RAIRO-Theor. Inf. Appl. 48 (2014) 431–451
DOI: 10.1051/ita/2014018

Available online at:
www.rairo-ita.org

TREE ALGEBRA OF SOFIC TREE LANGUAGES*

Nathalie Aubrun[1] and Marie–Pierre Béal[2]

Abstract. We consider the languages of finite trees called tree-shift languages which are factorial extensible tree languages. These languages are sets of factors of subshifts of infinite trees. We give effective syntactic characterizations of two classes of regular tree-shift languages: the finite type tree languages and the tree languages which are almost of finite type. Each class corresponds to a class of subshifts of trees which is invariant by conjugacy. For this goal, we define a tree algebra which is finer than the classical syntactic tree algebra based on contexts. This allows us to capture the notion of constant tree which is essential in the framework of tree-shift languages.

Mathematics Subject Classification. 68501, 37B10.

1. Introduction

Infinite k-ary trees have a natural structure of symbolic dynamical systems equipped with k shift transformations [1]. The ith shift transformation applied to a tree gives the subtree rooted at the child number i of the tree. A tree subshift is described by a set of finite block trees which are forbidden, *i.e.* which never appear as factor of some infinite tree of the subshift.

The set of trees which are factors of a subshift is a language of finite ranked trees is called a *tree-shift language*. It is closed and stable by any shift transformation. A tree-shift language is a language of finite ranked trees which is both *factorial*

Keywords and phrases. Symbolic dynamics, tree-shift, tree automata, tree algebra.

* *This work is supported by the French National Agency (ANR) through "Programme d'Investissements d'Avenir" (Project ACRONYME No. ANR-10-LABX-58) and through the ANR EQINOCS.*

[1] LIP, UMR 5668, ENS de Lyon, CNRS, France. nathalie.aubrun@ens-lyon.fr

[2] Université Paris-Est, Laboratoire d'informatique Gaspard-Monge, UMR 8049 CNRS, France. beal@univ-mlv.fr

Article published by EDP Sciences

© EDP Sciences 2014

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

and *extensible*. Hence if L is a tree-shift language, any subtree which a sub-pattern of a tree of L belongs to L (factoriality property). Furthermore, children can be added to all leaves of a tree of L so that the new tree belongs also to L (extensibility property). This tree-shift language characterizes the associated tree-subshift and interesting properties of the subshift can be read in its associated tree-shift language. The simplest class of these languages is the class of tree languages of finite type which corresponds to subshifts defined by a finite set of forbidden block trees (or patterns). Languages of finite patterns of tree-shifts of finite type are strictly locally testable tree languages [21] (also called k-testable tree languages, or k-grams in the case of sequences). For these languages, the effect of events that occurred beyond a certain depth window are ignored when processing a tree. Probabilistic k-testable models are used for pattern classification and stochastic learning [21].

In [1,3], we proved that the topological conjugacy of tree subshifts of finite type is decidable, thus extending Williams's conjugacy theorem for one-sided shifts of sequences [17]. A larger class of tree languages is the class of sofic tree languages (also called regular tree-shift languages), which corresponds to sofic subshifts of trees. Sofic tree-shifts have been studied in [2, 4, 10, 13]. These tree languages are accepted by essential tree automata where all states are both initial and final [2]. Among this class, the almost of finite type tree-shift languages have the property of being accepted by a (bottom-up) tree automaton which is both deterministic and co-deterministic with a finite delay. The corresponding class of subshifts constitutes a meaningful intermediate class in between irreducible tree-shifts of finite type and general sofic tree-shifts (see [9, 17]). In [2, 4], we have shown any irreducible sofic tree-shift has a minimal presentation which is synchronized. We also described an algorithm for checking whether a sofic tree subshift is almost of finite type. Almost of finite type shifts enjoy various properties which are not shared by all sofic shifts. It is the one big, natural class of nice sofic shifts.

In this paper, we give syntactic characterizations of tree-shift languages of finite type and of almost of finite type tree-shift languages. A syntactic characterization of almost of finite type word languages has been obtained in [5]. Logics for sofic and finite type multi-dimensional subshifts have also been explored in [16].

We first consider the three sorted syntactic tree algebra introduced by Wilke in [22] and we easily derive characterizations of tree-shift languages in this algebra. The concept of tree-shift languages of finite type is close to the notion of definite tree languages and forests studied in [6, 7, 14, 18], to the notion of frontier testable (also called reverse definite) tree languages in [22], and to the notion of generalized definite tree languages [15]. It is weaker than the notion of locally testable tree languages [19]. In [14], Heuter showed that it is decidable whether a regular tree language is definite. Nivat and Podelski obtained a syntactic characterization of this property in [18].

All these properties are however distinct from the finite type condition that we consider here. In order to characterize the tree-shift languages of finite type, we introduce the important notion of constant tree known for sequences [12]. However,

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

we show that the notion of constant tree is not well-captured in the syntactic tree algebra of [22]. We thus define a stronger tree algebra which is still computable and finite for regular languages. In this strong algebra, we give effective characterizations of tree languages of finite type and of tree languages which are almost of finite type. In the last section, we show that this algebra may be refined again while remaining finite and computable for regular languages.

The paper is organized as follows. The background of shifts of infinite trees is presented in Section 2. Basic notions of finite trees and contexts are recalled in Section 3. The notion of Wilke's tree algebra is briefly presented in Section 3.3 and a syntactic characterization of factorial extensible tree languages is given in Section 3.4. The notion of strong tree algebra is introduced in Section 4. The main results, the characterizations of tree-shift languages of finite type and almost of finite type are obtained in Sections 5.2 and 6. Both results use the notion of constant tree and constant tree class explained in Section 5.1.

2. Shifts of infinite trees

In this section we recall some basic definitions from symbolic dynamics concerning shifts of infinite trees which motivate this paper (see [4]). We consider infinite trees where each node has a fixed number k of children and is labeled in a finite alphabet. If $\Sigma = \{0, 1, \ldots, d-1\}$ is a finite alphabet of cardinality d, an *infinite tree* t over a finite alphabet A is a total function from Σ^* to A and t_x is the label of t at the node x.

A *finite pattern of an infinite tree* t is a function $p : P \to A$, where P is a finite prefix-closed subset[3] of Σ^* and such that $t_{xy} = p(y)$ for any $y \in P$. If p is not a pattern of t, one says that t *avoids* p.

A *tree-shift* over A is the set X_F of infinite trees over A avoiding a given set F of (forbidden) patterns. When the set F can be chosen finite, the tree-shift is said to be *of finite type* and when F is a regular set of finite trees, the tree-shift is said to be *sofic*.

An equivalent definition [4] can be obtained as follows. We define the shift transformations σ_i for $i \in \Sigma$ of infinite trees over A as follows. If t is a tree, $\sigma_i(t)$ is the tree rooted at the ith child of t, *i.e.* $\sigma_i(t)_x = t_{ix}$ for any $x \in \Sigma^*$.

A tree-shift over A is a set X of infinite trees over A such that $\sigma_i(X) = X$ for any $i \in \Sigma$ and which is closed for the product topology of the discrete topology on A.

The set of finite patterns of trees of a tree-shift X is a language of finite trees which is both factorial and extensible (see definitions below). Conversely, if L is a factorial and extensible language of finite trees, the set of infinite trees whose finite patterns belong to L, is a tree-shift.

[3] any prefix of a word of P belongs to P.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

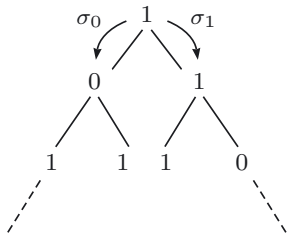


FIGURE 1. A tree of the tree-shift of finite type $X = \mathsf{X}_F$ on the alphabet $\{0, 1\}$, where F is the set of blocks of height 2 whose label sum is 1 modulo 2.

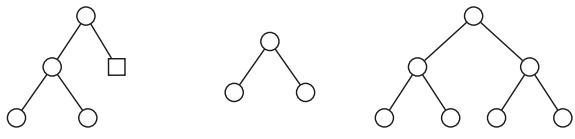
Many properties of tree-shifts can be viewed on their associated sets of finite patterns and the goal of this paper is to study these properties in algebra of finite trees.

3. Finite trees and contexts

3.1. Binary trees

The trees in the remaining of this paper are finite, labeled and have a fixed arity. We will consider only (complete) *binary trees* (each node has zero or two children) but all results extend to k-ary trees where k is a nonnegative integer. Formally, if Σ denotes the alphabet $\{0, 1\}$ and A is a finite alphabet, a tree is a partial function from Σ^* to A with a finite domain such that such that for each node x, $x0, x1$ are either both not in the domain or both in the domain. This implies that the domain is prefix-closed. A set L of trees over a given alphabet A is called a *tree language* over A.

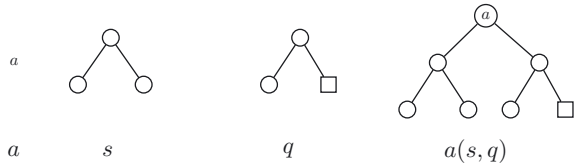
The one-node tree made with label a is denoted by (a), or simply by a when there is no confusion with the label a. The empty tree is not allowed. If a is a label and s, t are trees, then $a(s, t)$ denotes the tree rooted by a node labeled by a, with left child s and right child t. If we take a tree and replace one of the leaves by a special symbol $\Box$ called the hole, we obtain a *context*. The empty context, where the only node is the hole, is denoted by $\Box$. A tree s can be substituted in place of the hole of a context p, the resulting tree is denoted by ps, as illustrated below:



We will write pa instead of $p(a)$ for a context p and a letter a.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

There is a natural composition operation on contexts: the context qp is formed by replacing the hole of q with p. This context concatenation satisfies $(pq)s = p(qs)$ for all trees s. We also allow constructing contexts from a label a, a tree s, and context q. The resulting context is denoted by $a(s, q)$, as illustrated below:



It satisfies $a(s, q)t = a(s, qt)$ for all trees t. The context $a(p, t)$ is defined symmetrically. Note that $a(s, t)$ is the tree $a(s, qb)$ for any context q and label b such that $qb = t$. It is also equal to the tree $a(pc, t)$ for any context q and label c such that $qc = s$.

Borrowing conventions used in [6–8], in the implications and identities, the letters used for variables will implicitly identify the type of the variables. Trees will be denoted by the letters $s, t, u, ..$ Labels of nodes will be denoted by a, b, c. Contexts will be denoted using letters p, q, r. We use letters A, B, C to denote the alphabets, *i.e.* finite sets of labels. We use x, y, z to denote nodes, *i.e.* words in Σ^* included in the domain of some tree. We shall denote by $\mathcal{T}$ and $\mathcal{C}$ the set of trees and contexts respectively.

3.2. Sofic trees

A (bottom-up) *tree automaton* (see for instance [11]) works as follows. Fix a finite input alphabet A. The tree automaton has a finite set of states Q, a set of initial states I, a set of final states F, and a finite set of transitions of the form $(q_0, q_1) \xrightarrow{a} q$, where $q_1, q_2, q \in Q$ and $a \in A$. A *computation* (or a *run*) of the tree automaton on a tree t labeled on A is a tree s labeled on Q which is consistent with the transition function in the following sense. The domain of s is the domain of t plus the nodes $x0$ and $x1$ for each leaf x of t. If x is a node of s labeled by q with children $x0, x1$ labeled by q_0, q_1 respectively, then there is a transition $(q_0, q_1) \xrightarrow{t_x} q$, where t_x is the label of t at the node x. A computation s is *accepting* if its leaves are labeled by initial states and its root is labeled by a final state. The tree t is said to be *accepted* by the tree automaton. The set of trees accepted by the tree automaton is also called the language *recognized* by the tree automaton. A tree language is *regular* if it is recognized by some tree automaton. A tree automaton also accepts contexts (seen as incomplete binary trees without their box).

A state q of a tree automaton is *accessible* if there is a letter a and states q_0, q_1 such that $(q_0, q_1) \xrightarrow{a} q$ is a transition. A tree automaton is *essential* when all its states are accessible. A *sofic* tree language is a regular tree language accepted by an essential tree automaton where all states are *both initial and final*. Such a tree automaton is denoted (Q, A, Δ), where Δ is the set of transitions. Note that all its computations are accepting since we have assumed that all states are both initial

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

and final. The *full language* of trees is the set of all trees of A. It is called the *full tree-shift*.

An infinite (binary and complete) tree is a total map from Σ^* to A. If $x \in \Sigma^*$ and $E \subseteq \Sigma^*$, we denote by $x \cdot E$ the set of $\{xy \in \Sigma^* \mid y \in E\}$. A *factor* (or *subtree*) of some (finite or infinite) tree t with domain D is a finite tree s with domain E such that there is a node x of t such that $x \cdot E \subseteq D$ and s and t coincide on the domain of s. A *factorial* language of trees is a language of (finite) trees which is closed by factors, *i.e.* if $s \in L$ and t is a factor of s, then $t \in L$.

If s, t are trees, we say that $s \prec t$ if, for any node x in the domain of s, $x0$ and $x1$ belong to the domain of t, and s and t agree on the domain of s. Note that all leaves of s are extended strictly in all directions. Thus $s \not\prec s$. Roughly speaking, the extension is a "fat" extension. A language of trees L is *extensible* if and only if, for any tree s in L, there is a tree t in L such that $s \prec t$. A *tree-shift language* is a factorial extensible tree language.

It is shown in [2, 4] that the sofic tree languages are exactly the regular tree-shift languages. Moreover, regular tree-shift languages are the set of factors of sofic shifts of infinite trees.

3.3. Syntactic congruence

An equivalent definition of regular trees uses the Myhill–Nerode *syntactic congruence* introduced by Wilke in [22]. This congruence is a *three-sorted algebra* (labels, trees, contexts) called *Wilke's tree algebra* (see [20, 22]). It consists of three congruences, one for the trees, one for the contexts and one for the labels (seen as labels off internal nodes or as construction operators denoted $a(,)$).

Let L be a tree language. Two trees s, s' are called *equivalent under* L, written $s \sim_L s'$, if

$$ps \in L \Leftrightarrow ps' \in L$$

holds for every context p. Two contexts q, q' are called *equivalent under* L, written $q \sim_L q'$, if and only if, for any tree t, the two trees qt and $q't$ are equivalent under L. When the language L in question is clear from the context, we omit the subscript $\sim_L$ and simply write $\sim$. Two labels a, a' are called *equivalent under* L, written $a \sim_L a'$, if the two conditions

$$\begin{aligned} &pa(s,t) \in L \Leftrightarrow pa'(s,t) \in L, \\ &pa \in L \Leftrightarrow pa' \in L. \end{aligned}$$

hold for every context p and any trees s, t (see [20]).

Using standard techniques, one can show that a tree language is regular if and only if its three syntactic equivalences have finite index. Note that $\sim_L$ on A has always a finite index since A is finite.

The syntactic equivalences form a congruence with respect to the operations

$$a(s,t) \quad a(p,t) \quad a(s,q) \quad pq \quad ps \tag{3.1}$$

on trees s, t, contexts p, q and labels a, as defined above.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

Two elements (labels, trees, or contexts) are equivalent if they are of the same sort and relate to L in the same way in every possible context. We respectively denote by A, T and C the sets of classes of labels, trees, contexts in the syntactic algebra of L. If s is a tree, p a context, a a label, we will denote by $[s]$, $[p]$ and $[a]$ their class in this three-sorted algebra.

We define a partial order $\leq$ on trees as follows. If s is a tree, we define the *context* of the tree as the set (denoted $\mathrm{cont}(s)$) of contexts p such that $ps \in L$. Let s, t be trees. We set $s \leq t$ if and only if $\mathrm{cont}(s) \subseteq \mathrm{cont}(t)$. Note that $[s] = [t]$ if and only if $\mathrm{cont}(s) = \mathrm{cont}(t)$. We thus set $[s] \leq [t]$ if $\mathrm{cont}(s) \subseteq \mathrm{cont}(t)$. If $[s]$ is a tree class we call *context* of $[s]$ the set of context classes $[p]$ such that $ps \in L$.

3.4. Characterization of tree-shift languages

The goal of the paper is to give algebraic characterizations of some classes of tree-shift languages. These characterizations are not purely syntactic but dot-syntactic (*i.e.* with the use of the zero classes for non full tree-shift languages). They may be used to obtain decidable characterizations of several classes of sofic languages.

We first present the characterization of factorial extensible tree languages (*i.e.* tree-shift languages). This characterization is easy to obtain in the three-sorted algebra. The computation of all these properties can be done in the finite syntactic algebra for regular tree languages.

If L is a tree language, we denote by $\mathrm{Im}(L)$ the classes which are Images of trees of L in the tree algebra.

Proposition 3.1. *A tree language is factorial if and only if its syntactic algebra satisfies the implication:*

$$v\alpha(g,h) \in \mathrm{Im}(L) \quad \Rightarrow \quad v\alpha, \alpha, \text{ and } \alpha(g,h) \in \mathrm{Im}(L) \tag{3.2}$$

for any context class v, any tree classes g, h, and any label class α.

Proof. If the tree language L is factorial, implication (3.2) comes from the definition of factorial. Conversely, let us assume that implication (3.2) is true. Let s be a factor of a tree t in L. Then $t = pu$ where the domain of s is included in the domain of u. We have either $u = a$ or $u = a(u_1, u_2)$. Since $[pu] \in \mathrm{Im}(L)$, $[a], [a]([u_1], [u_2]) \in \mathrm{Im}(L)$. Thus $[u] \in \mathrm{Im}(L)$. Thus we can assume that s is a factor of t at the root of t. Let us assume that $s \neq t$. Then there is a context q and trees u_3, u_4 such that $t = qb(u_3, u_4)$ and the domain of s is included in the domain of $t' = qb$. By implication (3.2), $[t'] \in \mathrm{Im}(L)$ and t' has less nodes than t. By iterating this operation, we get that $[s] \in \mathrm{Im}(L)$. Hence $s \in L$. □

Proposition 3.2. *A factorial tree language is extensible if and only if its syntactic algebra satisfies the implication:*

$$v\alpha \in \mathrm{Im}(L) \quad \Rightarrow \quad \exists\, g, h \in T \;\; v\alpha(g,h) \in \mathrm{Im}(L) \tag{3.3}$$

for any context class v and any label class α.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

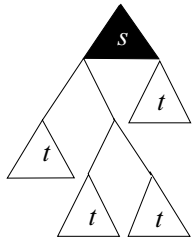


FIGURE 2. A tree u such that the tree s is a subtree of u at the root of u and the tree t is a subtree of u at each node x of the complete prefix code $\{00, 010, 011, 1\}$.

Proof. Let us assume that L is factorial and extensible. Let $v\alpha \in \mathrm{Im}(L)$. Let p, a with $[p] = v$ and $[a] = \alpha$. There is a tree $s \in L$ with $pa \prec s$. Hence there are trees t, u such that the domain of $pa(t, u)$ is included in the domain of s. Since L is factorial, $pa(t, u) \in L$, thus implication (3.3) holds with $g = [t]$ and $h = [u]$. Conversely, let s be a tree of L and a be the label of a leaf of s, E being the set of the other leaves of s. Then $s = pa$ and there are classes g, h such that $[p][a](g, h) \in \mathrm{Im}(L)$. Let t, u with $g = [t], h = [u]$. We get $pa(t, u) \in L$. We iterate this operation with the other leaves of E, which are also leaves of $pa(t, u)$, and get a tree $s' \in L$ such $s \prec s'$. □

When a tree-shift language L is not the full language, we denote by 0 the equivalence class of the trees that do not belong to L. We also denote by 0 the equivalence class of the contexts p such that, for any tree s, $ps \notin L$, and the equivalence class of any label a for which $pa(s, t) \notin L$ for any context p and trees s, t. Note the consistency of these definitions: $0[s] = 0$ for any tree s.

Transitive tree languages form an important class of tree-shift languages which contains the almost of finite type tree-shift languages of Section 6. The notion of transitivity suitable for tree-shift languages was introduced in [2]. A *finite complete prefix code* of Σ^* is a prefix-free set[4] X of finite words in Σ^* such that any word in Σ^* which is longer than the words of X has a prefix in X. This set is a set of uncomparable nodes with a node on each path in the tree.

A tree-shift language L is *transitive* if for each pair of trees $s, t \in L$ there is a tree $u \in L$ and a finite complete prefix code $X \subset \Sigma^*$ with words of length at least the height of s, such that s is a subtree of u rooted at ε, and t is a subtree of u rooted at x for each $x \in X$ (see Fig. 2).

This notion generalizes the notion of transitivity for subshifts in a strong way (remind that a subshift with language L is transitive if $u \in L$ and $w \in L$ implies the existence of a v such that $uvw \in L$), since we impose the transitivity along all possible paths of a certain finite complete prefix code.

[4] *i.e.* no word is prefix of another one.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

For the sake of completeness, we give a syntactic characterization of this property. Let P be a subset of a tree language. We denote by $A^*(P)$ the set of trees obtained by taking any tree in $\mathcal{T}$ and replacing its leaves with some tree in P. For a tree class h in T, we denote by $A^*(h)$ the set of classes of trees in $A^*(P)$ where P is the set of trees whose class is h.

Proposition 3.3. *A tree-shift language is transitive if and only if its syntactic algebra satisfies the following property.*

$$v \neq 0,\ h \neq 0 \quad \Rightarrow \quad \exists\, g \in A^*(h) \text{ such that } vg \neq 0$$

for any context class v and any tree class h.

Proof. Assume that the tree-shift language L is transitive. Let p, a, s with $[pa] \neq 0$, $[p] = v$ and $[s] = h$. By transitivity there is a tree t in $A^*(s) \cap L$ such that the domain of pa is contained in the domain of t. Moreover, if x denotes the box position of p, one can choose t such that the subtree u of t rooted at x also belongs to $A^*(s)$. Setting $g = [u]$, we get $g \in A^*(h)$ and $vg \neq 0$ since $pu \in L$.

Conversely, let s, t be trees of L. Let $s' \in L$ such that $s \prec s'$. Let a be the label of one leaf of s' and E be the set of the other ones. Then $s' = pa$. We note $v = [p]$ and $h = [t]$. There is a class $g \in A^*(h)$ such that $vg \neq 0$ and $t' \in A^*(t)$ such that $[t'] = g$. The leaves in E are also leaves of the tree pt'. By iterating the construction with the other leaves we obtain that any leaf of s can be extended into a tree in $A^*(t)$, the whole tree staying in L. This proves the transitivity. □

The computation of the above property for regular tree languages is based on the computation of the subset $A^*(h)$. We set $A^0(h) = \{h\}$, $A(h) = A^0(h) \cup \{a(h, h) \mid a \in A\}$. For any positive integer n, we define $A^n(h) = A^{n-1}(h) \cup \{a(f, g) \mid f, g \in A^{n-1}(h)\}$. Since the algebra is finite, there is an integer n such that $A^n(h) = A^{n+1}(h)$. This set is equal to $A^*(h)$.

Note that Propositions 3.1–3.3 hold for the full tree language since the full language is factorial, extensible and transitive.

4. Strong tree algebra

We assume that L is a tree-shift language. The image of trees not belonging to L in the tree algebra is thus 0. We denote by T^0 the set of tree classes distinct from 0. We will moreover assume that L is regular in order to work with a finite tree algebra.

Since the equivalence $\sim$ will be too weak to characterize synchronizing properties of tree-shift languages, we introduce the notion of *strong tree algebra* of tree-shift languages. We define a strong equivalence on trees, denoted $\approx$, as follows.

Two trees s, t are called *strong equivalent under* L, written $s \approx t$ if and only the set of equivalence classes of s' for $s \prec s'$ is equal to the set of equivalence classes of t' for $t \prec t'$.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

We denote by $[\![s]\!]$ the class of s for $\approx$. We write $[\![s]\!] \leq [\![t]\!]$ when the set of equivalence classes of s' for $s \prec s'$ is included into the set of equivalence classes of t' for $t \prec t'$. Note that $[\![s]\!] \leq [\![t]\!]$ implies $[s] \leq [t]$. Indeed, let p such that $ps \in L$. Since L is a tree-shift language, there is a tree $u \in L$ such that $ps \prec u$. Hence there is a tree s' such that $s \prec s'$ and $ps' \in L$. Thus there is a tree t' with $t \prec t'$ and $pt' \in L$, which implies $pt \in L$ since L is factorial.

Two contexts p, q are called *strong equivalent under* L, written $p \approx q$, if and only if, for any tree t, the two trees qt and $q't$ are strong equivalent under L. Finally, the strong equivalence coincides with the equivalence $\sim$ for labels. If p is a context, we denote by $[\![p]\!]$ the strong class of p. The null class is still denoted by 0.

One can easily check the following properties.

$$\begin{aligned} s \approx s' &\Rightarrow ps \approx ps', && (4.1)\\ a \approx a', s \approx s', t \approx t' &\Rightarrow a(s,t) \approx a'(s',t'), && (4.2)\\ a \approx a', p \approx p', t \approx t' &\Rightarrow a(p,t) \approx a'(p',t'), && (4.3)\\ a \approx a', s \approx s', q \approx q' &\Rightarrow a(s,q) \approx a'(s',q'), && (4.4)\\ p \approx p', q \approx q' &\Rightarrow pq \approx p'q', && (4.5)\\ p \approx p', s \approx s' &\Rightarrow ps \approx p's'. && (4.6) \end{aligned}$$

We respectively denote by A, $\mathbb{T}$ and $\mathbb{C}$ the sets of classes of labels, trees, contexts in this three-sorted strong tree algebra of L.

We define two special tree automata accepting tree-shift languages. The first one, the context tree automaton, is a deterministic tree automaton whose states are identified with nonnull tree classes. All its states are both initial and final. The second one, called the determinized context tree automaton, has a unique initial state and is obtained by determinization of the previous one.

The *context tree automaton* of a tree-shift language L is the (incomplete) deterministic tree automaton denoted by (T^0, A, Δ), where T^0 is the set of nonnull tree classes of L. All states of this tree automaton are initial and final. The transitions of Δ are $([s],[t]) \xrightarrow{a} [a(s,t)]$, where s, t are trees and $[a(s,t)]$ is nonnull. Since L is factorial and extensible, the automaton recognizes the language L. Indeed, if $s \in L$, we can extend s to a tree t by extending each leaf e of s with trees s_e (on the left) and t_e (on the right). Hence there is a computation of the context tree automaton on s starting with states $[s_e], [t_e]$, for all leaves e of s, and rooted with $[t]$. This proves that the context tree automaton accepts s. Conversely, if s is accepted by the context tree automaton, there is a computation of the tree automaton on s. Its root is some state $[t] \neq 0$ where $s \prec t$. Since t belongs to L, then s also.

The *determinized context tree automaton* is the deterministic tree automaton $(\mathfrak{P}(T^0), A, \delta, i, F)$ whose set of states are the subsets of T^0, with a unique initial state $i = T^0$, $F = \mathfrak{P}(T^0)$, and with transitions $(P,Q) \xrightarrow{a} \{a(g,h) \mid g \in P, h \in Q, a(g,h) \neq 0\}$. It accepts the language L. If T is finite, $\mathfrak{P}(T^0)$ is finite. Thus this tree automaton is finite if L is regular. We denote $\delta(i,s)$ by $I(s)$.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

Lemma 4.1. *Let s be a tree. We have $I(s) = \{[t] \mid s \prec t$ and $[t] \neq 0\}$.*

Proof. It is clear that if t is a tree such that $[t] \neq 0$ and $s \prec t$, then $[t] \in I(s)$. Conversely, if $g \in I(s) = \delta(i, s)$, then $g \neq 0$ and there is computation of s in the context tree automaton whose root is g. The leaves of this computation being nonnull classes of trees, g is the class of a tree t such that $s \prec t$. □

As a consequence of Lemma 4.1 we obtain the following corollaries.

Corollary 4.2. *Let s, t be two trees. We have $[\![s]\!] \leq [\![t]\!]$ if and only if $I(s) \subseteq I(t)$ and thus $[\![s]\!] = [\![t]\!]$ if and only if $I(s) = I(t)$.*

Corollary 4.3. *The strong tree algebra of a tree-shift language has finite index if and only if the language is regular.*

Proof. The strong equivalence is finer that the equivalence $\sim$. Thus it has an infinite number of classes when the language L is not regular. Conversely, if L is regular, the number of strong tree classes is bounded above by the number of states of the determinized context tree automaton. □

Let n be a positive integer. A *block tree* of height n is a tree whose domain is the set of all words of Σ^* of length at most $n-1$. We denote by T_n (resp. $T_{\geq n}$) the set of block trees of height n (resp. greater than or equal to n) and by $\mathbb{T}_n$(*resp.* $\mathbb{T}_{\geq n}$) the set of strong equivalence classes of trees in T_n (resp. $T_{\geq n}$). We denote by $\mathbb{T}^\omega$ the set of strong classes g such that, for any integer m, there is an integer $n \geq m$ and a block tree s of height n such that $g = [\![s]\!]$. We define $\mathbb{T}^\omega = \cap_{m \geq 1} \mathbb{T}_{\geq m}$. If the number of strong tree classes is finite, the decreasing sequence $(\mathbb{T}_{\geq m})_{m \geq 1}$ stabilizes. Hence there is a positive integer m such that $\mathbb{T}^\omega = \mathbb{T}_{\geq m}$. Similarly $\mathbb{C}^\omega$, is the set of strong context classes v such that, for any integer n, there is a context p and a tree s of height greater than n with $[\![p]\!] = v$ and ps is a block tree.

5. Tree-shift languages of finite type

5.1. Constant trees

In this section, we introduce the notion of constant tree (or intrinsically synchronizing tree). It corresponds to the notion of constant of a semigroup which is used to capture the synchronization properties of word languages (see [12]), or to the notion of intrinsically synchronizing word of symbolic dynamical systems ([17], Exercise 3.3.4, p. 85).

Let L be a tree-shift language[5]. A tree s is a *constant for* $P \subseteq L$ if the following implication holds.

$$pt, qu \in P \qquad \Rightarrow \qquad pu, qt \in P.$$

for any contexts p, q and any trees t, u with ($s \prec t$ or $s = t$) and ($s \prec u$ or $s = u$). A tree is a *constant* if it is a constant for L (see Fig. 3).

[5]The notion of constant tree can be defined for tree languages which are not tree-shift languages but we restrict us here to tree-shift languages.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

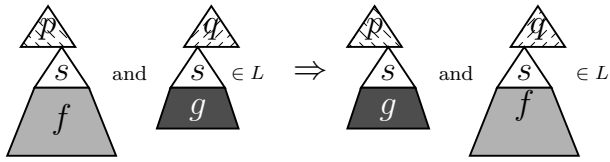


FIGURE 3. A constant tree s.

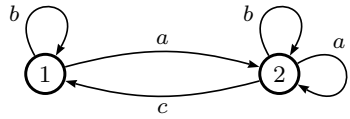


FIGURE 4. A finite automaton defining a path-testable tree language. All states are both initial and final.

Proposition 5.1. *Let s be a tree. Then s is a constant if and only if either $[s] = 0$ or $[t] = [s]$ for any tree t such that $s \prec t$ and $[t] \neq 0$.*

Proof. Assume that s is a constant and $[s] \neq 0$. Let t such that $s \prec t$ and $[t] \neq 0$. Let p be a context. If $pt \in L$, then $ps \in L$ since L is factorial. Let us assume that $ps \in L$. Since $[t] \neq 0$, $\Box \cdot t \in L$. Since s is a constant, we get $pt \in L$, and thus $[s] = [t]$.

Conversely, if $[s] = 0$, then s is trivially a constant. Let us assume that $[s] \neq 0$ and $[t] = [s]$ for any tree t such that $s \prec t$ and $[t] \neq 0$. Let p, q be contexts and t, u be trees with ($s \prec t$ or $s = t$) and ($s \prec u$ or $s = u$). Since $[s] \neq 0$, we also have $[t] \neq 0$ and $[u] \neq 0$ by factoriality of L. Assume that $pt, qu \in L$. Since $[s] = [t] = [u]$, we get $ps, qs \in L$ and $pu, qt \in L$. □

In particular, if s is constant and $s \prec t$, then t is constant. Similarly, a context p is a *constant* if and only if $[ps] = [pt]$ for any trees s, t such that $[ps] \neq 0$ and $[pt] \neq 0$.

The following proposition gives a characterization of constant trees. It is a corollary of Lemma 4.1.

Proposition 5.2. *Let s be a tree. Then s is a constant if and only if either $s \notin L$ or $\mathrm{card}(I(s)) = 1$.*

Note that some tree classes in the syntactic tree algebra may contain both constant and non constant trees. Indeed, let us consider the path-testable tree language (see [6]) whose trees have their branch labels (read from the leaves to the root) accepted by the finite automaton of Figure 4 where all states are both initial and final. In the three-sorted algebra, we have $[b] = [a]$. The tree a is constant while b is not. Indeed, $a(b(c, c), a), c(b(a, a), a)$ belong to the tree language but $c(b(c, c), a)$ does not. Hence the notion of constant is not well-captured in the syntactic tree algebra.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

We now define the notion of constant class in the syntactic tree algebra and in the strong tree algebra as follows. A *constant class* of T (resp. C) is a class for the equivalence $\sim$ containing only constant trees (resp. constant contexts). Similarly, a *constant class* of $\mathbb{T}$ (resp. $\mathbb{C}$) is a class for the equivalence $\approx$ containing only constant trees (resp. constant contexts).

As a consequence of Proposition 5.2 and 4.2, if s, t are trees and $[\![s]\!] = [\![t]\!]$, then s is a constant if and only if t is constant.

Corollary 5.3. *Let $[\![s]\!]$ be a strong tree class. Then $[\![s]\!]$ is constant if and only $[\![s]\!] = 0$ or $\mathrm{card}(I(s)) = 1$.*

A 0-*minimal* class (for the equivalence $\sim$) of trees is a nonnull class which is minimal for the partial order $\leq$. The following proposition proves the existence of nonnull constant tree classes in the syntactic tree algebra (and thus in the strong tree algebra).

Proposition 5.4. *Any 0-minimal tree class is constant.*

Proof. Let us assume that h is a nonnull minimal tree class for the equivalence $\sim$. We show that any tree of this class is constant.

If h is not constant, let s in h which is not constant. Let h_1, h_2 in $I(s)$ with $h_1 \neq h_2$. We have $h_1 \in I(s)$, $h_1 \leq h$ and $h_1 \neq 0$. Since h is 0-minimal, we get $h_1 = h$. Similarly $h_2 = h$ and thus $h_1 = h_2$, a contradiction.

Let us assume that $h_1 \neq h = [s]$. Then there is a context class v such that $vh_1 = 0$ and $vh \neq 0$. Hence $h_1 < h$, a contradiction. □

5.2. Characterization of tree-shifts of finite type

A tree-shift language is *of finite type* if it is defined by a finite set of forbidden factors, *i.e.* there is a finite set of trees F such that a *block tree* belongs to the language if and only if it does not contain any tree in F as factors. Note that the property applies only for block trees.

For instance if F is the set of all blocks of height 2, the tree-shift language obtained is the empty language. Thus the empty language (which is allowed as a tree-shift language) is of finite type.

Proposition 5.5. *A a tree-shift language is of finite type if there is a positive integer m such that any block tree of height m is a constant tree.*

Proof. If the tree-shift language L is of finite type, there is a finite set of trees F such that a block tree belongs to L if and only if it does not contain any tree in F as factors. Let m be the maximal height of trees in F. Let s be a block tree of height m. Let p, q be contexts and t, u be trees with $(s \prec t$ or $s = t)$ and $(s \prec u$ or $s = u)$. Assume that $pt, qu \in L$. There are contexts p', q' and trees t', u' such that $(pt \prec p't')$ or $(pt = p't')$ and $(qu \prec q'u')$ or $(qu = q'u')$ and $p't', q'u'$ are block trees. Since the property of belonging to L can be checked by scanning all subtrees

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

of a block tree which are blocks of height at most m, we get $p'u', q't' \in L$ and thus $pu, qt \in L$.

Conversely, assume that there is a positive integer m such that any block tree of height m is a constant tree. We set F as the set of block trees of height at most $m+1$ which do not belong to L. Then a block tree of L does not contain any tree in F as factors. Let s be a block tree which does not contain any tree in F as factors. If the height of s is at most $m+1$, then s belongs to L by definition of F. Let us show by recurrence on the height of s that $[s] = [t]$ where t is the block of height m subtree of s at the root of s. If the height of s is $m+1$ and since $[s] \neq 0$, we have $[s] = [t]$ where t is the block of height m subtree of s at the root of s. If s is of height $h > m+1$, then $s = a(s_1, s_2)$ where s_1, s_2 are block trees of height $h-1$. By recurrence hypothesis, $[s] = [s']$ with $s' = a(t_1, t_2)$, where t_1, t_2 are blocks of height m subtrees of s_1 and s_2 respectively at the root. Furthermore, $s' \in L$ since it is a block tree of height $m+1$ factor of s. Applying the result for this block tree of height $m+1$, we get that $[s'] = [t]$ where t is the block of height m subtree of s' at the root of s'. Thus $[s] = [t]$ where t is the block of height m subtree of s at the root of s and as a consequence $s \in L$ since $[t] \neq 0$. □

A tree-shift language of finite type is regular and a regular tree-shift language is of finite type if and only if it is recognized by an essential deterministic *local tree automaton*, where all states are initial and final (see [2]). The locality of a tree automaton is defined as follows. Let m be a positive integer. A *deterministic m-local tree automaton* is a tree automaton $\mathcal{A}$ such that any two computations of $\mathcal{A}$ on a same block tree of height m have the same root. A tree automaton is *local* (or *definite*) if it is m-local for some nonnegative integer m (and m stands for memory).

The notion of tree-shift languages of finite type is close to the notion of definite languages of trees and forests given in [6, 7, 14, 18], for which the membership depends only on the nodes of height at most n. It is however different and a syntactic characterization of finite type tree-shift languages may not be obtained in the syntactic tree algebra[6] but in the strong tree algebra.

Theorem 5.6. *A regular tree-shift language is of finite type if and only if the following property holds in the strong tree algebra,*

$$[\![s]\!] \in \mathbb{T}^\omega \quad \Rightarrow \quad [\![s]\!] \text{ is constant.}$$

Proof. Let L be a regular tree-shift language of finite type. There is a positive integer m such that every block tree of height m is constant. Let $[\![s]\!] \in \mathbb{T}^\omega$ which is nonnull. There is a block tree t in L of height greater than m such that $[\![t]\!] = [\![s]\!]$. The tree t is constant since it extends a block tree of height m. Thus $[\![s]\!]$ is a constant strong tree class.

[6] In particular the condition $(v^\omega g \neq 0, v^\omega h \neq 0) \Rightarrow (v^\omega g = v^\omega h)$ (obtained in [6,7] for definite languages) is sufficient but not necessary for a tree-shift language to be of finite type.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

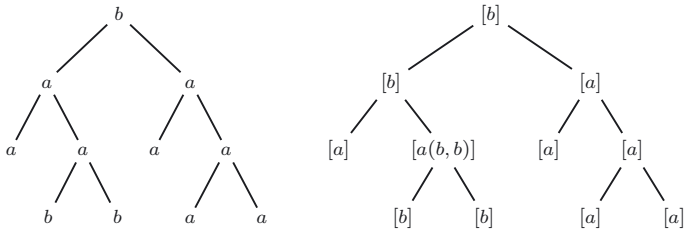


FIGURE 5. A tree-shift language which is not of finite type and the computation of its class in its syntactic tree algebra. Factor trees containing an odd number of a between two b on some top-down path going from one node to another in the tree are forbidden.

Conversely, let us assume that L is a regular tree-shift language such that any strong tree class of $\mathbb{T}^\omega$ is constant. Let m be an integer such that $\mathbb{T}^\omega = \mathbb{T}_{\geq m}$. Let s be a block tree of height $n \geq m$. Then $[\![s]\!] \in \mathbb{T}_n \subseteq \mathbb{T}^\omega$. It follows that $[\![s]\!]$ is constant and thus s is a constant tree. □

Example 1. In Figure 5 is pictured a tree of a tree-shift language on the alphabet $\{a, b\}$. The language is the set of trees containing an even number of a between two b on any top-down path in the tree. Moreover, any two top-down paths starting at nodes labelled by b and ending at a same node have the same number of a modulo 2. Hence the tree $a(a(b, b), b)$ is not allowed. This tree-shift language is not of finite type.

We have

$$T = \{[a], [b], [a(b, b)], 0\},$$
$$\mathbb{T} = \{[\![a]\!], [\![b]\!], [\![a(b, b)]\!], 0\},$$
$$\mathbb{T}^\omega = \mathbb{T}.$$

The strong tree class $[\![a]\!]$ is not a constant class. Indeed, $[\![a(b, b)]\!] \neq [\![a]\!]$. As a consequence the language L is not of finite type.

For regular tree-shift languages, the characterization obtained in Theorem 5.6 is computable in an effective way since the strong syntactic algebra, $\mathbb{T}^\omega$, $I(s)$ for a tree s and the property of being constant are computable. Nevertheless, the characterization does not to lead a polynomial-time decision. A polynomial-time algorithm for deciding the finite-type property of a regular language given by a deterministic tree automaton was obtained in [3].

6. Almost of finite type tree-shift languages

In this section, we define the notion of almost of finite type tree languages. We refer to [17] for the notion of almost of finite type shifts. These languages are

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

regular transitive tree-shift languages recognized by a tree automaton which is both deterministic and co-deterministic with a finite delay. There is no description in terms of blocks of almost of finite type languages as for finite-type languages. They correspond to a class of symbolic dynamical tree-shifts which is in between tree-shifts of finite type and sofic tree-shifts. The class of almost finite type tree-shift languages strictly contains transitive tree-shift languages of finite type and it is strictly smaller than the class of regular tree-shift languages.

A tree automaton is *irreducible* if for each pair of states p, q, there is a finite complete prefix code X of Σ^* and a finite computation c of the automaton on a tree such that $c_\varepsilon = p$ and $c_x = q$ for each $x \in X$. We say that a tree s (or a context p) is a *synchronizing* tree (or context) of a tree automaton if all computations of the tree automaton on s have the same root. A synchronizing tree of the context tree automaton is a constant tree and conversely. A tree automaton $\mathcal{A} = (V, A, \Delta)$ is *co-deterministic (or left-closing) with delay* m if any two computations of $\mathcal{A}$ on a same block tree of height $m + 1$ which have the same root, are equal at the two nodes $x = 0$ and $x = 1$.

It is proved in [2,4] that there is a unique minimal deterministic irreducible and synchronized tree automaton accepting a transitive regular tree-shift language. It is equal to the unique irreducible component of the context tree automaton, called the Fischer cover[7] of the tree language. Moreover a regular transitive tree-shift language is almost of finite type if and only if its right Fischer cover is co-deterministic with a finite delay.

Theorem 6.1. *A regular transitive tree-shift language is almost of finite type if and only if the following property holds in its strong tree algebra.*

$$\begin{cases} [\![s]\!] \leq [\![t]\!], [\![s']\!] \leq [\![t]\!] \\ [\![ps]\!] \neq 0, [\![ps']\!] \neq 0, \end{cases} \quad \Rightarrow \quad [\![s]\!] = [\![s']\!], \tag{6.1}$$

where $[\![s]\!]$, $[\![s']\!]$ *are constant strong tree classes,* $[\![p]\!]$ *is a constant strong context class of* $\mathbb{C}^\omega$*, and* $[\![t]\!]$ *is a strong tree class of* $\mathbb{T}^\omega$*.*

Proof. Let us assume that L is an almost of finite type tree language. We denote by $\mathcal{F}$ the unique irreducible component of the context tree automaton of L. There is a positive integer m such that the tree automaton $\mathcal{F}$ is co-deterministic with delay m.

Let $[\![s]\!]$, $[\![s']\!]$ be constant strong tree classes, $[\![t]\!]$ be a strong tree class of $\mathbb{T}^\omega$, $[\![p]\!]$ be a strong constant context class, with $[\![s]\!], [\![s']\!] \leq [\![t]\!]$ and $[\![ps]\!], [\![ps']\!] \neq 0$. These conditions imply $[\![pt]\!], [\![s]\!], [\![s']\!] \neq 0$. Since $[\![s]\!]$, $[\![s']\!]$ are nonnull constant classes and L is transitive, $I(s) = \{h\}$ and $I(s') = \{h'\}$, where h and h' are states of the irreducible component $\mathcal{F}$. Since $[\![p]\!] \in \mathbb{C}^\omega$, there is a context q and a tree t' of height greater than m with qt' is a block tree and $[\![p]\!] = [\![q]\!]$. We have $[\![qs]\!] = [\![ps]\!] \neq 0$ and $[\![qs']\!] \neq 0$. Since $[\![t]\!] \in \mathbb{T}^\omega$, there is a block tree u of height greater than m such that $[\![u]\!] = [\![t]\!]$. We have $h, h' \in I(u)$. Let c (resp. c') be a computation of $\mathcal{F}$ on u with

[7] also called the Shannon cover.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

root h (resp. h'). Let us assume that q has a its symbol $\Box$ at the node x. Since $[\![qs]\!] \neq 0$ (resp. $[\![qs']\!] \neq 0$), there is a computation d (resp. d') of $\mathcal{F}$ on qu such that the label of d at the node x is h (resp. h'). Since the context q is constant, these two computations d and d' have the same root. The irreducible tree automaton $\mathcal{F}$ being co-deterministic with delay m, we obtain that $h = h'$ and thus $[\![s]\!] = [\![s']\!]$.

Conversely, let us assume that implication (6.1) holds. Let m be an integer such that $\mathbb{T}^\omega = \mathbb{T}_{\geq m}$. Let $t = a(t_0, t_1)$ be a block tree of height greater than or equal to $m+1$ such that there are two computations c and c' of $\mathcal{F}$ on t ending with a same root h. The strong tree classes $[\![t_0]\!]$ and $[\![t_1]\!]$ belong to $\mathbb{T}_{\geq m} = \mathbb{T}^\omega$.

Let us assume that h_0 (resp. h'_0) is the label of c (resp. c') at the node 0 while h_1 (resp. h'_1) is the label of c (resp. c') at the node 1. Since h_0, h'_0, h_1, h'_1 are states of $\mathcal{F}$, and since $\mathcal{F}$ is irreducible, there are constant trees s_0, s'_0, s_1, s'_1 such that $[s_i] = h_i$, $[s'_i] = h'_i$, for $i = 0, 1$. We have $[\![s_0]\!] \leq [\![t_0]\!]$, $[\![s'_0]\!] \leq [\![t_0]\!]$ (resp. $[\![s_1]\!] \leq [\![t_1]\!]$, $[\![s'_1]\!] \leq [\![t_1]\!]$). By construction, the classes $[a(s_0, t_1)]$, $[a(s'_0, t_1)]$, $[a(t_0, s_1)]$, $[a(t_0, s'_1)]$ are nonnull.

Let u be a constant tree. Since $\mathcal{F}$ is irreducible, there is a computation in $\mathcal{F}$ on a tree t' with all initial states equal to h, with root $[u]$, and such that u is a subtree of t' at its root. Let t" be the the tree obtained from t' by extending all its leaves left and right with the tree t. The trees t', t" are constants. Let x be the position of one leaf of t'. Let p be the context obtained from t" by replacing the tree t_0 at position $x0$ by the box symbol. Let q be the context obtained from t" by replacing the tree t_1 at position $x1$ by the box symbol. We have $[\![p]\!]$, $[\![q]\!]$ in $\mathbb{C}^\omega$ and $[\![p]\!]$, $[\![q]\!]$ are constant contexts. We get $[\![ps_0]\!] \neq 0, [\![ps'_0]\!] \neq 0$ (resp. $[\![qs_1]\!] \neq 0, [\![qs'_1]\!] \neq 0$). By implication (6.1) used two times, we obtain $[\![s_0]\!] = [\![s'_0]\!]$ and $[\![s_1]\!] = [\![s'_1]\!]$. Hence $h_0 = h'_0$ and $h_1 = h'_1$, showing that $\mathcal{F}$ is co-deterministic with delay m and L is almost of finite type. □

Example 2. We want to check property (6.1) for the language L of Example 1. We have

$$\mathbb{T} = \mathbb{T}^\omega = \{[\![a]\!], [\![b]\!], [\![a(b,b)]\!]\}.$$

The nonnull constant strong tree classes are $[\![b]\!]$, $[\![a(b,b)]\!]$ with $[\![b]\!] \leq [\![a]\!]$, $[\![a(b,b)]\!] \leq [\![a]\!]$.

The nonnull constant strong context classes are $[\![b(\Box, b)]\!]$ and $[\![a(\Box, b)]\!]$. Since there is nothing to check in property (6.1) when $[\![t]\!]$ is a constant strong tree class, we only consider the case $[\![t]\!] = [\![a]\!]$. With $[\![p]\!] = [\![b(\Box, b)]\!]$, we have $[\![pb]\!] \neq 0$ but but $[\![pa(b,b)]\!] = 0$. Thus implication (6.1) is true. With $[\![p]\!] = [\![a(\Box, b)]\!]$, we have $[\![pb]\!] \neq 0$ but $[\![pa(b,b)]\!] = 0$. As a consequence, the language is almost of finite type.

A deterministic and co-deterministic tree automaton with delay 1 accepting L is described in Figure 6. The two states q_0 and q_1 control the parity of the number of a encountered from any last b below.

For regular tree-shift languages, the characterization obtained in Theorem 6.1 is computable in an effective way since the strong syntactic algebra is computable. Nevertheless, the characterization does not lead to a polynomial-time decision.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

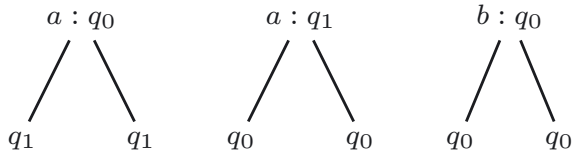


FIGURE 6. The transitions of a deterministic and co-deterministic with delay 1 tree automaton which accepts the trees whose paths have an odd number of a between two b.

A polynomial-time algorithm for deciding the almost of finite-type property of a regular language given by a deterministic tree automaton was obtained in [4].

7. Strong tree algebra of order two

In this final section, we show that an even stronger tree algebra, called the strong tree algebra of order two, can be defined for tree-shift languages. A new equivalence, denoted $\approx_2$, refines the strong equivalence as follows. In this section, we give an example of two trees which are strong equivalent and are not equivalent in the strong tree algebra of order two. This shows that the refinement is strict.

Let s, t be a trees, we say that $s \leq_2 t$ when the set of strong equivalence classes of s' for $s \prec s'$ is included into the set of strong equivalence classes of t' for $t \prec t'$.

Two trees s, t are called *equivalent in the strong tree algebra of order two*, written $s \approx_2 t$, if $s \leq_2 t$ and $t \leq_2 s$. We denote by $[\![s]\!]_2$ the class of s for $\approx_2$. We write $[\![s]\!]_2 \leq [\![t]\!]_2$ when $s \leq_2 t$. This property is independent of the choice of the class representative. For a tree-shift language L, we have

$$[\![s]\!]_2 \leq [\![t]\!]_2 \Rightarrow [\![s]\!] \leq [\![t]\!] \Rightarrow [s] \leq [t].$$

Indeed, the second implication was shown in Section 4. The first one comes from the following argument. If $[\![s]\!]_2 \leq [\![t]\!]_2$, then

$$\{\{\mathrm{cont}(s") \mid s' \prec s"\} \mid s \prec s'\} \subseteq \{\{\mathrm{cont}(t") \mid t' \prec t"\} \mid t \prec t'\}.$$

Since L is a tree-shift language, we have $\mathrm{cont}(s') = \bigcup_{s' \prec s"} \mathrm{cont}(s")$. Hence, for any tree s' with $s \prec s'$, there is a tree t' with $t \prec t'$ and $\mathrm{cont}(s') = \mathrm{cont}(t')$, or, equivalently, $[\![s]\!] \leq [\![t]\!]$.

Thus $s \approx_2 t$ implies $s \approx t$ but the converse is not true as is shown in Example 3 below. The equivalence $\approx_2$ is thus strictly stronger than $\approx$.

Two contexts p, q are called *equivalent in the strong tree algebra of order two*, written $p \approx_2 q$, if and only if, for any tree t, the two trees qt and $q't$ are equivalent for the strong equivalence of order two. Properties similar to the ones of equations (4.1) to (4.6) are satisfied.

Corollary 7.1. *The strong tree algebra of order two of a tree-shift language has finite index if and only if the language is regular.*

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

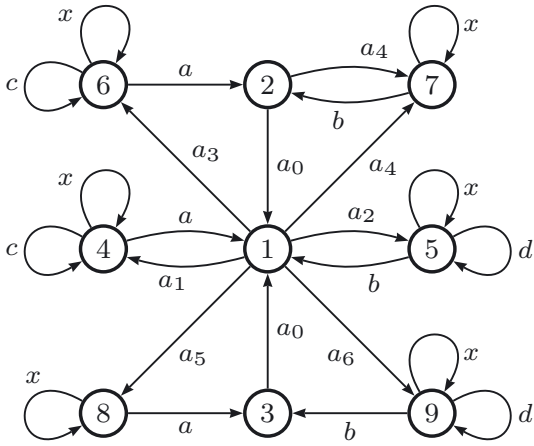


FIGURE 7. The automaton $\mathcal{A}$ on the alphabet $A = \{a, b, a_0, \ldots, a_6\}$.

Proof. The strong equivalence or order 2 is stronger that the equivalence. Thus it has an infinite number of classes when the language L is not regular. Conversely, if L is regular, the number of states of the determinized context tree automaton is finite. For any tree s, let $P(s)$ be the set of subsets of $I(s)$ equal to $I(t)$ for some tree t with $s \prec t$. We have $[\![s]\!]_2 = [\![t]\!]_2$ if and only if $P(s) = P(t)$. As a consequence, the number of strong tree classes is bounded above by the number of subsets of states of the determinized context tree automaton. □

Example 3. We consider the (path-testable) tree language where any of branch of a tree (from the bottom to the top) is the label of path in the finite word automaton of Figure 7. We have

$$\begin{aligned}
[\![a]\!] &= \{[a] = [a(c,c)], [a(a_1,a_1)], [a(a_3,a_3)], [a(a_5,a_5)]\},\\
[\![b]\!] &= \{[b] = [b(d,d)] = [a], [b(a_2,a_2)] = [a(a_1,a_1)],\\
[b(a_4,a_4)] &= [a(a_3,a_3)], [b(a_6,a_6)] = [a(a_5,a_5)]\}\\
&= [\![a]\!],
\end{aligned}$$

and

$$\begin{aligned}
[\![a(c,c)]\!] &= \{[a(c,c)] = [a], [a(c,c(a_3,a_3))] = [a(a_3,a_3)],\\
[a(c,c(a_1,a_1))] &= [a(a_1,a_1)]\},\\
[\![b(d,d)]\!] &= \{[b(d,d)] = [a], [d(b,b(a_2,a_2))] = [a(a_1,a_1)],\\
[d(b,b(a_6,a_6))] &= [a(a_5,a_5)]\}\\
&\neq [\![a(c,c)]\!].
\end{aligned}$$

Furthermore, one can check that there is no tree s with $b \prec s$ such that $[\![s]\!] = [\![a(c,c)]\!]$. As a consequence, we have $[\![a]\!] = [\![b]\!]$ but $[\![a]\!]_2 \neq [\![b]\!]_2$.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

This example suggests a strict infinite hierarchy of strong tree algebras extending the strong tree algebra of order 2. In this paper, we have considered algebras with contexts of arity 1 (*i.e.* with one box). Algebras with multicontexts may also be investigated (see [6]). Forest algebras (see [6]) may not be well suited for tree-shift languages since the arity of these trees is fixed.

Acknowledgements. The authors would like to thank anonymous reviewers for very helpful comments.

References

[1] N. Aubrun and M.-P. Béal, Decidability of conjugacy of tree-shifts of finite type. In *ICALP '09: Proc. of the 36th International Colloquium on Automata, Languages and Programming*. Springer-Verlag Berlin, Heidelberg (2009) 132–143.
[2] N. Aubrun and M.-P. Béal, Sofic and almost of finite type tree-shifts. In *5th International Computer Science Symposium in Russia, (CSR'10)*, edited by E. Mayr and F. Ablayev, number 6072 in *Lect. Notes Comput. Sci.* Springer-Verlag (2010) 12–24.
[3] N. Aubrun and M.-P. Béal, Tree-shifts of finite type. *Theoret. Comput. Sci.* **459** (2012) 16–25.
[4] N. Aubrun and M.-P. Béal, Sofic tree-shifts. *Theory Comput. Syst.* **53** (2013) 621–644.
[5] M.-P. Béal, F. Fiorenzi and D. Perrin, A hierarchy of shift equivalent sofic shifts. *Theoret. Comput. Sci.* **345** (2005) 190–205.
[6] M. Bojanczyk, Algebra for Trees. In *Handbook of Automata Theory*. To appear in EMS Publishing.
[7] M. Bojanczyk, Effective characterizations of tree logics. Tutorial at PODS 2008 (2008).
[8] M. Bojanczyk, L. Segoufin and H. Straubing, Piecewise testable tree languages. In *LICS* (2008) 442–451.
[9] M. Boyle, B. Kitchens and B. Marcus, A note on minimal covers for sofic systems. *Proc. Amer. Math. Soc.* **95** (1985) 403–411.
[10] T. Ceccherini–Silberstein, M. Coornaert, F. Fiorenzi and Z. Sunic, Cellular automata on regular rooted trees. In *CIAA 2012* (2012) 101–112.
[11] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi, Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, release October, 12th (2007).
[12] A. de Luca and A. Restivo, A characterization of strictly locally testable languages and its applications to subsemigroups of a free semigroup. *Inform. Control* **44** (1980) 300–319.
[13] G. Fici and F. Fiorenzi, Topological properties of cellular automata on trees. In *DCM* (2012) 255–266.
[14] U. Heuter, Definite tree languages. *Bulletin of the EATCS* **35** (1988) 137–142.
[15] U. Heuter, Generalized definite tree languages. In Mathematical Foundations of Computer Science 1989 (Porąbka-Kozubnik, 1989), vol. 379 of *Lect. Notes Comput. Sci.* Springer, Berlin (1989) 270–280.
[16] E. Jeandel and G. Theyssier, Subshifts, languages and logic. In *Developments in Language Theory, 13th International Conference, DLT 2009, Stuttgart, Germany, June 30 - July 3, 2009. Proceedings*, vol. 5583 of *Lect. Notes Comput. Sci.* Springer (2009).
[17] D. Lind and B. Marcus, *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, Cambridge (1995).
[18] M. Nivat and A. Podelski, Definite tree languages. *Bulletin of the EATCS* **38** (1989) 186–190.
[19] T. Place and L. Segoufin, A decidable characterization of locally testable tree languages. In *ICALP (2)*, *Lect. Notes Comput. Sci.* Springer (2009) 285–296.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press

[20] S. Salehi, A completeness property of wilke's tree algebras. In *MFCS*, vol. 2747 of *Lect. Notes Comput. Sci.* Springer (2003) 662–670.
[21] J. Verdú-Mas, R. Carrasco and J. Calera–Rubio, Parsing with probabilistic strictly locally testable tree languages. *IEEE Trans. Pattern Anal. Mach. Intell.* **27** (2005) 1040–1050.
[22] T. Wilke, An algebraic characterization of frontier testable tree languages. *Theoret. Comput. Sci.* **154** (1996) 85–106.

Communicated by Ch. Choffrut.
Received March 14, 2014. Accepted March 17, 2014.

https://doi.org/10.1051/ita/2014018 Published online by Cambridge University Press