Multiple tree automata: an alternative tree-oriented formalism for LCFRS

Gwendal Collet and Julien David

[1] LIX UMR 7161, École Polytechnique, 91128 Palaiseau cedex, France
[2] LIPN UMR 7030, Université Paris 13—CNRS, 99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, France **

Abstract. This paper introduces the Multiple Tree Automata that recognize languages of ordered trees. The authors originally created this model in order to solve a problem of automatic random sampling of ordered trees. This model is weakly equivalent to Linear Context Free Rewriting Systems, though it emphasizes the top-down aspect and the recognition of tree languages. We introduce a minimization algorithm for deterministic automata. We also propose a graphical representation for our automata which could be used for regular tree automata.

1 Introduction

In the literature, tree languages appear in two different contexts: either as the main object of study for program schemes, search in files and can typically dealt with Regular tree automata [2], Pushdown tree automata [4], either as derivation trees for word languages in language theory and come from Context-free grammars, Tree-adjoining grammars [5], Linear Context-Free Rewriting Systems [11, 7, 12, 9] (referred to as **LCFRS**)...

In [1], the authors studied the random sampling of ordered trees containing a given pattern. They gave an algorithm which produces a grammar recognizing trees containing the pattern. The grammar, in order to be used by random sampling methods, could not decompose the pattern branch by branch, as it is done, for instance, in [3]. As a matter of fact, the authors had to come up with a new model of tree grammar (which is presented in this paper) adapted to their problem. This may be seen as a generalization of a top-down regular tree grammar.

This model appears to be weakly equivalent to **LCFRS**, as their yields have the same expressive power, which suggests it can be used in natural language processing. The tree languages of both models are incomparable, as Multiple tree automata allow to describe trees whose internal nodes can be labelled, and **LCFRS** are more powerful than Multiple Tree Automata in the unlabelled case. We note that when using **LCFRS** in natural language processing, it is a common practice to compensate the absence of labels on internal nodes by using the labels

** We would especially like to thanks Joseph Le Roux for his precious comments on this work.

of the non-terminal from which they are derivated. This is sufficient to obtain a syntactic tree of a sentence.

Automata	Tree languages	Yields
Pushdown Tree Automata	Context-free tree languages	Indexed languages
Multiple Tree Automata	—	Linear CF rewriting languages
Regular Tree Automata	Regular tree languages	Context-free languages

Fig. 1. Proper hierarchy between models of tree automata.

The paper is organized as follows. Section 2 introduces notations and definitions. In Section 3, we present the Multiple Tree Automata. Section 4 contains a pumping lemma. Section 5 describes a minimization algorithm for Deterministic Top-down Multiple Tree Automata. In Section 7, we discuss the closure properties of languages recognized by Multiple Tree Automata. We state some decidability results in Section 8. Section 9 is a small discussion on the possible applications of Multiple Tree Automata in natural language processing.

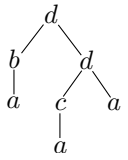
2 Definitions

Definition 1. *A* **ranked alphabet** *is a couple* $(\Sigma, arity)$ *where* Σ *is a finite alphabet* $\{a_1, \ldots, a_k\}$ *of* k *symbols and* $arity : \Sigma \to \mathbb{N}$. *Any symbol* $a \in \Sigma$ *such that* $arity(a) = 0$ *(resp.* $arity(a) > 0$*) is a* **leaf** *(resp. an* **internal node***). A tree-alphabet is a ranked alphabet with at least one leaf.*

Definition 2. *A tree* t *over a tree-alphabet* Σ *is defined inductively as follows:*

- $\forall a \in \Sigma$ *a leaf,* a *is a tree,*
- $\forall b \in \Sigma$ *an internal node,* $\forall t_1, \ldots, t_{arity(b)}$ *trees,* $b(t_1, \ldots, t_{arity(b)})$ *is a tree.*

Example 1. Let $\Sigma = \{a, b, c, d\}$ be a tree-alphabet such that $arity(a) = 0$, $arity(b) = arity(c) = 1$ and $arity(d) = 2$. $d(b(a), d(c(a), a)))$ is a tree over Σ, with its graphical representation on the right.



We define the height of a tree inductively by:

$$height(t) = \begin{cases} 0 \text{ if } t \text{ is reduced to a leaf,} \\ 1 + max\{height(t_i) \mid t = b(t_1, \ldots, t_{arity(b)})\} \text{ otherwise.} \end{cases}$$

$\mathfrak{P}$ denotes the combinatorial class of set partitions. Let $\vec{n} = (n_1, \ldots, n_k)$ be a tuple. We have: $\mid \vec{n} \mid = k$ and $\mid\mid \vec{n} \mid\mid = \sum_{i=1}^{k} n_i$. Also, for all $i \in \{1, \ldots, k\}$ we write $\mid\mid \vec{n} \mid\mid_i = \sum_{j=1}^{i} n_j$. Let $\vec{a} = (a_1, \ldots, a_k)$ be a k-tuple and $P = \{p_1, \ldots, p_l\}$ be a subset of $\{1, \ldots, k\}$. We define : $\vec{a}_{|P} = (a_{p_1}, \ldots, a_{p_l})$.

A k-forest is a k-tuple of trees. Let $\mathcal{F}$ be the set of forests and $\mathcal{F}_k$ the subset of k-forests. A tree language is a set of trees. The *yield* of a tree language L is the set of the words built by reading all the leaves from left to right in a tree of L. It can also be defined as the image of L by the morphism *yield*:

$$yield(t) = \begin{cases} t \text{ if } t \text{ is reduced to a leaf,} \\ yield(t_1) \cdot \ldots \cdot yield(t_{arity(b)}) \text{ if } t = b(t_1, \ldots, t_{arity(b)}) \end{cases}$$

Definition 3 (Context). *Let $F \in \mathcal{F}_k$ and $(n_1, \ldots, n_l)$ a tuple of nodes, called* **buds***, such that each n_i is a distinguished leaf of a tree in F. The pair $C = \{F, (n_1, \ldots, n_l)\}$ is called a context[3]. The empty tuple of buds is denoted by ε and we consider that a forest is a context with an empty tuple of buds, that is $F = \{F, \varepsilon\}$.*

Definition 4. *Let $C_1 = \{F_1, (n_1, \ldots, n_l)\}$ and $C_2 = \{F_2, (n'_1, \ldots, n'_{l'})\}$ be two contexts where $F_2 \in \mathcal{F}_l$. We define $C_1[C_2]$ as the context $\{F_3, (n'_1, \ldots, n'_{l'})\}$ where F_3 is obtained by replacing each n_i in F_1 by the tree t_i in F_2. If $F_1 \in \mathcal{F}_l$, we define $C_1^n (n \geq 1)$ the following way: $C_1^n = \begin{cases} C_1, \text{ if } n = 1 \\ C_1[C_1^{n-1}], \text{otherwise} \end{cases}$*

3 Model Presentation

The Multiple Tree Automata we present in this section is a generalization of Regular Tree Automata. The main idea is that the transitions of the automaton can handle a bounded number of nodes of same depth at once, instead of having independency between siblings.

Although this model can handle labelled trees, in the following most examples will feature unlabelled trees, *i.e.* tree-alphabet with at most one symbol for each arity.

3.1 Multiple Tree Automata

In this section, we present the Multiple Tree Automata, which can equivalently be seen as top-down grammars (building the tree from the root to the leaves). A Multiple Tree Automaton is a tuple $\mathcal{A} =< Q, rank, I, \Sigma, T >$ such that:

- $(Q, rank)$ is a finite ranked set of states,
- $I \subseteq Q$ is the set of initial states, such that for all $i \in I$, $rank(i) = 1$.
- Σ is a tree-alphabet.
- $T \subseteq Q \times \Sigma^{\mathbb{N}} \times \mathfrak{P} \times Q^{\mathbb{N}}$ is a finite set of transitions. For any transition $t \in T$, $t = (n, (a_1, \ldots, a_{rank(n)}), \mathcal{P}, \overrightarrow{o})$, we have : $\mathcal{P} = \{P_1, \ldots, P_k\}$ is a partition of the set $\{1, \ldots, \sum_{i=1}^{rank(n)} arity(a_i)\}$ into k parts, where $\mid \overrightarrow{o} \mid = k$ and for all $1 \leq j \leq k$ we have $rank(o_j) = |P_j|$. Figure 2 shows an example of transition.

  The size of such an automaton is the sum of the size of each transition: $|\mathcal{A}| = \sum_{t \in T} |start(t)|$.

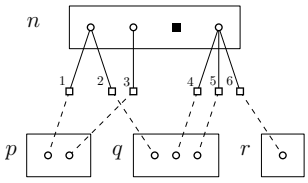


Fig. 2. In this example, the tree-alphabet is $\Sigma = \{\blacksquare, \mathbin{|}, \curlywedge, \curlywedge\!\mathbin{|}\}$. The automaton contains at least four states: n, p, q, r where $rank(n) = 4$, $rank(p) = 2$, $rank(q) = 3$, $rank(r) = 1$. The transition described in the example is encoded by $(n, (\curlywedge, \mathbin{|}, \blacksquare, \curlywedge\!\mathbin{|}), \{\{1,3\},\{2,4,5\},\{6\}\}, (p,q,r))$

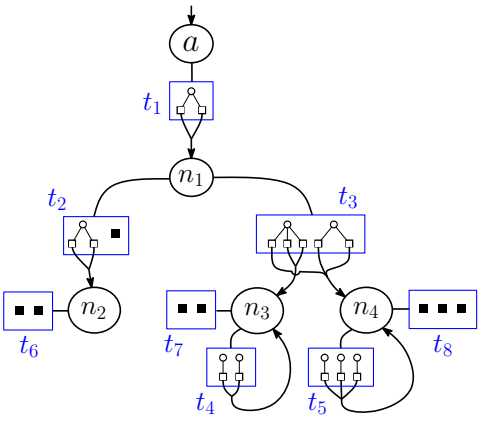
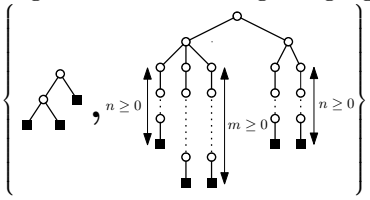


Fig. 3. The automaton on the left recognizes the following language:



or $\{\curlywedge\blacksquare\blacksquare, \curlywedge\!\mathbin{|}^n\blacksquare\mathbin{|}^m\blacksquare\mathbin{|}^m\blacksquare\curlywedge\!\mathbin{|}^n\blacksquare\mathbin{|}^n\blacksquare\}$. The arrows indicates that the unary node $\mathbin{|}$ can be repeated respectively n times or m times, with $n \geq 0$ and $m \geq 0$. We have $rank(a) = 1$, $rank(n_1) = 2$, $rank(n_2) = 1$, $rank(n_3) = 2$, $rank(n_4) = 3$.

Let $p \in Q$ be a state in a Multiple Tree Automaton. We call p_i, for $i \in \{1, \ldots, rank(p)\}$, a **seed** of p. Intuitively, one can see a state of rank k as a k-tuple of seeds on which the transitions act.

Let $t = (p, \vec{a}, \mathcal{P}, \vec{q})$ be a transition in a Multiple Tree Automaton. We write $start(t) = p$, $label(t) = \vec{a}$, $part(t) = \mathcal{P}$ and $end(t) = \vec{q}$. Let p be a state in a Multiple Tree Automaton. We write

$$label(p) = \bigcup_{\substack{t \in T \\ start(t)=p}} \{label(t)\} \text{ and } end(p) = \bigcup_{\substack{t \in T \\ start(t)=p}} \{end(t)\}$$

Let p_i be a seed of p, with $p = start(t)$. We write $endSeeds(t, i)$ the tuple of states seeds from $end(t)$ associated to p_i by t. For instance, in Figure 2, which can equivalently be considered as a transition in an automaton or a rule in a grammar, we have $rank(N_1) = 4$ and $endSeeds(t, 1) = p_1 q_1$, $endSeeds(t, 2) = p_2$, $endSeeds(t, 3) = \varepsilon$[4], $endSeeds(t, 4) = q_2 q_3 r_1$.

A **course** of a Multiple Tree Automaton is a tree in which nodes are transitions of the Multiple Tree Automaton and for each node t, the node's children

[3] This notion will be used in Section 4.

[4] ε denotes the empty tuple

are a tuple $\vec{t'}$ of transitions such that $|end(t)| = |\vec{t'}|$ and for all $i \leq |\vec{t'}|$, $end(t)_i = start(t'_i)$.

Let $F = (a_1(T_{1,1}, \ldots, T_{1,arity(a_1)}), \ldots, a_k(T_{k,1}, \ldots, T_{k,arity(a_k)}))$ be a k-forest. A context $(F, \vec{l})$ **labels** a course ρ if:
let t be the root of ρ, we have:

- $rank(start(t)) = k$,
- $label(t) = (a_1, \ldots, a_k)$
- $part(t)$ is a partition of $\{1, \ldots, \sum_i arity(a_i)\}$ into $|end(t)|$ parts,
- let $F' = (T_{1,1}, \ldots, T_{k,arity(a_k)})$: for $i \leq |end(t)|$, the context $(F'_{|part(t)_i}, \vec{l})$ labels the sub-course of ρ enrooted in the i-th child of t.

Figure 4 shows an example of a course labelled by a context and a course labelled by a tree.

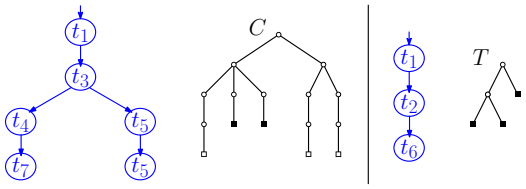


Fig. 4. The example on the left (resp. on the right) is a course labelled by a context C (resp. a tree T). The values $t_1, \ldots, t_7$ denote the transitions of the automaton in Figure 3. The black squares in C and T denote leaves, whereas white square denote buds.

The **right-language** of a state p, or simply the language of a state p, is the forest that labels a course enrooted in p. The tree language recognized by a Multiple Tree Automaton is then the union of right languages of the initial states.

A Multiple Tree Automaton $< Q, rank, I, \Sigma, T >$ is **deterministic** if:

- given a value k, there is at most one initial i,
- for all state $q \in Q$ and all tuple $\vec{a}$ of symbols in $\Sigma^{rank(q)}$ there is **at most** one pair $\{P, \vec{q}\}$ such that $(q, \vec{a}, P, \vec{q}) \in T$.

In Section 7, we show that some languages recognized by a Multiple Tree Automaton cannot be recognized by a deterministic one.

Remark 1 (Bottom-up). One could also define a Bottom-up analogue of these automata, reading a tree starting from its leaves, better suited for parsing purposes. Similarly to Regular Tree automata, this can be done be reversing transitions. In the non-deterministic case, the expressive power of Bottom-up and Top-down MTA are the same. This is no longer true in the deterministic case, much like Regular Tree Automata.

A Multiple Tree Automaton is **accessible** if each of its states is accessible. A state q is accessible if it is an initial state or if there exists a course c enrooted in an initial state such that $q \in end(t)$, with t a node of c.

A Multiple Tree Automaton is **co-accessible** if each of its states is co-accessible. A state q is co-accessible if its right language is not empty.

A Multiple Tree Automaton is **trimmed** if it is both accessible and co-accessible.

Lemma 1. *A Multiple Tree Automaton can be trimmed in linear time and space.*

Proof. Computing the accessible part of the automaton can be done with exactly the same method as for finite word automata and is linear in the number of transitions. To compute the co-accessible part, the idea is to keep a data structure which indicates for each state p, the list of transitions t and the list of positions i such that $end(t)_i = p$. One starts by marking each state which can read a tuple of leaves. Then we use the list mentioned before to mark the transitions t such that the states $end(t)$ are all marked (each transition is associated to a counter, therefore such test can be made in constant time). When such transition is marked, $start(t)$ is also marked, and so on. At the end, unmarked states and transitions are deleted from the automaton.

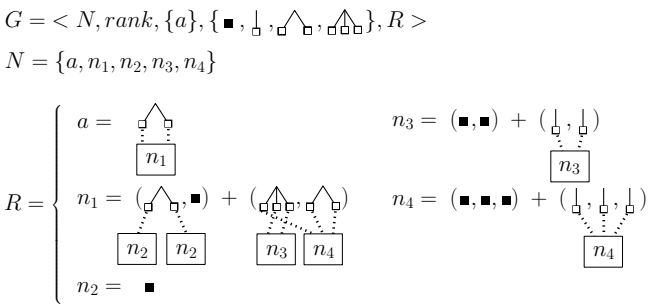


Fig. 5. A Grammar representation of the automaton in Figure 3.

Remark 2 (Multiple tree grammar). One could rather see our automata as grammars. Figure 5 shows a grammar representation of the automaton in Figure 3. Since the formal definition is fairly similar, we skip it.

4 Pumping Lemma

In the following, the height of a k-forest is the maximal height of its trees. The lemma is illustrated by Figure 6.

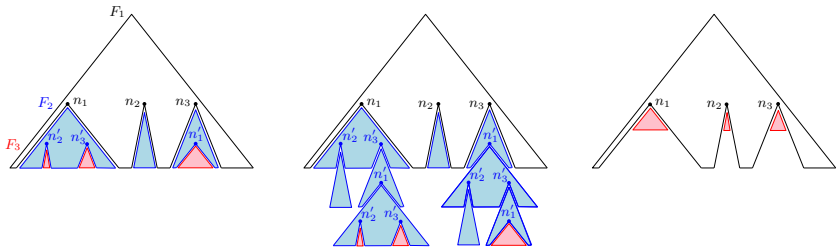


Fig. 6. F_1, F_2 and F_3 respectively are the forest in white, blue and red. If we define $C_1 = \{F_1, (n_1, n_2, n_3)\}$, $C_2 = \{F_2, (n'_1, n'_2, n'_3)\}$ and $C_3 = \{F_3, \emptyset\}$, then we can see from left to right $C_1[C_2[C_3]]$, $C_1[C_2^3[C_3]]$ and $C_1[C_3]$.

Lemma 2 (Pumping Lemma). *If $\mathcal{L}$ is a tree language recognized by a Multiple Tree Automaton, there exists a value h such that for all trees $F \in \mathcal{L}$ such that $height(F) \geq h$ then:*

- *there exist three forests $F_1 \in \mathcal{F}_1$ and $F_2, F_3 \in \mathcal{F}_l$, with F_2 non reduced to leaves,*
- *there exists a tuple of nodes $(n_1, \ldots, n_l)$ with same depth in F_1 and we write $C_1 = \{F_1, (n_1, \ldots, n_l)\}$,*
- *there exists a tuple of nodes $(n'_1, \ldots, n'_l)$ with same depth in F_2 and we write $C_2 = \{F_2, (n_1, \ldots, n_l)\}$ and $C_3 = \{F_3, \emptyset\}$,*
- *such that $F = C_1[C_2[C_3]]$.*

For all $m \geq 1$, the trees $C_1[C_2^m[C_3]]$ are in $\mathcal{L}$ and so is $C_1[C_3]$.[5]

The following lemma shows an example of application of the Pumping Lemma and of a language that is not recognized by a Multiple Tree Automaton.

Lemma 3. *The language of complete binary trees cannot be recognized by a finite Multiple Tree Automaton.*

Proof. Suppose that such an automaton exists. Then let h be greater than the number of transitions in the automaton. The complete binary tree T of height h is recognized. For all contexts $C_1 = \{F_1, (n_1, \ldots, n_l)\}$, $C_2 = \{F_2, (n'_1, \ldots, n'_l)\}$, C_3 such that $T = C_1[C_2[C_3]]$, the tree $C_1[C_3]$ is not a complete binary tree. We conclude using the Pumping Lemma.

5 Minimization Algorithm

One of the key feature of Multiple Tree Automata, in contrast to Regular Tree Automata, is that a state might have a rank greater than 1. If this generalization might prove useful to describe some languages, there might be some cases in

[5] Note for the reviewer: a proof can be found in the annex.

which a state can be replaced by several states of lower ranks, without modifying the language recognized by the automaton. We say that we "*split* a state" in such case. To split states in a Multiple Tree Automaton is necessary to build a minimization algorithm.

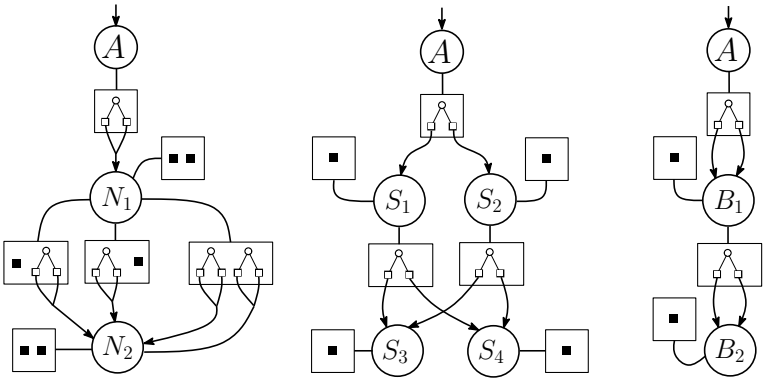


Fig. 7. Multiple Tree Automata on this figure are equivalent. The states N_1 and N_2, both of rank 2, can respectively be splited into $\{S_1, S_2\}$ and $\{S_3, S_4\}$ which are all of rank 1. Then, S_1 and S_2 (resp. S_3 and S_4) can be merged into B_1 (resp. B_2). The right-most automaton is minimal.

Let $\mathcal{A} =< Q, rank, I, \Sigma, T >$ be a deterministic trimmed Multiple Tree Automaton. Let p and q be two states of $\mathcal{A}$. Without loss of generality, we consider that $rank(p) \leq rank(q)$. Let p_i (resp. q_j) be a seed of p (resp. q) with $i \in \{1, \ldots, rank(p)\}$ (resp. $j \in \{1, \ldots, rank(q)\}$). The two seeds p_i and q_j are equivalent, denoted by $p_i \sim_s q_j$, if :

- there exists an injection $inj : \{1, \ldots, rank(p)\} \to \{1, \ldots, rank(q)\}$ from the seeds of p to the seeds of q such that $inj(i) = j$.
- for all transitions t such that $start(t) = p$, there exists at least one transition t' such that $start(t') = q$ and which satisfies the following property:
  - for all $k \in \{1, \ldots, rank(p)\}$, $label(t)_k = label(t')_{inj(k)}$.

  We call it the *matching* property.
- for all transitions t such that $start(t) = p$, and for all transitions t' satisfying the *matching* property with the same injection, for all $k \leq |endSeeds(t, i)|$, we have $endSeeds(t, i)_k \sim_s endSeeds(t', j)_k$.

To split a state p according to a partition $\mathcal{B} = \{B_1, \ldots, B_l\}$ of the set $\{1, \ldots, rank(p)\}$ into l parts is the action of replacing the state p by l states $B_i(p)$, where the seeds of each state $B_i(p)$ correspond to the seeds indexed by B_i in p. To split a state p implies to split each of its ingoing and outgoing transitions. Doing so without modifying the language of the automata signifies that two seeds p_i and p_j which are not in the same state after the split are *independant*. Two seeds of p are dependant if for all transitions t with $start(t) = p$, $label(t)_i$ can have a given value iff $label(t)_j$ also have a given value. In order to

test the indepedancy of seeds according to a given split, we define the function mix. Let A be a set of k-tuples and $\mathcal{B}$ be a partition of $\{1, \ldots, k\}$. We have $mix(A, \mathcal{B}) = \{ \vec{b} \mid \forall i, \exists \vec{a} \in A, \vec{b}_{|B_i} = \vec{a}_{|B_i} \}$.

Example 2. For instance, define $A = \{(a, b, c), (d, e, f)\}$ and $\mathcal{B} = \{\{1, 3\}, \{2\}\}$. Then we have $mix(A, \mathcal{B}) = \{(a, b, c), (d, b, f), (a, e, c), (d, e, f)\}$.

A state p is *splitable* according to a partition $\mathcal{B}$ of the set $\{1, \ldots, rank(p)\}$ if:

- $label(p) = mix(label(p), \mathcal{B})$,
- for all part $B_i \in \mathcal{B}$, for all transitions t, t' such that $start(t) = start(t') = p$, $label(t')_{|B_i} = label(t)_{|B_i}$, then for all $j \in B_i$ and all ℓ, we have $endSeeds(t, j)_\ell \sim_s endSeeds(t', j)_\ell$.
- For all transitions t such that $start(t) = p$ and $label(t) = \vec{a}$, $\mathcal{B}$ must be compatible with t, meaning we have the following property. For all $i, i' \in \{1, \ldots, rank(p)\}$ such that there exist $j \in endSeeds(t, i)$ and $j' \in endSeeds(t, i')$ where j and j' belong to the same part in $part(t)$, then i and i' belong to the same part in $\mathcal{B}$.

Set partitions form a lattice. We can compute the most refined partition which can split a state using this information.

Two states p and q are equivalent, denoted by $p \sim q$ if:

- $rank(p) = rank(q)$,
- for all $i \in \{1, \ldots, rank(p)\}$, $p_i \sim_s q_i$.

A deterministic trimmed Multiple Tree Automaton is minimal if it does not contain a splitable state or a pair of equivalent states.

In the finite automata theory, a minimal automaton is the complete deterministic automaton with fewest states recognizing a given language. While in Multiple Tree Automata, the number of states is an insufficient size measure for an automaton. Indeed, an automaton whose states have high ranks might take more space than automata with more states but smaller ranks. Furthermore one can notice that splitting a state strictly reduces the size of the automaton.

Theorem 1. *There is a unique, up to labelling, minimal Multiple Tree Automaton associated to each language. It is the* smallest *deterministic Multiple Tree Automaton recognizing a given language.*

Theorem 2. *The minimization algorithm is polynomial for Multiple Tree Automaton if the rank of each state is bounded by a constant.*

6 Comparison with other formalisms

In this section, we compare the languages recognized by Multiple Tree Automata with other formalisms. Though, in order to make a meaningful analysis, we distinguish two kinds of recognized languages:

Algorithm 1: Minimization Algorithm

Data: A Multiple Tree Automaton $\mathcal{A}$
$Min \leftarrow \mathcal{A}$;
repeat
  $\mathcal{A} \leftarrow Min$;
  $\sim_s \leftarrow$ Compute equivalence on seeds of $\mathcal{A}$;
  $SP \leftarrow$ Split states of $\mathcal{A}$ according $\sim_s$;
  $Min \leftarrow$ Minimize SP according to $\sim_s$;
until $Min = \mathcal{A}$;
return Min;

- Tree languages, *i.e.* the set of trees accepted by a Multiple Tree Automaton.
- Yields: the collection of words read along the leaves of the tree languages. The yields are the ones interesting for natural language processing. If two formalisms are equivalent at yield level, they are said to be **weakly equivalent**. For instance, **LCFRS** with fan-out 2, Linear Indexed Grammars, Combinatory Categorial Grammars, Tree-adjoining Grammars and Head Grammars have been proven to be weakly equivalent in [10]. *A fortiori,* if two formalisms can express the same tree languages, then they are also weakly equivalent.

Lemma 4 (Yields of Multiple Tree Automata). *Multiple Tree Automata are weakly equivalent to LCFRS.*

The idea of the proof is that Multiple Tree Automata are weakly equivalent to Simple Ordered LCFRS, which are themselves weakly equivalent to LCFRS.

7 Closure Properties

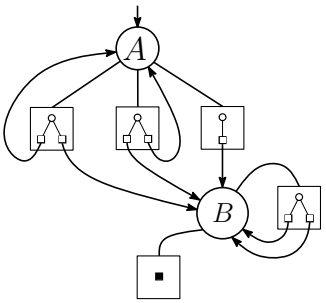


Fig. 8. A non-deterministic Multiple Tree Automaton over $\{\blacksquare, \overset{|}{\blacksquare}, \overset{\diagup\diagdown}{\blacksquare}\}$ which recognizes the language of trees with exactly one unary node. This automaton cannot be determinized, as it would require to handle dependencies between an unbounded number of siblings.

Theorem 3. *The non-deterministic automata are strictly more powerful than deterministic automata.*

Proof. Consider the language of unary-binary trees with exactly one unary node, recognized by the non-deterministic MTA in Figure 8.

Lemma 5. *The family of languages recognized by Multiple Tree Automata is closed under union and concatenation, but not under complementation.*

Proof. The construction of the automaton is almost straightforward for union and concatenation. MTA are not closed under complementation as one can construct a MTA for the complement of the language of complete binary trees (not recognizable, as proven in Lemma 3).

Conjecture 1. The family of languages recognized by Multiple Tree Automata is closed under intersection.

The intersection of two automata $\mathcal{A}_1 =< Q_1, rank_1, I_1, F_1, \Sigma, T_1 >$ and $\mathcal{A}_2 =< Q_2, rank_2, I_2, F_2, \Sigma, T_2 >$ can be computed by creating an automaton $\mathcal{A}_3 =< Q_3, rank_3, I_3, F_3, \Sigma, T_3 >$ such that each state q_3 is a couple $((\vec{q}_1, \mathcal{P}_1), (\vec{q}_2, \mathcal{P}_2))$, where $\vec{q}_1$ (resp $\vec{q}_2$) is a tuple of states from Q_1 (resp. Q_2) and $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) is a partition of the seeds in $\vec{q}_1$ (resp. $\vec{q}_2$). A transition $t_3 \in T_3$, such that $start(t_3) = q_3$ has the following property: let $label(t_3) = w$, we have:

$$\forall P_i \in \mathcal{P}_1, \exists t_1 \in T_1 \text{ s.t. } start(t_1) = q_{1,i},\ label(t_1) = w_{P_{i,1}} \cdots w_{P_{i,|P_i|}}.$$

$$\forall P_i \in \mathcal{P}_2, \exists t_2 \in T_2 \text{ s.t. } start(t_2) = q_{1,i},\ label(t_2) = w_{P_{i,1}} \cdots w_{P_{i,|P_i|}}.$$

What we fail to prove yet is that this automaton will always be finite. The states of both input automata describe a finite number of dependencies between seeds. Our intuition is that the conjunction of those dependencies will also be finite. We notice that to the best of our knowledge, the closure of **LCFRS** under intersection is still an open problem.

8 (Almost obvious) Decidability Results

In the following, the complexity results consider that the size of an automaton is its number of transitions.

Lemma 6. *Given a Multiple Tree Automaton $\mathcal{A}$ and a tree alphabet Σ:*

- ***Universality:*** *if $\mathcal{A}$ is deterministic, to decide whether it recognizes all trees over Σ can be done in linear time. It $\mathcal{A}$ is non-deterministic, the same problem is in coNP.*
- ***Emptiness:*** *to decide whether the language recognized by $\mathcal{A}$ is empty can be done in linear time.*
- ***Membership:*** *given a tree t, to decide whether $t \in \mathcal{L}_\mathcal{A}$, can be done in time $\Theta(|t|)$ if the automaton is deterministic and in time $\mathcal{O}(|t|n)$ in the general case (where n is the number of states in $\mathcal{A}$).*
- ***Finiteness:*** *to decide whether the language recognized by $\mathcal{A}$ is finite can be done in linear time.*
- ***Equivalence:*** *if $\mathcal{A}$ is deterministic, and given a second deterministic Multiple Tree automaton, to decide whether they recognize the same language is polynomial.*

9 Perspectives

We recall that Multiple Tree Automaton were already used for the random sampling of trees containing a given pattern. **LCFRS** could not have been given a result as general, since it could not have dealt with pattern whose internal nodes are labelled.

The intersection of two Multiple Tree Automata should be investigated.

Multiple Tree Automata could be an interesting model for natural language processing[8]. We believe the parsing complexity will be the same as LCFRS.

In order to make this model interesting, a solid implementation of the model will have to be built, using fast algorithms for construction, membership and parsing.

References

1. Gwendal Collet, Julien David, and Alice Jacquot. Random sampling of ordered trees according to the number of occurrences of a pattern. *to be submitted*, 2014. available at http://lipn.univ-paris13.fr/ david/articles/cdj.pdf.
2. 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, 2007. release October, 12th 2007.
3. T. Flouri, B Melichar, and J. Janousek. Subtree matching by deterministic pushdown automata. In *IMCSIT'09*, volume 4, pages 659–666. IEEE Computer Society Press, 2009.
4. Irène Guesserian. Pushdown tree automata. *Math. Systems Theory*, 16:237 – 263, 1983.
5. Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. Tree adjunct grammars. *J. Comput. Syst. Sci.*, 10(1):136–163, February 1975.
6. Laura Kallmeyer. *Parsing Beyond Context-Free Grammars*. Springer Publishing Company, Incorporated, 1st edition, 2010.
7. Laura Kallmeyer. Linear context-free rewriting systems. *Language and Linguistics Compass*, 7(1):22–38, 2013.
8. Andreas Maletti and Giorgio Satta. Parsing algorithms based on tree automata. In *IWPT*, pages 1–12. The Association for Computational Linguistics, 2009.
9. Benoît Sagot and Giorgio Satta. Optimal rank reduction for linear context-free rewriting systems with fan-out two. In *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden*, pages 525–533, 2010.
10. K. Vijay-Shanker and David J. Weir. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:27–511, 1994.
11. K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*, ACL '87, pages 104–111, Stroudsburg, PA, USA, 1987. Association for Computational Linguistics.
12. David J. Weir. Linear context-free rewriting systems and deterministic tree-walking transducers. In *Proceedings of the 30th Annual Meeting on Association for Computational Linguistics*, ACL '92, pages 136–143, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

Annex

Pumping Lemma and Swapping Lemma

Proof (Lemma 2). Assume that $\mathcal{L}$ is recognized by a Multiple Tree Automaton with h transitions. Then for any forest $\vec{f} \in \mathcal{L}$ such that $height(\vec{f}) \geq h$, there exists a course c of root t such that $start(t)$ is an initial state of the automaton and whose leaves match terminal states of the automaton. c has height $height(\vec{f}) \geq h$, therefore there exists a path in c where a transition t of the automaton occurs at least twice. Let c_1 be c in which we replaced the subtree enrooted on the first encounter of t by a leaf. Let c_2 be the subtree of c enrooted in the first encounter of t, in which we replaced the subtree enrooted on the first encounter of t by a leaf. Let c_3 be the subtree of c enrooted in the second encounter of t. We obtain a well-formed course of the automaton, named $c_1[c_3]$, by replacing the subtree enrooted on the first encounter of p in c by c_3. We also obtain a well-formed course by inserting a copy of c_2 at the second encounter of p. This insertion can be iterated since it adds a third encounter of p. This yields well-formed courses written $c_1[c_2^n[c_3]], (n \geq 1)$. Then we have $l = rank(p)$, $(n_1, \ldots, n_l)$ and $(n'_1, \ldots, n'_l)$.

Lemma 7 (Swapping Lemma). *Let $\mathcal{L}$ be a language recognized by a Multiple Tree Automaton $< Q, rank, I, \Sigma, T >$. Let a forest $F \in \mathcal{L}$ and $i \leq height(F)$ such that the number of node in F of depth i is greater than $\sum_{q \in Q} rank(q)$. Then there exist three forests F_1, F_2 and F_3 such that*

- *there exist two disjoints tuple of leaves of depth i in F_1.*
- *we have $(F, \varepsilon) = C_1[(F_2 F_3, \varepsilon)]$, with $C_1 = \{F_1, (n_1, \ldots, n_l)\}$*
- *The following forest are also in $\mathcal{L}$: $C_1[(F_2 F_2, \varepsilon)]$, $C_1[(F_3 F_3, \varepsilon)]$, $C_1[(F_3 F_2, \varepsilon)]$*

Proof. Let $F \in \mathcal{L}$ be such that, at a given depth i, the number of nodes is greater than the sum of ranks of all the states, then a same state p appears at least twice at depth i in the course labelled by F. Therefore the sub-courses enrooted in the occurrences of p can be swapped or replaced by one another.

Determinism

Proof (Theorem 3). Assume that there is a deterministic automaton recognizing the same language. For $h \geq 0, 0 \leq i \leq 2^h$, let $B(h, i)$ be the unary-binary tree built from a complete binary tree of height h with one unary node attached on its i^{th} leaf. We have: $\{B(h, i), h \geq 0, 0 \leq i \leq 2^h\} \subset \mathcal{L}$. Hence the complete binary tree of height h labels a valid course of the automaton and has 2^h buds. Since the automaton is deterministic, this course C is the same for all i. In order to recognize each $B(h, i)$ for all i, one must be able to read either a leaf or an unary node from each bud. If the buds are handled with at least two states, the transitions are chosen independently in each state, losing control on the total number of unary nodes. And if the buds are handled with a unique state, it has rank 2^h, leading to unbounded ranks and states number in the automaton.

Minimization Algorithm

Theorem 4. *There is a unique, up to labelling, minimal Multiple Tree Automaton associated to each language. It is the* smallest *deterministic Multiple Tree Automaton recognizing a given language.*

Proof. Let $\mathcal{A}_1 =< Q_1, rank_1, I_1, \Sigma, T_1 >$ be a Multiple Tree automaton of minimal size. Let $\mathcal{A}_2 =< Q_2, rank_2, I_2, \Sigma, T_2 >$ be a distinct automaton recognizing the same language. Given a context C, two courses labelled by C in $\mathcal{A}_1$ and $\mathcal{A}_2$ are said to be isomorphic if the unlabelled tree structures are the same and there exists a bijection $\mu : Q_1 \to Q_2$ between the states appearing in both courses such that the labels (i.e. the transitions associated to each node) are equivalent. Two transitions t_1, t_2 are equivalent iff $t_1 = (p_1, \overrightarrow{a}, \mathcal{P}, (q_1, \ldots, q_\ell))$ and $t_2 = (\mu(p_1), \overrightarrow{a}, \mathcal{P}, (\mu(q_1), \ldots, \mu(q_\ell))$. Notice that two states related by μ are equivalent. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are distinct but recognize the same language, there exists a context which labels two non-isomorphic courses. Suppose we have two isomorphic courses and that we add a transition $t_1 = (p_1, \overrightarrow{a}, \mathcal{P}, (q_1, \ldots, q_\ell))$ in first one. Necessarily, we can add a transition $t_2 = (\mu(p_1), \overrightarrow{a}, \mathcal{P}', \overrightarrow{q'})$ in the second course. If $\mathcal{P} \neq \mathcal{P}'$, then the dependencies between seeds are not the same, which implies that one automaton has at least one splitable state. If there is no bijection ν compatible with μ which guarantees that $\overrightarrow{q'} = (\nu(q_1), \ldots, \nu(q_\ell))$, then there exists at least one element in the domain (resp. the image) of ν that should have more than one image (resp. preimage). This would imply that one of the automata contains two equivalent states. Since splitting states and merging equivalent states both strictly reduce the size of the automaton, $\mathcal{A}_2$ can be reduced ($\mathcal{A}_1$ being minimal). Finally, if there are no splitable state nor equivalent states, then $\mathcal{A}_2$ must be isomorphic to $\mathcal{A}_1$, which is then the unique minimal automaton.

Theorem 5. *The minimization algorithm is polynomial for Multiple Tree Automaton if the rank of each state is bounded by a constant.*

Proof. An automaton can only be splited and minimized a polynomial number of times. Indeed, each state p can at most be splited $rank(p)$ times. If states are not splited, some have to be merged by the minimization step, or the algorithm will stop. This can happen at most $\sum rank(p)$. Each step inside the loop can be made in polynomial time:

- **Seeds equivalence** can naively be done by comparing each pair of seeds. Each comparison can be done in constant time if the size of the alphabet and the rank of each state are bounded by a constant. Also, the number of injections from one state to another is also a constant.
- **To split the states** requires, for each state, to compute the finest partition which satisfies the three predicates given in the definition. Each of those predicates are anti-monotonic and can be checked in polynomial time. Since the rank is bounded by a constant, so is the size of the set partition lattice.

In order to find the finest partition, one can perform a breadth-first search in the lattice and stop at the first partition which satisfies the predicates.

– **State equivalence** can be computed in linear time using $\sim_s$. This step is almost identical as an iteration of Moore's algorithm on finite automata: Each state is associated to a word made of the equivalence class of its seeds. The states are sorted using a lexicographical sort. A state is equivalent to the one it follows in the list if they are both associated to the same word.

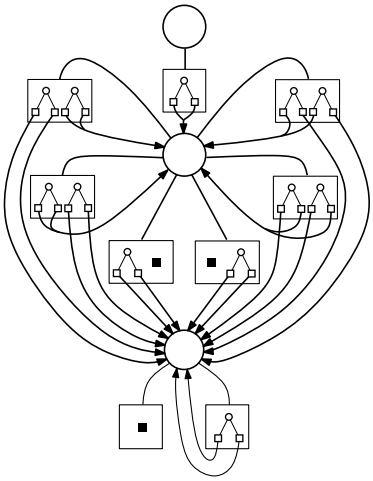


Fig. 9. A non-deterministic Multiple Tree Automaton over $\{\blacksquare, \bigwedge\}$ which recognizes the language of binary trees which are not complete, by accepting a tree only if a pair $(\blacksquare, \bigwedge)$ is read somewhere in the tree. The complement of this language cannot be recognized by a finite MTA.

Multiple Tree Automata and LCFRS

Lemma 8. *Ordered simple LCFRS and general LCFRS are equivalent.*

Proof. First, notice that ordered LCFRS and general LCRFS has already been proven to be weakly equivalent to general LCFRS in [6].

Simple LCFRS are LCFRS where the tuples appearing in the left-hand side of any production rule are made of either only terminals or only variables (while it can be a combination of both in the general case). This constraint does not restrict expressiveness, as it is sufficient to do the following substitution when $\vec{w_i}$ contains terminals and variables:

$$A(\underbrace{w_{1,0}x_{1,1}w_{1,1}\cdots x_{1,\ell_1}w_{1,\ell_1}}_{\vec{w_1}},\ldots,\underbrace{w_{n,0}x_{n,1}w_{n,1}\cdots x_{n,\ell_n}w_{n,\ell_n}}_{\vec{w_n}}) \to \ldots, \text{ with } w_{i,j} \in T^*$$

$$\Rightarrow \begin{cases} A(\underbrace{y_{1,0}x_{1,1}y_{1,1}\cdots x_{1,\ell_1}y_{1,\ell_1}}_{\vec{w_1}}, \ldots, \underbrace{y_{n,0}x_{n,1}y_{n,1}\cdots x_{n,\ell_n}y_{n,\ell_n}}_{\vec{w_n}}) \\ \to A'(y_{1,0}, x_{1,1}, y_{1,1}, \ldots, x_{1,\ell_1}, y_{1,\ell_1}, \ldots, y_{n,0}, x_{n,1}, y_{n,1}, \ldots, x_{n,\ell_n}, y_{n,\ell_n}) \\ \text{and} \\ A'(w_{1,0}, x_{1,1}, w_{1,1}, \ldots, x_{1,\ell_1}, w_{1,\ell_1}, \ldots, w_{n,0}, x_{n,1}, w_{n,1}, \ldots, x_{n,\ell_n}, w_{n,\ell_n}) \to \ldots \end{cases}$$

where A' is a new non-terminal, and $y_{i,j}$ are new variables.

Lemma 9 (Yields of Multiple Tree Automata). *Multiple Tree Automata are weakly equivalent to LCFRS.*

Proof. An **ordered simple** linear context-free rewriting system[6] is a tuple $< N, T, P, S >$ where N is a set of non-terminal symbols, $S \in N$ is a start symbol, T is a set of terminal symbols and P is a set of production rule of the form:

$$A(w_1, \ldots, w_{rank(A)}) \to B_1(x_{1,1}, \ldots, x_{1,rank(B_1)}) \cdots B_\ell(x_{\ell,1}, \ldots, x_{\ell,rank(B_\ell)}),$$

where $A, B_1, \ldots, B_\ell \in N$, the $x_{i,j}$ are variables and the w_k are either tuples of terminals or tuples of variables. A production rule in an ordered LCFRS satisfies the following properties:

- linearity: for each i, j, k, l with $i \neq k$ or $j \neq l$ $x_{i,j} \neq x_{k,l}$,
- regularity: each $x_{i,j}$ appears exactly once in $w_1 \cdots w_{rank(A)}$,
- partial ordering: if $x_{i,j}$ occurs before $x_{k,l}$ in $w_1 \cdots w_{rank(A)}$, then $\forall i, x_{i,j}$ occurs before $x_{i,l}$ in B_i if $j < l$.

We describe an algorithm which transforms an ordered simple LCFRS into a Multiple Tree Automaton, where the ranked alphabet $(\Sigma, arity)$ contains at most one internal node of each arity (only leaves carry labels). Let $\alpha : \mathbb{N} \to \Sigma$ be be the inverse function of arity. We define a function β as follows:

$$\beta(w) = \begin{cases} w, & \text{if } w \in T, \\ \alpha(|w|), & \text{otherwise.} \end{cases}$$

Let $p = A(w_1, \ldots, w_{rank(A)}) \to B_1(x_{1,1}, \ldots, x_{1,rank(B_1)}) \cdots B_\ell(x_{\ell,1}, \ldots, x_{\ell,rank(B_\ell)})$ be a production rule.

$$pos(x_{i,j}) = \text{position of } x_{i,j} \text{ in } w_1 \cdots w_{rank(A)}$$

$$P(n_i) = \{pos(x_{i,j}) | 1 \leq j \leq rank(B_i)\}$$

$$Part(p) = \{P(n_1), \ldots, P(B_\ell)\}$$

The partition $Part(p)$ encodes the repartition of variables in the production rule of a LCFRS.

[6] equivalent to general LCFRS, see Annex

Algorithm 2: From ordered simple LCFRS to MTA

Data: A Linear Context Free Rewriting System $< N, T, P, S >$
Result: A Multiple Tree Automaton $\mathcal{A} =< Q, rank, I, \Sigma, Tr >$

forall $a \in T$ **do**
    Add a state $s(a)$ in Q with $rank(s(a)) = 1$;
    Add a transition $(s(a), (a), \emptyset, \varepsilon)$ in Tr;
forall $n \in N$ **do**
    Add a state $s(n)$ in Q with $rank(s(n)) = 1$;
Add $s(S)$ in I;
forall $A(w_1, \ldots, w_{rank(A)}) \to B_1(x_{1,1}, \ldots, x_{1,rank(B_1)}) \cdots B_\ell(x_{\ell,1}, \ldots, x_{\ell,rank(B_\ell)}) \in P$ **do**
    Add the transition $(A, (\beta(w_1), \ldots, \beta(w_{rank(A)})), Part(p), (B_1, \ldots, B_\ell))$ in Tr;
return $< Q, rank, I, \Sigma, Tr >$;

In order to see that this transformation is a bijection, one also has to consider Multiple Tree Automata for which all transitions t have the following property: $label(t)$ contains either only leaves, or only internal nodes (the *transition-property*). In order to convert a Multiple Tree Automaton into one that satisfies those properties, one has to add a state A_a for each leaf a. Then, for each transition that do not satisfy the transition-property, replace each occurrence of a by a letter b of arity 1, A being the arrival state for the new bud. For each state A_a, one has to add the transition $(A_a, (a), \emptyset, \varepsilon)$.

Properties

Lemma 10 (Universality). *Given a deterministic Multiple Tree Automaton and a tree alphabet Σ, to decide whether it recognizes all trees over Σ can be done in linear time. Given a non-deterministic Multiple Tree Automaton, the same problem is in coNP.*

Proof. Indeed, a deterministic Multiple Tree Automaton is universal if and only if it is complete, which can easily be checked. If the automaton is non-deterministic with n states, then it is sufficient to show that for all tree of height $n+1$ is recognized and use the pumping lemma. That can be done in exponential time and linear space. On the other hand, if the automaton is not universal, one can check a counter-example in polynomial time.

Lemma 11 (Emptiness). *Given a Multiple Tree Automaton, to decide whether the language it recognizes is empty can be done in linear time.*

Proof. The language recognized by an automaton is empty if does not contain any state after being trimmed. This can be done in linear time according to Lemma 1.

Lemma 12 (Equivalence). *Given two deterministic Multiple Tree Automata, to decide whether they recognize the same language is polynomial.*

Proof. First minimize both automata. Since the minimal automaton is unique, one just needs to compare the two resulting minimal automata.

Lemma 13 (Membership). *Given a Multiple Tree Automaton with n states recognizing a language $\mathcal{L}$ and a tree t, to decide whether $t \in \mathcal{L}$, can be done in time $\Theta(|t|)$ if the automaton is deterministic and in time $\mathcal{O}(|t|n)$ in the general case.*

To test the membership in a Multiple Tree Automaton is similar to finite state automata, though the implementation, which we will not describe here, is more complicated. Note that the computation can be parallelized since it requires to do independent test on the subtrees of the input tree.

Lemma 14 (Finiteness). *Given a Multiple Tree Automaton, to decide whether the language it recognizes is finite can be done in linear time.*

Proof. A trimmed Multiple Tree Automaton recognizes a finite language if it does not contain a loop, which can be checked in linear time. We conclude using Lemma 1.