{V}\}$$

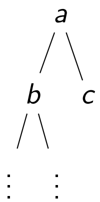
The alphabet $C_{\mathcal{V}} \cup C_{\mathcal{V},\Sigma}$ is called the coding alphabet. It is a nominal orbit-finite alphabet.

Tree coding and the coding alphabet

A word $c \in C_{\mathcal{V}} C^*_{\mathcal{V},\Sigma}$ is called a tree coding. A coding c evaluates to a finite tree $T(c)$.

Coding of trees

How to build the following tree ?



$[x]$

$[\cdot_x a(x, y)]$

$[\cdot_x b(x, z)]$

$[\cdot_y c]$

...

$C_{\mathcal{V}} = \{[x] \mid x \in \mathcal{V}\}$

$C_{\mathcal{V},\Sigma} = \{[\cdot_x a(x_0, \ldots, x_{n-1})] \mid a \in \Sigma_n, x, x_0, \ldots, x_{n-1} \in \mathcal{V}\}$

The alphabet $C_{\mathcal{V}} \cup C_{\mathcal{V},\Sigma}$ is called the coding alphabet. It is a nominal orbit-finite alphabet.

Tree coding and the coding alphabet

A word $c \in C_{\mathcal{V}} C^*_{\mathcal{V},\Sigma}$ is called a tree coding. A coding c evaluates to a finite tree $T(c)$.

Coding languages describing tree languages

A language L of codings describes a language $K \subseteq T_\emptyset$ of trees if, for every coding c such that $T(c) \in T_\emptyset$, $c \in L$ if and only if $T(c) \in K$.

Dealing with missing variables

Let $c = [x][\cdot_x a(x, y)][\cdot_z c]$. What is $T(c)$?

Dealing with missing variables

Let $c = [x][\cdot_x a(x, y)][\cdot_z c]$. What is $T(c)$?

Dealing with missing variables

Let $c = [x][\cdot_x a(x, y)][\cdot_z c]$. What is $T(c)$?

$$\mathrm{create}_z \colon X \to X \cup \{z\}$$

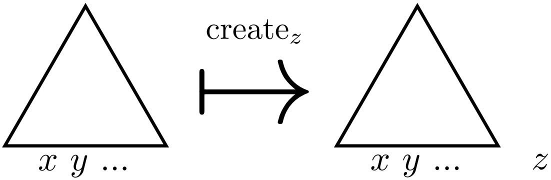
such that $\mathrm{create}_z(x) = x$ for all $x \in X$.

Dealing with missing variables

Let $c = [x][\cdot_x a(x,y)][\cdot_z c]$. What is $T(c)$?

$$\mathrm{create}_z \colon X \to X \cup \{z\}$$

such that $\mathrm{create}_z(x) = x$ for all $x \in X$.

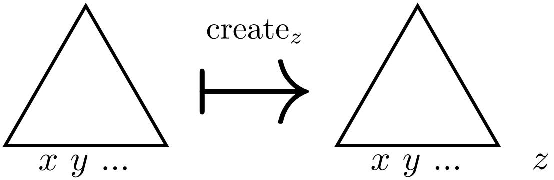


Dealing with missing variables

Let $c = [x][\cdot_x a(x,y)][\cdot_z c]$. What is $T(c)$?

$$\mathrm{create}_z \colon X \to X \cup \{z\}$$

such that $\mathrm{create}_z(x) = x$ for all $x \in X$.



$$T(c) = \mathrm{create}_z(a(x,y)) \cdot_z c = a(x,y)$$

Coding automata

Coding languages describing tree languages

A language L of codings describes a language $K \subseteq T_\emptyset$ of trees if, for every coding c such that $T(c) \in T_\emptyset$, $c \in L$ if and only if $T(c) \in K$.

Coding automata

Coding languages describing tree languages

A language L of codings describes a language $K \subseteq T_\emptyset$ of trees if, for every coding c such that $T(c) \in T_\emptyset$, $c \in L$ if and only if $T(c) \in K$.

Example $L =$ "codings c such that $T(c) \in K$"

Coding automata

Coding languages describing tree languages

A language L of codings describes a language $K \subseteq T_\emptyset$ of trees if, for every coding c such that $T(c) \in T_\emptyset$, $c \in L$ if and only if $T(c) \in K$.

Example $L =$ "codings c such that $T(c) \in K$"
Example $L =$ "the third letter is of the form $[\cdot_y c]$", $\Sigma = \{(a, 2), (c, 0)\}$.

Coding automata

Coding languages describing tree languages

A language L of codings describes a language $K \subseteq T_\emptyset$ of trees if, for every coding c such that $T(c) \in T_\emptyset$, $c \in L$ if and only if $T(c) \in K$.

Example $L =$ "codings c such that $T(c) \in K$"
Example $L =$ "the third letter is of the form $[\cdot_y c]$", $\Sigma = \{(a, 2), (c, 0)\}$.

$$c = [x][\cdot_x a(x, y)][\cdot_y c][\cdot_x a(y, y)][\cdot_y c] \qquad c' = [x][\cdot_x a(x, y)][\cdot_x a(y, y)][\cdot_y c]$$

$$T(c) = T(c') = a(a(c, c), c)$$

Coding automata

Coding languages describing tree languages

A language L of codings describes a language $K \subseteq T_\emptyset$ of trees if, for every coding c such that $T(c) \in T_\emptyset$, $c \in L$ if and only if $T(c) \in K$.

Example $L =$ "codings c such that $T(c) \in K$"
Example $L =$ "the third letter is of the form $[\cdot_y c]$", $\Sigma = \{(a, 2), (c, 0)\}$.

$$c = [x][\cdot_x a(x, y)][\cdot_y c][\cdot_x a(y, y)][\cdot_y c] \qquad c' = [x][\cdot_x a(x, y)][\cdot_x a(y, y)][\cdot_y c]$$

$$T(c) = T(c') = a(a(c, c), c)$$

Coding automaton

A deterministic orbit-finite nominal automaton over the coding alphabet is a coding automaton if it recognizes a language L of codings that describes a tree language K. We say that it describes K.

Coding automata

Coding languages describing tree languages

A language L of codings describes a language $K \subseteq T_\emptyset$ of trees if, for every coding c such that $T(c) \in T_\emptyset$, $c \in L$ if and only if $T(c) \in K$.

Example $L =$ "codings c such that $T(c) \in K$"
Example $L =$ "the third letter is of the form $[\cdot_y c]$", $\Sigma = \{(a,2),(c,0)\}$.

$$c = [x][\cdot_x a(x,y)][\cdot_y c][\cdot_x a(y,y)][\cdot_y c] \qquad c' = [x][\cdot_x a(x,y)][\cdot_x a(y,y)][\cdot_y c]$$

$$T(c) = T(c') = a(a(c,c),c)$$

Coding automaton

A deterministic orbit-finite nominal automaton over the coding alphabet is a coding automaton if it recognizes a language L of codings that describes a tree language K. We say that it describes K.

We assume that there is no transition toward the initial state q_0.

Language described by a coding automaton

$K =$ "trees with a c at depth 1", where $\Sigma = \{(a,2),(c,0)\}$.

Language described by a coding automaton

$K =$ "trees with a c at depth 1", where $\Sigma = \{(a, 2), (c, 0)\}$.

x c $a(x, y)$ $a(x, x)$ $a(x, *)$ $a(*, x)$ $a(*, *)$ $a(x, c)$ $a(c, x)$ $a(c, c)$

Language described by a coding automaton

$K =$ "trees with a c at depth 1", where $\Sigma = \{(a,2),(c,0)\}$.

x c $a(x,y)$ $a(x,x)$ $a(x,*)$ $a(*,x)$ $a(*,*)$ $a(x,c)$ $a(c,x)$ $a(c,c)$

Remark we should also consider $a(c,*)$ and $a(*,c)$.

Language described by a coding automaton

$K =$ "trees with a c at depth 1", where $\Sigma = \{(a,2),(c,0)\}$.

	x	c	$a(x,y)$	$a(x,x)$	$a(x,*)$	$a(*,x)$	$a(*,*)$	$a(x,c)$	$a(c,x)$	$a(c,c)$
q_0	x	c	$a(x,y)$	$a(x,x)$	$a(x,*)$	$a(*,x)$	$a(*,*)$	$a(x,c)$	$a(c,x)$	$a(c,c)$

Remark we should also consider $a(c,*)$ and $a(*,c)$.

Language described by a coding automaton

$K =$ "trees with a c at depth 1", where $\Sigma = \{(a, 2), (c, 0)\}$.

	x	c	$a(x, y)$	$a(x, x)$	$a(x, *)$	$a(*, x)$	$a(*, *)$	$a(x, c)$	$a(c, x)$	$a(c, c)$
q_0	x	c	$a(x, y)$	$a(x, x)$	$a(x, *)$	$a(*, x)$	$a(*, *)$	$a(x, c)$	$a(c, x)$	$a(c, c)$
q_0	x	$\bot$	$a\{x, y\}$	$a\{x\}$	$a\{x\}$	$a\{x\}$	$\bot$	$\top$	$\top$	$\top$

Remark we should also consider $a(c, *)$ and $a(*, c)$.

Language described by a coding automaton

$K =$ "trees with a c at depth 1", where $\Sigma = \{(a, 2), (c, 0)\}$.

	x	c	$a(x,y)$	$a(x,x)$	$a(x,*)$	$a(*,x)$	$a(*,*)$	$a(x,c)$	$a(c,x)$	$a(c,c)$
q_0	x	c	$a(x,y)$	$a(x,x)$	$a(x,*)$	$a(*,x)$	$a(*,*)$	$a(x,c)$	$a(c,x)$	$a(c,c)$
q_0	x	$\bot$	$a\{x,y\}$	$a\{x\}$	$a\{x\}$	$a\{x\}$	$\bot$	$\top$	$\top$	$\top$

Remark we should also consider $a(c,*)$ and $a(*,c)$.

Language described by a coding automaton

$K =$ "trees with a c at depth 1", where $\Sigma = \{(a,2),(c,0)\}$.

	x	c	$a(x,y)$	$a(x,x)$	$a(x,*)$	$a(*,x)$	$a(*,*)$	$a(x,c)$	$a(c,x)$	$a(c,c)$
q_0	x	c	$a(x,y)$	$a(x,x)$	$a(x,*)$	$a(*,x)$	$a(*,*)$	$a(x,c)$	$a(c,x)$	$a(c,c)$
q_0	x	$\bot$	$a\{x,y\}$	$a\{x\}$	$a\{x\}$	$a\{x\}$	$\bot$	$\top$	$\top$	$\top$

Remark we should also consider $a(c,*)$ and $a(*,c)$.

Language described by a coding automaton

$K =$ "trees with a c at depth 1", where $\Sigma = \{(a,2),(c,0)\}$.

	x	c	$a(x,y)$	$a(x,x)$	$a(x,*)$	$a(*,x)$	$a(*,*)$	$a(x,c)$	$a(c,x)$	$a(c,c)$
q_0	x	c	$a(x,y)$	$a(x,x)$	$a(x,*)$	$a(*,x)$	$a(*,*)$	$a(x,c)$	$a(c,x)$	$a(c,c)$
q_0	x	$\bot$	$a\{x,y\}$	$a\{x\}$	$a\{x\}$	$a\{x\}$	$\bot$	$\top$	$\top$	$\top$

Remark we should also consider $a(c,*)$ and $a(*,c)$.

Language described by a coding automaton

$K =$ "trees with a c at depth 1", where $\Sigma = \{(a, 2), (c, 0)\}$.

	x	c	$a(x,y)$	$a(x,x)$	$a(x,*)$	$a(*,x)$	$a(*,*)$	$a(x,c)$	$a(c,x)$	$a(c,c)$
q_0	x	c	$a(x,y)$	$a(x,x)$	$a(x,*)$	$a(*,x)$	$a(*,*)$	$a(x,c)$	$a(c,x)$	$a(c,c)$
q_0	x	$\bot$	$a\{x,y\}$	$a\{x\}$	$a\{x\}$	$a\{x\}$	$\bot$	$\top$	$\top$	$\top$

Remark we should also consider $a(c,*)$ and $a(*,c)$.

A state is an abstraction of a tree, that possibly *forgot* some variables.

Minimizing coding automata

Myhill-Nerode relation of a tree language L. Let $c, c' \in C_{\mathcal{V}} C^*_{\mathcal{V},\Sigma}$ be tree codings. $c \equiv_L c'$ if

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

Minimizing coding automata

Myhill-Nerode relation of a tree language L. Let $c, c' \in C_{\mathcal{V}} C^*_{\mathcal{V},\Sigma}$ be tree codings. $c \equiv_L c'$ if

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

Minimizing coding automata

Myhill-Nerode relation of a tree language L. Let $c, c' \in C_{\mathcal{V}} C^*_{\mathcal{V},\Sigma}$ be tree codings. $c \equiv_L c'$ if

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

The minimal automaton Min_L of L is defined as follows:

- the set of states is $Q = \{q_0\} \uplus C_{\mathcal{V}} C^*_{\mathcal{V},\Sigma} / \equiv_L$,
- $[c]_{\equiv_L}$ is accepting if $[c]_{\equiv_L} \subseteq L$,
- $\delta(q_0, [x]) = [\ [x]\ ]_{\equiv_L}, \qquad \delta([c]_{\equiv_L}, v) = [cv]_{\equiv_L}$.

Minimizing coding automata

Myhill-Nerode relation of a tree language L. Let $c, c' \in C_{\mathcal{V}} C^*_{\mathcal{V},\Sigma}$ be tree codings. $c \equiv_L c'$ if

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

The minimal automaton Min_L of L is defined as follows:

- the set of states is $Q = \{q_0\} \uplus C_{\mathcal{V}} C^*_{\mathcal{V},\Sigma} / \equiv_L$,
- $[c]_{\equiv_L}$ is accepting if $[c]_{\equiv_L} \subseteq L$,
- $\delta(q_0, [x]) = [\ [x]\ ]_{\equiv_L}, \qquad \delta([c]_{\equiv_L}, v) = [cv]_{\equiv_L}$.

Minimal automaton

For L a tree language described by a coding automaton, Min_L is a coding automaton which describes L.

From coding automata to tree algebras 1/2

From coding automata to tree algebras

Every tree language L described by a coding automaton is recognized by a tree algebra that has polynomial complexity and bounded orbit complexity.

From coding automata to tree algebras 1/2

From coding automata to tree algebras

Every tree language L described by a coding automaton is recognized by a tree algebra that has polynomial complexity and bounded orbit complexity.

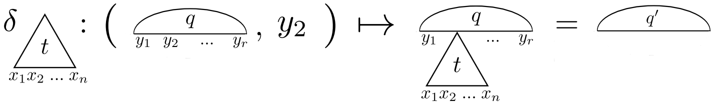
Idea: start from Min_L and define a tree algebra $\mathcal{A}$ that recognizes L.

From coding automata to tree algebras 1/2

From coding automata to tree algebras

Every tree language L described by a coding automaton is recognized by a tree algebra that has polynomial complexity and bounded orbit complexity.

Idea: start from Min_L and define a tree algebra $\mathcal{A}$ that recognizes L.
Fix a tree t with variables $x_1, \ldots, x_n$, we define a function δ_t as



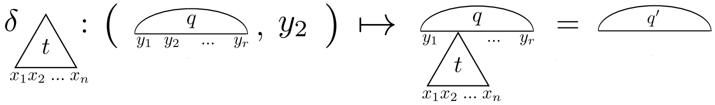
where $q \in Q \setminus \{q_0\}$ is a state supported by $\{y_1, \ldots, y_m\}$.

From coding automata to tree algebras 1/2

From coding automata to tree algebras

Every tree language L described by a coding automaton is recognized by a tree algebra that has polynomial complexity and bounded orbit complexity.

Idea: start from Min_L and define a tree algebra $\mathcal{A}$ that recognizes L.
Fix a tree t with variables $x_1, \ldots, x_n$, we define a function δ_t as



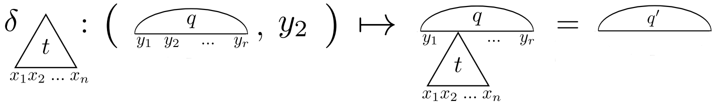
where $q \in Q \setminus \{q_0\}$ is a state supported by $\{y_1, \ldots, y_m\}$.
Example For $t = a(x_1, c)$, this is defined by $q' = \delta(q, [\cdot_{y_2} a(x_1, z)][\cdot_z c])$.

From coding automata to tree algebras 1/2

From coding automata to tree algebras

Every tree language L described by a coding automaton is recognized by a tree algebra that has polynomial complexity and bounded orbit complexity.

Idea: start from Min_L and define a tree algebra $\mathcal{A}$ that recognizes L.
Fix a tree t with variables $x_1, \ldots, x_n$, we define a function δ_t as



where $q \in Q \setminus \{q_0\}$ is a state supported by $\{y_1, \ldots, y_m\}$.
Example For $t = a(x_1, c)$, this is defined by $q' = \delta(q, [\cdot_{y_2} a(x_1, z)][\cdot_z c])$.

δ_t is well defined

The definition of δ_t does not depend on a particular choice of coding.
Let $\mathrm{Trans}(\mathrm{Min}_L)$ be the set of all functions δ_t.

From coding automata to tree algebras 2/2

We define the tree as algebra $\mathcal{A}$ as

$$A_X = \{\delta_t \in \mathrm{Trans}(\mathrm{Min}_L) \mid \delta_t \text{ is supported by } X\} \ .$$

From coding automata to tree algebras 2/2

We define the tree as algebra $\mathcal{A}$ as

$$A_X = \{\delta_t \in \mathrm{Trans}(\mathrm{Min}_L) \mid \delta_t \text{ is supported by } X\} \ .$$

The operations are defined so that $\alpha\colon t \mapsto \delta_t$ is the evaluation morphism.

From coding automata to tree algebras 2/2

We define the tree as algebra $\mathcal{A}$ as

$$A_X = \{\delta_t \in \mathrm{Trans}(\mathrm{Min}_L) \mid \delta_t \text{ is supported by } X\} \ .$$

The operations are defined so that $\alpha \colon t \mapsto \delta_t$ is the evaluation morphism.

Support of δ_t

The size of the supports of the δ_t's is bounded by an integer K.

Let A and B be orbit-finite nominal sets. The set of all functions from A to B with support of size at most K is orbit-finite.

From coding automata to tree algebras 2/2

We define the tree as algebra $\mathcal{A}$ as

$$A_X = \{\delta_t \in \mathrm{Trans}(\mathrm{Min}_L) \mid \delta_t \text{ is supported by } X\} \ .$$

The operations are defined so that $\alpha\colon t \mapsto \delta_t$ is the evaluation morphism.

Support of δ_t

The size of the supports of the δ_t's is bounded by an integer K.

Let A and B be orbit-finite nominal sets. The set of all functions from A to B with support of size at most K is orbit-finite.

$\mathcal{A}$ **has bounded orbit complexity.** $\mathrm{Trans}(\mathrm{Min}_L)$ has finitely many orbits. $f, g \in A_X$ are on the same $\mathbf{Sym}(X)$-orbit if and only if they are on the same $\mathbf{Sym}(\mathcal{V})$-orbit.

From coding automata to tree algebras 2/2

We define the tree as algebra $\mathcal{A}$ as

$$A_X = \{\delta_t \in \mathrm{Trans}(\mathrm{Min}_L) \mid \delta_t \text{ is supported by } X\} \ .$$

The operations are defined so that $\alpha \colon t \mapsto \delta_t$ is the evaluation morphism.

Support of δ_t

The size of the supports of the δ_t's is bounded by an integer K.

Let A and B be orbit-finite nominal sets. The set of all functions from A to B with support of size at most K is orbit-finite.

$\mathcal{A}$ **has bounded orbit complexity.** $\mathrm{Trans}(\mathrm{Min}_L)$ has finitely many orbits. $f, g \in A_X$ are on the same $\mathbf{Sym}(X)$-orbit if and only if they are on the same $\mathbf{Sym}(\mathcal{V})$-orbit.

$\mathcal{A}$ **has polynomial complexity.** A_X has boundedly many orbits. On any orbit, there are at most $\frac{|X|!}{(|X|-k)!}$ elements under the action of $\mathbf{Sym}(X)$.

From tree algebras to coding automata

From tree algebra to coding automata

Every language of trees L recognized by a tree algebra of polynomial complexity or of bounded orbit complexity is described by a coding automaton.

From tree algebras to coding automata

From tree algebra to coding automata

Every language of trees L recognized by a tree algebra of polynomial complexity or of bounded orbit complexity is described by a coding automaton.

Structure of the proof.

1. Extend the notion of support to tree algebras, which are a collection of $\mathbf{Sym}(X)$-sets for $X \subseteq \mathcal{V}$ finite.

From tree algebras to coding automata

From tree algebra to coding automata

Every language of trees L recognized by a tree algebra of polynomial complexity or of bounded orbit complexity is described by a coding automaton.

Structure of the proof.

1. Extend the notion of support to tree algebras, which are a collection of $\mathbf{Sym}(X)$-sets for $X \subseteq \mathcal{V}$ finite.
2. Prove that tree algebras of polynomial complexity or bounded orbit complexity have supports of bounded size (say K).

From tree algebras to coding automata

From tree algebra to coding automata

Every language of trees L recognized by a tree algebra of polynomial complexity or of bounded orbit complexity is described by a coding automaton.

Structure of the proof.

1. Extend the notion of support to tree algebras, which are a collection of $\mathbf{Sym}(X)$-sets for $X \subseteq \mathcal{V}$ finite.
2. Prove that tree algebras of polynomial complexity or bounded orbit complexity have supports of bounded size (say K).
3. Thus, only the elements in sorts A_X where $|X| \leq K$ matter. Let

$$Q = \bigcup_{|X| \leq K} A_X \ .$$

This is used to define a coding automaton that describes L.

Decidability

Decidability

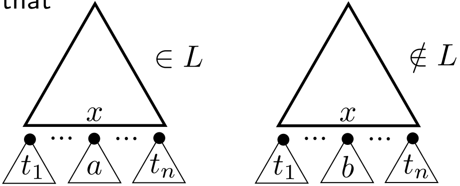
There is an algorithm which, given a regular tree language, decides whether it is recognizable by a tree algebra of polynomial complexity.

Decidability

Decidability

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

Fix L. A tree $t \in T_{\{\bullet\}}$ is L-sensitive to a leaf x if there exist trees $a, b, t_1, \ldots, t_n$ such that

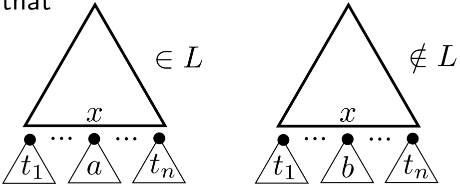


Decidability

Decidability

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

Fix L. A tree $t \in T_{\{\bullet\}}$ is L-sensitive to a leaf x if there exist trees $a, b, t_1, \ldots, t_n$ such that



Lemma

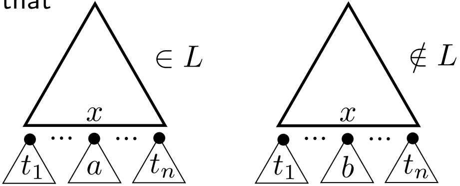
A regular language of trees L is described by a coding automaton if and only if there is a bound on the number of L-sensitive leaves in trees.

Decidability

Decidability

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

Fix L. A tree $t \in T_{\{\bullet\}}$ is L-sensitive to a leaf x if there exist trees $a, b, t_1, \ldots, t_n$ such that



Lemma

A regular language of trees L is described by a coding automaton if and only if there is a bound on the number of L-sensitive leaves in trees.

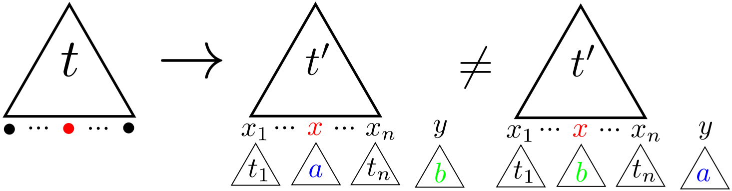
The existence of such a bound can be encoded into cost-MSO. Thus, it is decidable.

From a coding automaton to a bound

From a coding automaton to a bound

If L is described by a coding automaton, then there is a bound on the number of L-sensitive leaves in trees.

Suppose $t \in T_{\{\bullet\}}$ is L-sensitive to some leaf (and let $a, b, t_1, \ldots, t_n$ be the associated trees).

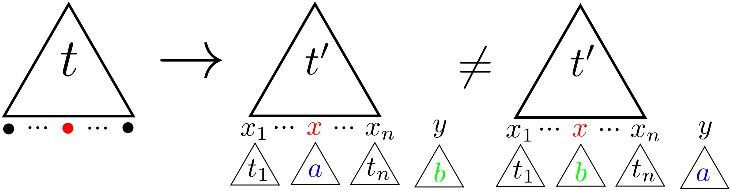


From a coding automaton to a bound

From a coding automaton to a bound

If L is described by a coding automaton, then there is a bound on the number of L-sensitive leaves in trees.

Suppose $t \in T_{\{\bullet\}}$ is L-sensitive to some leaf (and let $a, b, t_1, ..., t_n$ be the associated trees).



Let c be a coding such that $T(c) = t'$.

$$t'[\cdot_x a][\cdot_y b] \neq ((x\ y)t')[\cdot_x a][\cdot_y b]$$

Then x is in the support of $\delta(q_0, c)$ in Min_L.

From a bound to a coding automaton 1/2

From a bound to a coding automaton

If there is a bound on the number of L-sensitive leaves in trees, then L is described by a coding automaton.

Let $\mathcal{A} = (Q, q_0, F, \delta)$ be the minimal tree automaton for L, and suppose K is a bound on the number of L-sensitive leaves.

From a bound to a coding automaton 1/2

From a bound to a coding automaton

If there is a bound on the number of L-sensitive leaves in trees, then L is described by a coding automaton.

Let $\mathcal{A} = (Q, q_0, F, \delta)$ be the minimal tree automaton for L, and suppose K is a bound on the number of L-sensitive leaves.
Let $X = \{x_1, ..., x_n\}$. The profile $p_t \colon Q^X \to \{0, 1\}$ of a tree $t \in T_X$ is

$$p_t(q_{x_1}, ..., q_{x_n}) = 1 \text{ if and only if } t[x_1 \leftarrow q_{x_1}, ..., x_n \leftarrow q_{x_n}] \text{ is accepted.}$$

From a bound to a coding automaton 1/2

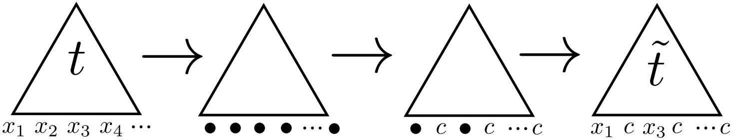
From a bound to a coding automaton

If there is a bound on the number of L-sensitive leaves in trees, then L is described by a coding automaton.

Let $\mathcal{A} = (Q, q_0, F, \delta)$ be the minimal tree automaton for L, and suppose K is a bound on the number of L-sensitive leaves.
Let $X = \{x_1, ..., x_n\}$. The profile $p_t \colon Q^X \to \{0, 1\}$ of a tree $t \in T_X$ is

$$p_t(q_{x_1}, ..., q_{x_n}) = 1 \text{ if and only if } t[x_1 \leftarrow q_{x_1}, ..., x_n \leftarrow q_{x_n}] \text{ is accepted.}$$

The reduct of tree t is the tree $\tilde{t}$ defined as follows



From a bound to a coding automaton 1/2

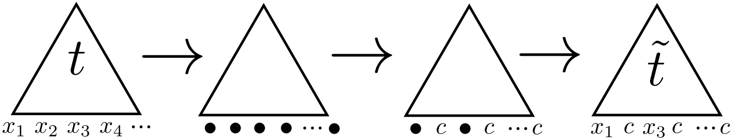
From a bound to a coding automaton

If there is a bound on the number of L-sensitive leaves in trees, then L is described by a coding automaton.

Let $\mathcal{A} = (Q, q_0, F, \delta)$ be the minimal tree automaton for L, and suppose K is a bound on the number of L-sensitive leaves.
Let $X = \{x_1, ..., x_n\}$. The profile $p_t \colon Q^X \to \{0, 1\}$ of a tree $t \in T_X$ is

$$p_t(q_{x_1}, ..., q_{x_n}) = 1 \text{ if and only if } t[x_1 \leftarrow q_{x_1}, ..., x_n \leftarrow q_{x_n}] \text{ is accepted.}$$

The reduct of tree t is the tree $\tilde{t}$ defined as follows



The reduct of a tree has at most K different variables.

From a bound to a coding automaton 2/2

Let s and t be Σ, X-trees. Suppose $p_{\tilde{t}} = p_{\tilde{s}}$, then

$$t[x_1 \leftarrow q_{x_1}, ..., x_n \leftarrow q_{x_n}] \in L \text{ if and only if } s[x_1 \leftarrow q_{x_1}, ..., x_n \leftarrow q_{x_n}] \in L \ .$$

From a bound to a coding automaton 2/2

Let s and t be Σ, X-trees. Suppose $p_{\tilde{t}} = p_{\tilde{s}}$, then

$$t[x_1 \leftarrow q_{x_1}, ..., x_n \leftarrow q_{x_n}] \in L \text{ if and only if } s[x_1 \leftarrow q_{x_1}, ..., x_n \leftarrow q_{x_n}] \in L \ .$$

Define an equivalence relation $\sim$ on trees by $s \sim t$ whenever $p_{\tilde{t}} = p_{\tilde{s}}$.

The set of tree modulo $\sim$ is orbit-finite and nominal.

From a bound to a coding automaton 2/2

Let s and t be Σ, X-trees. Suppose $p_{\tilde{t}} = p_{\tilde{s}}$, then

$$t[x_1 \leftarrow q_{x_1}, ..., x_n \leftarrow q_{x_n}] \in L \text{ if and only if } s[x_1 \leftarrow q_{x_1}, ..., x_n \leftarrow q_{x_n}] \in L\ .$$

Define an equivalence relation $\sim$ on trees by $s \sim t$ whenever $p_{\tilde{t}} = p_{\tilde{s}}$.

The set of tree modulo $\sim$ is orbit-finite and nominal.

Define a coding automaton as follows:

- the set of states is the set of trees modulo $\sim$,
- the transitions are defined so that, for all coding c, $\delta(q_0, c) = p_{\widetilde{T(c)}}$.

From a bound to a coding automaton 2/2

Let s and t be Σ, X-trees. Suppose $p_{\tilde{t}} = p_{\tilde{s}}$, then

$$t[x_1 \leftarrow q_{x_1}, ..., x_n \leftarrow q_{x_n}] \in L \text{ if and only if } s[x_1 \leftarrow q_{x_1}, ..., x_n \leftarrow q_{x_n}] \in L\ .$$

Define an equivalence relation $\sim$ on trees by $s \sim t$ whenever $p_{\tilde{t}} = p_{\tilde{s}}$.

The set of tree modulo $\sim$ is orbit-finite and nominal.

Define a coding automaton as follows:

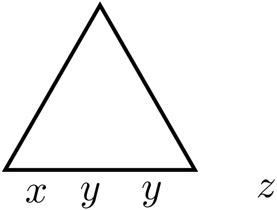
- the set of states is the set of trees modulo $\sim$,
- the transitions are defined so that, for all coding c, $\delta(q_0, c) = p_{\widetilde{T(c)}}$.

Decidability

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

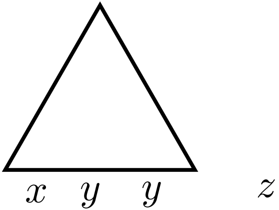
Different types of tree algebras

Unrestrained tree algebras

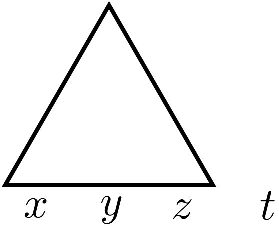


Different types of tree algebras

Unrestrained tree algebras

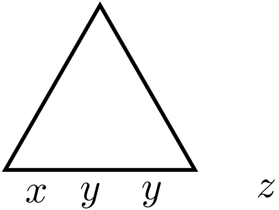


Sublinear tree algebras

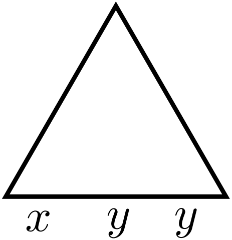


Different types of tree algebras

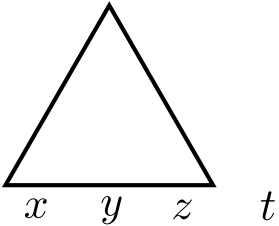
Unrestrained tree algebras



Superlinear tree algebras

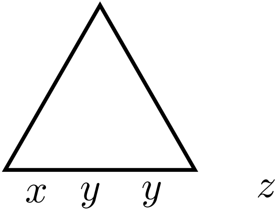


Sublinear tree algebras

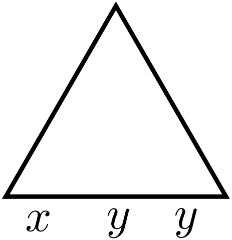


Different types of tree algebras

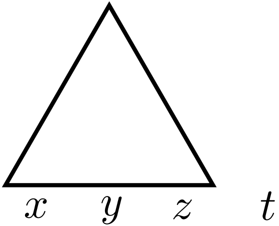
Unrestrained tree algebras



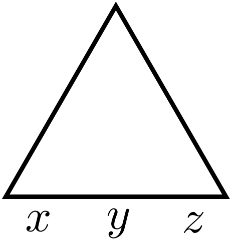
Superlinear tree algebras



Sublinear tree algebras

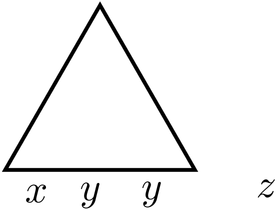


Linear tree algebras

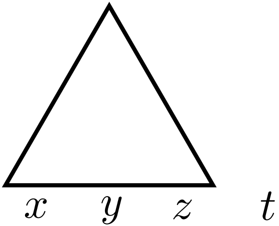


Different types of tree algebras

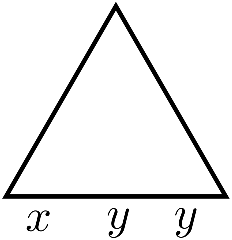
Unrestrained tree algebras



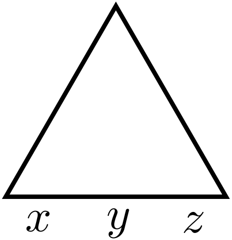
Sublinear tree algebras



Superlinear tree algebras



Linear tree algebras



Conclusion

Equivalence theorem

For a regular language of finite trees, the following properties are equivalent:

a. Being recognized by a finite tree algebra of polynomial complexity.
b. Being recognized by a finite tree algebra of bounded orbit complexity.
c. Being described by a coding automaton.

Conclusion

Equivalence theorem

For a regular language of finite trees, the following properties are equivalent:

a. Being recognized by a finite tree algebra of polynomial complexity.
b. Being recognized by a finite tree algebra of bounded orbit complexity.
c. Being described by a coding automaton.

Decidability

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

Conclusion

Equivalence theorem

For a regular language of finite trees, the following properties are equivalent:

a. Being recognized by a finite tree algebra of polynomial complexity.
b. Being recognized by a finite tree algebra of bounded orbit complexity.
c. Being described by a coding automaton.

Decidability

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

Future work. Tree algebras of exponential complexity.