Journal of Machine Learning Research 16 (2015) 2465-2500 Submitted 11/14; Revised 6/15; Published 12/15

Complexity of Equivalence and Learning for Multiplicity Tree Automata

Ines Marušić INES.MARUSIC@CS.OX.AC.UK
James Worrell JBW@CS.OX.AC.UK
Department of Computer Science
University of Oxford
Parks Road, Oxford OX1 3QD, UK

Editor: Alexander Clark

Abstract

We consider the query and computational complexity of learning multiplicity tree automata in Angluin's exact learning model. In this model, there is an oracle, called the Teacher, that can answer membership and equivalence queries posed by the Learner. Motivated by this feature, we first characterise the complexity of the equivalence problem for multiplicity tree automata, showing that it is logspace equivalent to polynomial identity testing.

We then move to query complexity, deriving lower bounds on the number of queries needed to learn multiplicity tree automata over both fixed and arbitrary fields. In the latter case, the bound is linear in the size of the target automaton. The best known upper bound on the query complexity over arbitrary fields derives from an algorithm of Habrard and Oncina (2006), in which the number of queries is proportional to the size of the target automaton and the size of a largest counterexample, represented as a tree, that is returned by the Teacher. However, a smallest counterexample tree may already be exponential in the size of the target automaton. Thus the above algorithm has query complexity exponentially larger than our lower bound, and does not run in time polynomial in the size of the target automaton.

We give a new learning algorithm for multiplicity tree automata in which counterexamples to equivalence queries are represented as DAGs. The query complexity of this algorithm is quadratic in the target automaton size and linear in the size of a largest counterexample. In particular, if the Teacher always returns DAG counterexamples of minimal size then the query complexity is quadratic in the target automaton size—almost matching the lower bound, and improving the best previously-known algorithm by an exponential factor.

Keywords: exact learning, query complexity, multiplicity tree automata, Hankel matrices, DAG representations of trees, polynomial identity testing

1. Introduction

Trees are a basic object in computer science and a natural model of hierarchical data, such as syntactic structures in natural language processing and XML data on the web. Trees arise across a broad range of applications, including natural text and speech processing, computer vision, bioinformatics, web information extraction, and social network analysis. Many of these applications require representing probability distributions over trees and more general functions from trees into the real numbers. A broad class of such functions can be defined by *multiplicity tree automata*, a powerful algebraic model which strictly generalises probabilistic tree automata.

Multiplicity tree automata were introduced by Berstel and Reutenauer (1982) under the terminology of linear representations of tree series. They augment classical finite tree automata by assigning to each transition a value in a field. They also generalise multiplicity word automata, introduced by Schützenberger (1961), since words are a special case of trees. Multiplicity tree automata define many natural structural properties of trees and can be used to model probabilistic processes running on trees. Multiplicity word and tree automata have been applied to a wide variety of machine learning problems, including speech recognition, image processing, character recognition, and grammatical inference; see the paper of Balle and Mohri (2012) for references.

The task of learning automata from examples and queries has been extensively studied since the 1960s. Two notable results in this domain show the impossibility of efficiently learning deterministic finite automata from positive and negative examples alone. First, Gold (1978) showed that the problem of exactly identifying the smallest deterministic finite automaton consistent with a set of accepted and rejected words is NP-hard. Later, Kearns and Valiant (1994) showed that the concept class of regular languages is not efficiently PAC learnable using any polynomially-evaluable hypothesis class under standard cryptographic assumptions.

A significant positive result on learning regular languages was achieved by Angluin (1987), who considered a Learner that did not just passively receive data but that was also able to ask queries. Specifically, Angluin considered *membership queries*, in which the Learner asks an oracle whether a given word belongs to the target language, and *equivalence queries*, in which the Learner asks an oracle whether a hypothesis is correct, obtaining a counterexample if it is not. Subsequent research has sought to establish the learnability of many other hypothesis classes in the same setting, including classes of Boolean formulae, decision trees, context-free languages, and polynomials; see the book of Kearns and Vazirani (1994, Chapter 8) for more details and references.

In this paper we study the problem of learning multiplicity tree automata in the *exact learning model* of Angluin (1988), outlined above. Formally, in this model a Learner actively collects information about the target function from a Teacher through *membership queries*, which ask for the value of the function on a specific input, and *equivalence queries*, which suggest a hypothesis to which the Teacher provides a counterexample if one exists. A class of functions $\mathscr{C}$ is *exactly learnable* if there exists an exact learning algorithm such that for any function $f \in \mathscr{C}$, the Learner identifies f using polynomially many membership and equivalence queries in the size of a shortest representation of f and the size of a largest counterexample returned by the Teacher during the execution of the algorithm. The exact learning model is an important theoretical model of the learning process. It is well known that learnability in the exact learning model also implies learnability in the PAC model with membership queries (Valiant, 1985).

We are interested in questions of succinctness and computational efficiency, both from the point of view of the Teacher and the Learner. From the point of view of the Teacher, one of the main questions is checking *equivalence* of multiplicity tree automata, i.e., whether two multiplicity tree automata define the same function on trees. Seidl (1990) proved that equivalence of multiplicity tree automata is decidable in polynomial time assuming unit-cost arithmetic, and in randomised polynomial time in the usual bit-cost model. No finer analysis of the complexity of this problem exists to date. In contrast, the complexity of equivalence for classical nondeterministic word and tree automata has been completely characterised: PSPACE-complete over words (Aho et al., 1974) and EXPTIME-complete over trees (Seidl, 1990).

Our first contribution, in Section 3, is to show that the equivalence problem for multiplicity tree automata is logspace equivalent to polynomial identity testing, i.e., the problem of deciding whether a polynomial given as an arithmetic circuit is zero. The latter problem is known to be solvable in randomised polynomial time (DeMillo and Lipton, 1978; Schwartz, 1980; Zippel, 1979), whereas solving it in deterministic polynomial time is a well-studied and longstanding open problem (see Arora and Barak, 2009).

Our second contribution, in Section 5, is to give lower bounds on the number of queries needed to learn multiplicity tree automata in the exact learning model, both for the case of an arbitrary and a fixed underlying field. The bound in the former case is linear in the automaton size. In the latter case, the bound is linear in the automaton size for alphabets of a fixed maximal rank. To the best of our knowledge, these are the first lower bounds on the query complexity of exactly learning multiplicity tree automata.

Habrard and Oncina (2006) give an algorithm for learning multiplicity tree automata in the exact learning model. Consider a target multiplicity tree automaton whose minimal representation A has n states. The algorithm of Habrard and Oncina, *op. cit.*, makes at most n equivalence queries and number of membership queries proportional to $|A| \cdot s$, where $|A|$ is the size of A and s is the size of a largest counterexample returned by the Teacher. Since this algorithm assumes that the Teacher returns counterexamples represented explicitly as trees, s can be exponential in $|A|$, even for a Teacher that returns counterexamples of minimal size (see Example 3). This observation reveals an exponential gap between the query complexity of the algorithm of Habrard and Oncina (2006) and our above-mentioned lower bound, which is only linear in $|A|$. Another consequence is that the worst-case time complexity of this algorithm is exponential in the size of the target automaton.

Given two inequivalent multiplicity tree automata with n states in total, the algorithm of Seidl (1990) produces a subtree-closed set of trees of cardinality at most n that contains a tree on which the automata differ. It follows that the counterexample contained in this set has at most n subtrees, and hence can be represented as a DAG with at most n vertices (see Section 3.2). Thus in the context of exact learning it is natural to consider a Teacher that can return succinctly-represented counterexamples, i.e., trees represented as DAGs.

DAGs have been used as succinct representations of trees in a number of domains, including classification problems (Sperduti and Starita, 1997) and query evaluation for XML (Buneman et al., 2003; Frick et al., 2003). Tree automata that run on DAG representations of finite trees were first introduced by Charatonik (1999) as extensions of ordinary tree automata, and were further studied by Anantharaman et al. (2005). The automata considered by Charatonik (1999) and Anantharaman et al. (2005) run on fully-compressed DAGs. Fila and Anantharaman (2006) extend this definition by introducing tree automata that run on DAGs that may be partially compressed. In this paper, we employ the latter framework in the context of learning multiplicity automata.

In Section 4, we present a new exact learning algorithm for multiplicity tree automata that achieves the same bound on the number of equivalence queries as the algorithm of Habrard and Oncina (2006), while using number of membership queries quadratic in the target automaton size and linear in the largest counterexample size, even when counterexamples are given as DAGs. Assuming that the Teacher provides minimal DAG representations of counterexamples, our algorithm therefore makes quadratically many queries in the target automaton size. This is exponentially fewer queries than the best previously-known algorithm (Habrard and Oncina, 2006) and quadratic in the above-mentioned lower bound. Furthermore, our algorithm performs a quadratic number of arithmetic operations in the size of the target automaton, and can be implemented in randomised polynomial time in the Turing model.

Like the algorithm of Habrard and Oncina (2006), our algorithm constructs a matricial representation of the target automaton, called the *Hankel matrix* (Carlyle and Paz, 1971; Fliess, 1974). However on receiving a counterexample tree z, the former algorithm adds a new column to the Hankel matrix for every suffix of z, while our algorithm adds (at most) one new row for each subtree of z. Crucially the number of suffixes may be exponential in the size of a DAG representation of z, whereas the number of subtrees is only linear in the size of a DAG representation.

An extended abstract (Marušić and Worrell, 2014) of this work appeared in the proceedings of MFCS 2014. The current paper contains full proofs of all results reported there, the formal definition of multiplicity tree automata running on DAGs, and a refined complexity analysis of the learning algorithm.

1.1 Related Work

One of the earliest results about the exact learning model was the proof of Angluin (1987) that deterministic finite automata are learnable. This result was generalised by Drewes and Högberg (2007) to show exact learnability of deterministic finite (bottom-up) tree automata, generalising also a result of Sakakibara (1990) on the exact learnability of context-free grammars from their structural descriptions[1].

The learning algorithm of Drewes and Högberg (2007) was generalised by Maletti (2007) to show that deterministic weighted tree automata over a (commutative) semifield are exactly learnable, generalising also an earlier result of Drewes and Vogler (2007) which was restricted to the class of deterministic *all-accepting* (i.e., every final weight is non zero) weighted tree automata. Recently, a unifying framework for exact learning of deterministic weighted tree automata over a semifield has been proposed (Drewes et al., 2011). Specifically, Drewes et al., *op. cit.*, introduce the notion of *abstract observation tables*, an abstract data type for learning deterministic weighted tree automata in the exact learning model, and show that every correct implementation of abstract observation tables yields a correct learning algorithm.

Exact learnability of nondeterministic weighted automata over a field (here called *multiplicity automata*) has also been extensively studied. Beimel et al. (2000) show that multiplicity word automata can be learned efficiently, and apply this to learn various classes of DNF formulae and polynomials. These results were generalised by Klivans and Shpilka

[1] *Structural descriptions* of a context-free grammar are unlabelled derivation trees of the grammar.

(2006) to show exact learnability of restricted algebraic branching programs and noncommutative set-multilinear arithmetic formulae. Bisht et al. (2006) give an almost tight (up to a *log* factor) lower bound on the number of queries made by any exact learning algorithm for the class of multiplicity word automata.

An exact learning algorithm for a class of nondeterministic tree automata, namely *residual finite* tree automata, is given by Kasprzik (2013). The latter paper identifies the size of counterexamples as a hidden exponential factor in the complexity of the learning algorithm, observing in particular that a smallest counterexample can have exponential size in the number of states of the target automaton. Such a phenomenon does not prevent the class of tree automata from being exactly learnable since in the exact learning model the complexity measure takes into account the size of a largest counterexample. However, this does raise the question of developing a learning algorithm whose complexity would be polynomial in the size of succinctly-represented counterexamples, which is one of the motivations for the present work.

Denis and Habrard (2007) consider the problem of learning probability distributions over trees that are recognised by a multiplicity tree automaton from samples drawn independently according to the target distribution. They give an inference algorithm that exactly identifies such recognisable probability distributions in the limit with probability one (with respect to the randomly-drawn examples). Most closely related to the topic of the present paper is the work of Habrard and Oncina (2006), who give an algorithm for learning multiplicity tree automata in the exact learning model, as discussed above.

A variety of spectral methods have been employed for learning multiplicity word and tree automata (Bailly et al., 2009; Balle and Mohri, 2012; Denis et al., 2014; Gybels et al., 2014). This line of research originates in earlier work of Hsu et al. (2012) that gives a spectral learning algorithm (based on singular value decomposition) for hidden Markov models. Particularly close to the present paper is the work of Bailly et al. (2010), which learns probability distributions over trees that are recognised by some multiplicity tree automaton. Their approach lies within a passive learning framework in which one is given a sample of trees independently drawn according to a target distribution, and the aim is to infer a multiplicity tree automaton that approximates the target. As in our approach, the notion of a Hankel matrix plays a central role in the algorithm of Bailly et al. (2010). There the Hankel matrix is called an *observation matrix*, and it encodes an empirical distribution on trees obtained by sampling from the target distribution. Bailly et al., *op. cit.*, apply principal component analysis in order to identify a low-dimensional approximation of the vector space spanned by the residuals of the target probability distribution. From this approximation they build an automaton whose associated tree series approximates the target distribution. They moreover obtain bounds on the estimation error of the output tree series with respect to the target distribution in terms of the sample size and the desired confidence.

In contrast to the above-described approach of Bailly et al. (2010), in our work the target dimension (i.e., number of states) is not part of the input since our aim is to learn a *minimal* multiplicity tree automaton that exactly represents the target tree series. Moreover, in the present paper the entries of the Hankel matrix are determined by active queries rather than passive observations, and the learning process continues until we know a sufficient number of entries to be able to exactly construct a representation of the target.

2. Preliminaries

Let $\mathbb{N}$ and $\mathbb{N}_0$ denote the set of all positive and nonnegative integers, respectively. Let $n \in \mathbb{N}$. We write $[n]$ for the set $\{1, 2, \ldots, n\}$ and I_n for the identity matrix of order n. For every $i \in [n]$, we write e_i for the i^{th} n-dimensional coordinate row vector. For any n-dimensional vector v, we write v_i for its i^{th} entry.

For any matrix A, we write A_i for its i^{th} row, A^j for its j^{th} column, and $A_{i,j}$ for its $(i,j)^{\text{th}}$ entry. Given nonempty subsets I and J of the rows and columns of A, respectively, we write $A_{I,J}$ for the submatrix $(A_{i,j})_{i \in I, j \in J}$ of A. For singletons, we write simply $A_{i,J} := A_{\{i\},J}$ and $A_{I,j} := A_{I,\{j\}}$. We also consider matrices whose rows and columns are indexed by tuples of natural numbers ordered lexicographically.

Given a set V, we denote by V^* the set of all finite ordered tuples of elements from V. For any subset $S \subseteq V$, the *characteristic function* of S (relative to V) is the function $\chi_S : V \to \{0,1\}$ such that $\chi_S(x) = 1$ if $x \in S$, and $\chi_S(x) = 0$ otherwise.

2.1 Kronecker Product

Let A be a matrix of dimension $m_1 \times n_1$ and B a matrix of dimension $m_2 \times n_2$. The *Kronecker product* of A by B, written as $A \otimes B$, is a matrix of dimension $m_1 m_2 \times n_1 n_2$ where $(A \otimes B)_{(i_1,i_2),(j_1,j_2)} = A_{i_1,j_1} \cdot B_{i_2,j_2}$ for every $i_1 \in [m_1]$, $i_2 \in [m_2]$, $j_1 \in [n_1]$, $j_2 \in [n_2]$.

The Kronecker product is bilinear, associative, and has the following *mixed-product property*: For any matrices A, B, C, D such that products $A \cdot C$ and $B \cdot D$ are defined, it holds that $(A \otimes B) \cdot (C \otimes D) = (A \cdot C) \otimes (B \cdot D)$.

Let $k \in \mathbb{N}$ and $A_1, \ldots, A_k$ be matrices such that for every $l \in [k]$, matrix A_l has n_l rows. It can easily be shown using induction on k that for every $(i_1, \ldots, i_k) \in [n_1] \times \cdots \times [n_k]$, it holds that

$$(A_1 \otimes \cdots \otimes A_k)_{(i_1,\ldots,i_k)} = (A_1)_{i_1} \otimes \cdots \otimes (A_k)_{i_k}. \tag{1}$$

We write $\bigotimes_{l=1}^{k} A_l := A_1 \otimes \cdots \otimes A_k$.

For every $k \in \mathbb{N}_0$ we define the *k-fold Kronecker power* of a matrix A, written as $A^{\otimes k}$, inductively by $A^{\otimes 0} = I_1$ and $A^{\otimes k} = A^{\otimes (k-1)} \otimes A$ for $k \geq 1$.

Let $k \in \mathbb{N}_0$. For any square matrices A and B, we have

$$(A \otimes B)^k = A^k \otimes B^k. \tag{2}$$

For any matrices $A_1, \ldots, A_k$ and $B_1, \ldots, B_k$ where product $A_l \cdot B_l$ is defined for every $l \in [k]$, we have

$$(A_1 \otimes \cdots \otimes A_k) \cdot (B_1 \otimes \cdots \otimes B_k) = (A_1 \cdot B_1) \otimes \cdots \otimes (A_k \cdot B_k). \tag{3}$$

Equations (2) and (3) follow easily from the mixed-product property by induction on k.

2.2 Finite Trees

A *ranked alphabet* is a tuple (Σ, rk) where Σ is a nonempty finite set of symbols and $rk : \Sigma \to \mathbb{N}_0$ is a function. Ranked alphabet (Σ, rk) is often written Σ for short. For every

$k \in \mathbb{N}_0$, we define the set of all *k-ary* symbols $\Sigma_k := rk^{-1}(\{k\})$. If $\sigma \in \Sigma_k$ then we say that σ has *rank* (or *arity*) k. We say that Σ has *rank* m if $m = max\{rk(\sigma) : \sigma \in \Sigma\}$.

The set of Σ-*trees* (*trees* for short), written as T_Σ, is the smallest set T satisfying the following two conditions: (i) $\Sigma_0 \subseteq T$; and (ii) if $k \geq 1$, $\sigma \in \Sigma_k$, $t_1, \ldots, t_k \in T$ then $\sigma(t_1, \ldots, t_k) \in T$. Given a Σ-tree t, a *subtree* of t is a Σ-tree consisting of a node in t and all of its descendants in t. The set of all subtrees of t is denoted by $Sub(t)$.

Let Σ be a ranked alphabet and $\mathbb{F}$ be a field. A *tree series* over Σ with coefficients in $\mathbb{F}$ is a function $f : T_\Sigma \to \mathbb{F}$. For every $t \in T_\Sigma$, we call $f(t)$ the *coefficient* of t in f. The set of all tree series over Σ with coefficients in $\mathbb{F}$ is denoted by $\mathbb{F}\langle\langle T_\Sigma \rangle\rangle$.

We define the tree series $height, size, \#_\sigma \in \mathbb{Q}\langle\langle T_\Sigma \rangle\rangle$ where $\sigma \in \Sigma$, as follows: (i) if $t \in \Sigma_0$ then $height(t) = 0$, $size(t) = 1$, $\#_\sigma(t) = \chi_{\{t=\sigma\}}$; and (ii) if $t = a(t_1, \ldots, t_k)$ where $k \geq 1$, $a \in \Sigma_k$, $t_1, \ldots, t_k \in T_\Sigma$ then $height(t) = 1 + max_{i\in[k]} height(t_i)$, $size(t) = 1 + \sum_{i\in[k]} size(t_i)$, $\#_\sigma(t) = \chi_{\{a=\sigma\}} + \sum_{i\in[k]} \#_\sigma(t_i)$, respectively. For every $n \in \mathbb{N}_0$, we define the sets $T_\Sigma^{<n} := \{t \in T_\Sigma : height(t) < n\}$, $T_\Sigma^n := \{t \in T_\Sigma : height(t) = n\}$, and $T_\Sigma^{\leq n} := T_\Sigma^{<n} \cup T_\Sigma^n$.

Let $\Box$ be a nullary symbol not contained in Σ. The set C_Σ of Σ-*contexts* (*contexts* for short) is the set of all $(\{\Box\} \cup \Sigma)$-trees in which $\Box$ occurs exactly once. The *concatenation* of $c \in C_\Sigma$ and $t \in T_\Sigma \dot\cup C_\Sigma$, written as $c[t]$, is the tree obtained by substituting t for $\Box$ in c. Intuitively, the $\Box$-labelled leaf of c acts as a variable in that substituting a Σ-tree (respectively, Σ-context) t for that variable yields a new Σ-tree (Σ-context) $c[t]$.

A *suffix* of a Σ-tree t is a Σ-context c such that $t = c[t']$ for some Σ-tree t'. The *Hankel matrix* of a tree series $f \in \mathbb{F}\langle\langle T_\Sigma \rangle\rangle$ is the matrix $H : T_\Sigma \times C_\Sigma \to \mathbb{F}$ such that $H_{t,c} = f(c[t])$ for every $t \in T_\Sigma$ and $c \in C_\Sigma$.

2.3 Multiplicity Tree Automata

Let $\mathbb{F}$ be a field. An $\mathbb{F}$-*multiplicity tree automaton* ($\mathbb{F}$-*MTA*) is a quadruple $A = (n, \Sigma, \mu, \gamma)$ which consists of the *dimension* $n \in \mathbb{N}_0$ representing the number of states, a ranked alphabet Σ, a family of *transition matrices* $\mu = \{\mu(\sigma) : \sigma \in \Sigma\}$, where $\mu(\sigma) \in \mathbb{F}^{n^{rk(\sigma)} \times n}$, and the *final weight vector* $\gamma \in \mathbb{F}^{n \times 1}$. The *size* of the automaton A, written as $|A|$, is defined as

$$|A| := \sum_{\sigma \in \Sigma} n^{rk(\sigma)+1} + n.$$

That is, the size of A is the total number of entries in all transition matrices and the final weight vector.[2]

Example 1 *Let $\Sigma = \{0, 1, +, \times, -\}$ be a ranked alphabet where $0, 1$ are nullary symbols and $+, \times, -$ are binary symbols. We define an $\mathbb{F}$-multiplicity tree automaton $A = (2, \Sigma, \mu, \gamma)$ as follows. Automaton A has two states, q_1 and q_2, and has final weight vector $\gamma = \begin{bmatrix} 0 & 1 \end{bmatrix}^\top$. This means that states q_1 and q_2 have final weights $\gamma_1 = 0$ and $\gamma_2 = 1$, respectively. Given a symbol $\sigma \in \Sigma$ of rank k, the transition matrix $\mu(\sigma)$ has dimension $2^k \times 2$ and stores the weights of transitions from each k-tuple of origin states to each destination state. Let the*

[2] We measure size assuming explicit rather than sparse representations of the transition matrices and final weight vector because minimal automata are only unique up to change of basis (see Theorem 4).

transition matrices of A be $\mu(0) = \begin{bmatrix} 1 & 0 \end{bmatrix}$, $\mu(1) = \begin{bmatrix} 1 & 1 \end{bmatrix}$,

$$\mu(+) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \; \mu(-) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \; \textit{and } \mu(\times) = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

Entry $\mu(1)_2 = 1$ means that there is a transition $1 \xrightarrow{1} q_2$ with weight 1 into state q_2 on reading symbol 1. Similarly, entry $\mu(+)_{(2,1),2} = 1$ means that there is a transition $+(q_2, q_1) \xrightarrow{1} q_2$ with weight 1 from pair of states (q_2, q_1) into state q_2 on reading symbol $+$.

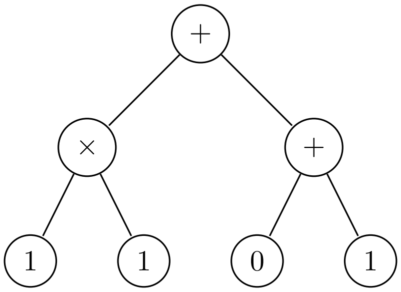
We extend μ from Σ to T_Σ by defining

$$\mu(\sigma(t_1, \dots, t_k)) := (\mu(t_1) \otimes \cdots \otimes \mu(t_k)) \cdot \mu(\sigma)$$

for every $\sigma \in \Sigma_k$ and $t_1, \dots, t_k \in T_\Sigma$. The tree series $\|A\| \in \mathbb{F}\langle\langle T_\Sigma \rangle\rangle$ *recognised* by A is defined by $\|A\|(t) = \mu(t) \cdot \gamma$ for every $t \in T_\Sigma$. Note that a 0-dimensional multiplicity tree automaton necessarily recognises a zero tree series. Two automata A_1, A_2 are said to be *equivalent* if $\|A_1\| \equiv \|A_2\|$.

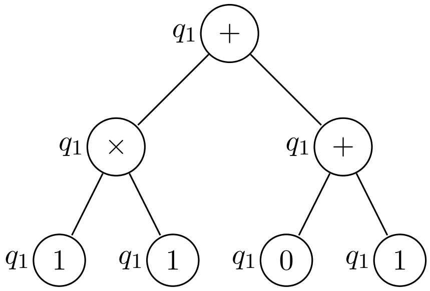
We further extend μ from T_Σ to C_Σ by treating $\square$ as a unary symbol and defining $\mu(\square) := I_n$. This allows to define $\mu(c) \in \mathbb{F}^{n \times n}$ for every $c = \sigma(t_1, \dots, t_k) \in C_\Sigma$ inductively by writing $\mu(c) := (\mu(t_1) \otimes \cdots \otimes \mu(t_k)) \cdot \mu(\sigma)$. It can easily be shown that $\mu(c[t]) = \mu(t) \cdot \mu(c)$ for every $t \in T_\Sigma$ and $c \in C_\Sigma$.

Example 2 *Let us consider the computation of $\mathbb{F}$-MTA $A = (2, \Sigma, \mu, \gamma)$ from Example 1 on the following Σ-tree t:*

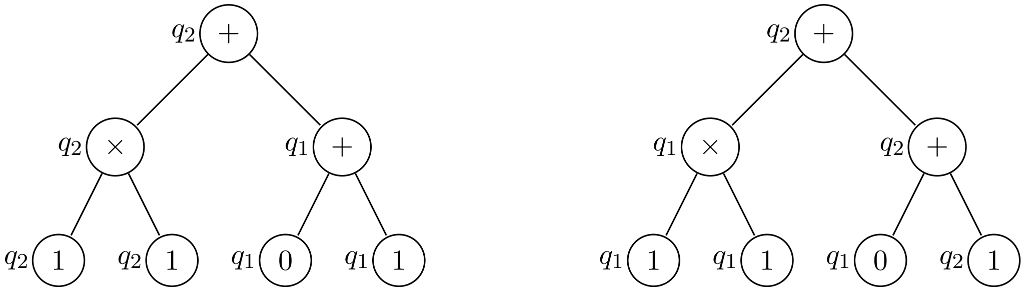


The transition matrices define bottom-up runs of A on t. Intuitively, a run on t corresponds to multiple copies of automaton A walking along t from leaves to the root. Every such run has a weight in $\mathbb{F}$ which is defined as the product of the weights of all transitions taken.

On tree t, automaton A has one nonzero-weight run ending in state q_1, as follows:



Moreover, automaton A has two nonzero-weight runs ending in state q_2, as follows:



Each of the above three runs has weight 1. Therefore, the total weight of all runs of automaton A on tree t in which the root is labelled q_1 is 1, and the total weight of all runs in which the root is labelled q_2 is 2. Indeed, algebraically, by definition of μ we have that

$$\begin{aligned}
\mu(t) &= (\mu(\times(1,1)) \otimes \mu(+(0,1))) \cdot \mu(+) \\
&= (((\mu(1) \otimes \mu(1)) \cdot \mu(\times)) \otimes ((\mu(0) \otimes \mu(1)) \cdot \mu(+))) \cdot \mu(+) \\
&= \left(\left(\left(\begin{bmatrix}1 & 1\end{bmatrix} \otimes \begin{bmatrix}1 & 1\end{bmatrix}\right) \cdot \begin{bmatrix}1 & 0\\0 & 0\\0 & 0\\0 & 1\end{bmatrix} \right) \otimes \left(\left(\begin{bmatrix}1 & 0\end{bmatrix} \otimes \begin{bmatrix}1 & 1\end{bmatrix}\right) \cdot \begin{bmatrix}1 & 0\\0 & 1\\0 & 1\\0 & 0\end{bmatrix} \right) \right) \cdot \begin{bmatrix}1 & 0\\0 & 1\\0 & 1\\0 & 0\end{bmatrix} \\
&= \left(\left(\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix} \cdot \begin{bmatrix}1 & 0\\0 & 0\\0 & 0\\0 & 1\end{bmatrix} \right) \otimes \left(\begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix} \cdot \begin{bmatrix}1 & 0\\0 & 1\\0 & 1\\0 & 0\end{bmatrix} \right) \right) \cdot \begin{bmatrix}1 & 0\\0 & 1\\0 & 1\\0 & 0\end{bmatrix} \\
&= \left(\begin{bmatrix}1 & 1\end{bmatrix} \otimes \begin{bmatrix}1 & 1\end{bmatrix}\right) \cdot \begin{bmatrix}1 & 0\\0 & 1\\0 & 1\\0 & 0\end{bmatrix} = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix} \cdot \begin{bmatrix}1 & 0\\0 & 1\\0 & 1\\0 & 0\end{bmatrix} = \begin{bmatrix}1 & 2\end{bmatrix}.
\end{aligned}$$

Finally, the weight $\|A\|(t)$ of tree t is the sum of the weights of all runs on t, where the weight of each run is multiplied by the final weight of its root label. Algebraically, we have

$$\|A\|(t) = \mu(t) \cdot \gamma = \begin{bmatrix}1 & 2\end{bmatrix} \cdot \begin{bmatrix}0 & 1\end{bmatrix}^\top = 2.$$

Let $A_1 = (n_1, \Sigma, \mu_1, \gamma_1)$ and $A_2 = (n_2, \Sigma, \mu_2, \gamma_2)$ be two $\mathbb{F}$-multiplicity tree automata. The *product* of A_1 by A_2, written as $A_1 \times A_2$, is the $\mathbb{F}$-multiplicity tree automaton (n, Σ, μ, γ) where:

- $n = n_1 \cdot n_2$;
- If $\sigma \in \Sigma_k$ then $\mu(\sigma) = P_k \cdot (\mu_1(\sigma) \otimes \mu_2(\sigma))$ where P_k is a permutation matrix of order $(n_1 \cdot n_2)^k$ uniquely defined (see Remark 1 below) by

$$(u_1 \otimes \cdots \otimes u_k) \otimes (v_1 \otimes \cdots \otimes v_k) = ((u_1 \otimes v_1) \otimes \cdots \otimes (u_k \otimes v_k)) \cdot P_k \qquad (4)$$

  for all $u_1, \ldots, u_k \in \mathbb{F}^{1 \times n_1}$ and $v_1, \ldots, v_k \in \mathbb{F}^{1 \times n_2}$;

- $\gamma = \gamma_1 \otimes \gamma_2$.

Remark 1 *We argue that for every $k \in \mathbb{N}_0$ such that k is the rank of a symbol in Σ, matrix P_k is well-defined by Equation (4). In order to do this, it suffices to show that P_k is well-defined on a set of basis vectors of $\mathbb{F}^{1\times n_1}$ and $\mathbb{F}^{1\times n_2}$ and then extend linearly. To that end, let $(e^1_i)_{i\in[n_1]}$ and $(e^2_j)_{j\in[n_2]}$ be bases of $\mathbb{F}^{1\times n_1}$ and $\mathbb{F}^{1\times n_2}$, respectively. Let us define sets of vectors*

$$E_1 := \{(e^1_{i_1} \otimes \cdots \otimes e^1_{i_k}) \otimes (e^2_{j_1} \otimes \cdots \otimes e^2_{j_k}) : i_1, \ldots, i_k \in [n_1], j_1, \ldots, j_k \in [n_2]\}$$

and

$$E_2 := \{(e^1_{i_1} \otimes e^2_{j_1}) \otimes \cdots \otimes (e^1_{i_k} \otimes e^2_{j_k}) : i_1, \ldots, i_k \in [n_1], j_1, \ldots, j_k \in [n_2]\}.$$

Then, E_1 and E_2 are two bases of the vector space $\mathbb{F}^{1\times n_1 n_2}$. Therefore, P_k is well-defined as an invertible matrix mapping basis E_1 to basis E_2.

Essentially the same product construction as in the proof of the first part of the following proposition is given by Berstel and Reutenauer (1982, Proposition 5.1) using the terminology of linear representations of tree series rather than multiplicity tree automata.

Proposition 2 *Let A_1 and A_2 be $\mathbb{F}$-multiplicity tree automata over a ranked alphabet Σ. Then, for every $t \in T_\Sigma$ it holds that $\|A_1 \times A_2\|(t) = \|A_1\|(t) \cdot \|A_2\|(t)$. Furthermore, in case $\mathbb{F} = \mathbb{Q}$, automaton $A_1 \times A_2$ can be computed from A_1 and A_2 in logarithmic space.*

Proof Let $A_1 = (n_1, \Sigma, \mu_1, \gamma_1)$, $A_2 = (n_2, \Sigma, \mu_2, \gamma_2)$, and $A_1 \times A_2 = (n, \Sigma, \mu, \gamma)$. First we show that for any $t \in T_\Sigma$,

$$\mu(t) = \mu_1(t) \otimes \mu_2(t). \tag{5}$$

We prove that Equation (5) holds for all $t \in T_\Sigma$ using induction on *height*(t). The base case $t = \sigma \in \Sigma_0$ holds immediately by definition since $P_0 = I_1$. For the induction step, let $h \in \mathbb{N}_0$ and assume that Equation (5) holds for every $t \in T^{\leq h}_\Sigma$. Take any $t \in T^{h+1}_\Sigma$. Then $t = \sigma(t_1, \ldots, t_k)$ for some $k \geq 1$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T^{\leq h}_\Sigma$. By induction hypothesis, Equation (4), and the mixed-product property of Kronecker product we now have

$$\begin{aligned}
\mu(t) &= (\mu(t_1) \otimes \cdots \otimes \mu(t_k)) \cdot \mu(\sigma)\\
&= ((\mu_1(t_1) \otimes \mu_2(t_1)) \otimes \cdots \otimes (\mu_1(t_k) \otimes \mu_2(t_k))) \cdot P_k \cdot (\mu_1(\sigma) \otimes \mu_2(\sigma))\\
&= ((\mu_1(t_1) \otimes \cdots \otimes \mu_1(t_k)) \otimes (\mu_2(t_1) \otimes \cdots \otimes \mu_2(t_k))) \cdot (\mu_1(\sigma) \otimes \mu_2(\sigma))\\
&= ((\mu_1(t_1) \otimes \cdots \otimes \mu_1(t_k)) \cdot \mu_1(\sigma)) \otimes ((\mu_2(t_1) \otimes \cdots \otimes \mu_2(t_k)) \cdot \mu_2(\sigma))\\
&= \mu_1(t) \otimes \mu_2(t).
\end{aligned}$$

This completes the proof of Equation (5) by induction. For every $t \in T_\Sigma$, we now have

$$\begin{aligned}
\|A_1 \times A_2\|(t) &= \mu(t) \cdot \gamma = (\mu_1(t) \otimes \mu_2(t)) \cdot (\gamma_1 \otimes \gamma_2)\\
&= (\mu_1(t) \cdot \gamma_1) \otimes (\mu_2(t) \cdot \gamma_2) = \|A_1\|(t) \otimes \|A_2\|(t) = \|A_1\|(t) \cdot \|A_2\|(t).
\end{aligned}$$

Automaton $A_1 \times A_2$ can be computed by a Turing machine which scans the transition matrices and the final weight vectors of A_1 and A_2, and then writes down the entries of the transition matrices and the final weight vector of their product $A_1 \times A_2$ onto the output tape. This computation requires maintaining only a constant number of counters to store the indices of transition matrices, which takes logarithmic space in the representation of automata A_1 and A_2. Hence, the Turing machine computing $A_1 \times A_2$ uses logarithmic space in the work tape. ■

A tree series f is called *recognisable* if it is recognised by some multiplicity tree automaton; such an automaton is called an *MTA-representation* of f. An MTA-representation of f that has the smallest dimension is called *minimal.* The set of all recognisable tree series in $\mathbb{F}\langle\langle T_\Sigma \rangle\rangle$ is denoted by $\mathrm{Rec}(\Sigma, \mathbb{F})$.

The following result was first shown by Bozapalidis and Louscou-Bozapalidou (1983); an essentially equivalent result was later shown by Habrard and Oncina (2006).

Theorem 3 (Bozapalidis and Louscou-Bozapalidou, 1983) *Let Σ be a ranked alphabet and $\mathbb{F}$ be a field. Let $f \in \mathbb{F}\langle\langle T_\Sigma \rangle\rangle$ and let H be the Hankel matrix of f. It holds that $f \in Rec(\Sigma, \mathbb{F})$ if and only if H has finite rank over $\mathbb{F}$. In case $f \in Rec(\Sigma, \mathbb{F})$, the dimension of a minimal MTA-representation of f equals the rank of H.*

The following result by Bozapalidis and Alexandrakis (1989, Proposition 4) states that for any recognisable tree series, its minimal MTA-representation is unique up to change of basis.

Theorem 4 (Bozapalidis and Alexandrakis, 1989) *Let Σ be a ranked alphabet and $\mathbb{F}$ be a field. Let $f \in Rec(\Sigma, \mathbb{F})$ and let r be the rank (over $\mathbb{F}$) of the Hankel matrix of f. Let $A_1 = (r, \Sigma, \mu_1, \gamma_1)$ be an MTA-representation of f. Given an $\mathbb{F}$-multiplicity tree automaton $A_2 = (r, \Sigma, \mu_2, \gamma_2)$, it holds that A_2 recognises f if and only if there exists an invertible matrix $U \in \mathbb{F}^{r \times r}$ such that $\gamma_2 = U \cdot \gamma_1$ and $\mu_2(\sigma) = U^{\otimes rk(\sigma)} \cdot \mu_1(\sigma) \cdot U^{-1}$ for every $\sigma \in \Sigma$.*

2.4 DAG Representations of Finite Trees

A *directed multigraph* consists of a set of nodes V and a multiset of directed edges $E \subseteq V \times V$. We say that a directed multigraph is *acyclic* if the underlying directed graph has no cycles; we say it is *ordered* if a linear order on the successors of each node is assumed. A directed multigraph is *rooted* if there is a distinguished *root* node v such that all other nodes are reachable from v.

Let Σ be a ranked alphabet. A *DAG representation of a Σ-tree* (Σ-*DAG* or *DAG* for short) is a rooted acyclic ordered directed multigraph whose nodes are labelled with symbols from Σ such that the outdegree of each node is equal to the rank of the symbol it is labelled with. Formally a Σ-DAG consists of a set of nodes V, for each node $v \in V$ a list of successors $succ(v) \in V^*$, and a node labelling $\lambda : V \to \Sigma$ where for each node $v \in V$ it holds that $\lambda(v) \in \Sigma_{|succ(v)|}$. Note that Σ-trees are a subclass of Σ-DAGs.

Let G be a Σ-DAG. The *size* of G, denoted by $size(G)$, is the number of nodes in G. The *height* of G, denoted by $height(G)$, is the length of a longest directed path in G. For any node v in G, the *sub-DAG* of G *rooted at* v, denoted by $G|_v$, is the Σ-DAG consisting of the node v and all of its descendants in G. Clearly, if v is the root of G then $G|_v = G$. The set $\{G|_v : v \text{ is a node in } G\}$ of all the sub-DAGs of G is denoted by $\mathit{Sub}(G)$.

For any Σ-DAG G, we define its *unfolding* into a Σ-tree, denoted by $\mathit{unfold}(G)$, inductively as follows: If the root of G is labelled with a symbol σ and has the list of successors $v_1, \dots, v_k$, then

$$\mathit{unfold}(G) = \sigma(\mathit{unfold}(G|_{v_1}), \dots, \mathit{unfold}(G|_{v_k})).$$

The next proposition follows easily from the definition.

Proposition 5 *If G is a Σ-DAG, then $\mathit{Sub}(\mathit{unfold}(G)) = \mathit{unfold}[\mathit{Sub}(G)]$.*

Because a context has exactly one occurrence of symbol $\Box$, any *DAG representation of a Σ-context* is a $(\{\Box\} \cup \Sigma)$-DAG that has a unique path from the root to the (unique) $\Box$-labelled node. The *concatenation* of a DAG K, representing a Σ-context, and a Σ-DAG G is the Σ-DAG, denoted by $K[G]$, obtained by substituting the root of G for $\Box$ in K.

Proposition 6 *Let K be a DAG representation of a Σ-context, and let G be a Σ-DAG. Then, $\mathit{unfold}(K[G]) = \mathit{unfold}(K)[\mathit{unfold}(G)]$.*

Proof The proof is by induction on $\mathit{height}(K)$. For the base case, let $\mathit{height}(K) = 0$. Then, we have that $K = \Box$ and therefore $\mathit{unfold}(\Box[G]) = \mathit{unfold}(G) = \mathit{unfold}(\Box)[\mathit{unfold}(G)]$ for any Σ-DAG G.

For the induction step, let $h \in \mathbb{N}_0$ and assume that the result holds if $\mathit{height}(K) \leq h$. Let K be a DAG representation of a Σ-context such that $\mathit{height}(K) = h + 1$. Let the root of K have label σ and list of successors $v_1, \dots, v_k$. By definition, there is a unique path in K going from the root to the $\Box$-labelled node. Without loss of generality, we can assume that the $\Box$-labelled node is a successor of v_1. Take an arbitrary Σ-DAG G. Since $\mathit{height}(K|_{v_1}) \leq h$, we have by the induction hypothesis that

$$\begin{aligned}\mathit{unfold}(K[G]) &= \sigma(\mathit{unfold}(K|_{v_1}[G]), \mathit{unfold}(K|_{v_2}), \dots, \mathit{unfold}(K|_{v_k}))\\ &= \sigma(\mathit{unfold}(K|_{v_1})[\mathit{unfold}(G)], \mathit{unfold}(K|_{v_2}), \dots, \mathit{unfold}(K|_{v_k}))\\ &= \sigma(\mathit{unfold}(K|_{v_1}), \mathit{unfold}(K|_{v_2}), \dots, \mathit{unfold}(K|_{v_k}))[\mathit{unfold}(G)]\\ &= \mathit{unfold}(K)[\mathit{unfold}(G)].\end{aligned}$$

This completes the proof by induction. ■

2.5 Multiplicity Tree Automata on DAGs

In this section, we introduce the notion of a multiplicity tree automaton running on DAGs. To the best of our knowledge, this notion has not been studied before.

Let $\mathbb{F}$ be a field, and $A = (n, \Sigma, \mu, \gamma)$ be an $\mathbb{F}$-multiplicity tree automaton. The computation of automaton A on a Σ-DAG $G = (V, E)$ is defined as follows: A *run* of A on G is a mapping $\rho : \mathit{Sub}(G) \to \mathbb{F}^n$ such that for every node $v \in V$, if v is labelled with σ and has the list of successors $\mathit{succ}(v) = v_1, \dots, v_k$ then

$$\rho(G|_v) = (\rho(G|_{v_1}) \otimes \cdots \otimes \rho(G|_{v_k})) \cdot \mu(\sigma).$$

Automaton A assigns to G a *weight* $\|A\|(G) \in \mathbb{F}$ where $\|A\|(G) = \rho(G) \cdot \gamma$.

In the following proposition, we show that the weight assigned by a multiplicity tree automaton to a DAG is equal to the weight assigned to its tree unfolding.

Proposition 7 *Let $\mathbb{F}$ be a field, and $A = (n, \Sigma, \mu, \gamma)$ be an $\mathbb{F}$-multiplicity tree automaton. For any Σ-DAG G, it holds that $\rho(G) = \mu(\mathit{unfold}(G))$ and $\|A\|(G) = \|A\|(\mathit{unfold}(G))$.*

Proof Let V be the set of nodes of G. First we show that for every $v \in V$,

$$\rho(G|_v) = \mu(\mathit{unfold}(G|_v)). \tag{6}$$

The proof is by induction on $\mathit{height}(G|_v)$. For the base case, let $\mathit{height}(G|_v) = 0$. This implies that $G|_v = \sigma \in \Sigma_0$. Therefore, by definition we have that

$$\rho(G|_v) = \mu(\sigma) = \mu(\mathit{unfold}(\sigma)) = \mu(\mathit{unfold}(G|_v)).$$

For the induction step, let $h \in \mathbb{N}_0$ and assume that Equation (6) holds for every $v \in V$ such that $\mathit{height}(G|_v) \leq h$. Take any $v \in V$ such that $\mathit{height}(G|_v) = h + 1$. Let the root of $G|_v$ be labelled with a symbol σ and have list of successors $\mathit{succ}(v) = v_1, \ldots, v_k$. Then for every $j \in [k]$, we have that $\mathit{height}(G|_{v_j}) \leq h$ and thus $\rho(G|_{v_j}) = \mu(\mathit{unfold}(G|_{v_j}))$ holds by the induction hypothesis. This implies that

$$\begin{aligned}
\rho(G|_v) &= (\rho(G|_{v_1}) \otimes \cdots \otimes \rho(G|_{v_k})) \cdot \mu(\sigma) \\
&= (\mu(\mathit{unfold}(G|_{v_1})) \otimes \cdots \otimes \mu(\mathit{unfold}(G|_{v_k}))) \cdot \mu(\sigma) \\
&= \mu(\sigma(\mathit{unfold}(G|_{v_1}), \ldots, \mathit{unfold}(G|_{v_k}))) \\
&= \mu(\mathit{unfold}(G|_v)),
\end{aligned}$$

which completes the proof of Equation (6) for all $v \in V$ by induction.

Taking v to be the root of G, we get from Equation (6) that $\rho(G) = \mu(\mathit{unfold}(G))$. Therefore, $\|A\|(G) = \rho(G) \cdot \gamma = \mu(\mathit{unfold}(G)) \cdot \gamma = \|A\|(\mathit{unfold}(G))$. ■

Example 3 *Let $\Sigma = \{\sigma_0, \sigma_2\}$ be a ranked alphabet such that $rk(\sigma_0) = 0$ and $rk(\sigma_2) = 2$. Take any $n \in \mathbb{N}$. Let t_n, depicted in Figure 1, be the perfect binary Σ-tree of height $n - 1$. Note that $\mathit{size}(t_n) = O(2^n)$. Define an $\mathbb{F}$-MTA $A = (n, \Sigma, \mu, e_1)$ such that $\mu(\sigma_0) = e_n \in \mathbb{F}^{1 \times n}$ and $\mu(\sigma_2) \in \mathbb{F}^{n^2 \times n}$ where $\mu(\sigma_2)_{(i+1,i+1),i} = 1$ for every $i \in [n-1]$, and all other entries of $\mu(\sigma_2)$ are zero. It is easy to see that $\|A\|(t_n) = 1$ and $\|A\|(t) = 0$ for every $t \in T_\Sigma \setminus \{t_n\}$.*

Let B be the 0-dimensional $\mathbb{F}$-MTA over Σ (so that $\|B\| \equiv 0$). Suppose we were to check whether automata A and B are equivalent. Then the only counterexample to their equivalence, namely the tree t_n, has size $O(2^n)$. Note, however, that t_n has an exponentially more succinct DAG representation G_n, given in Figure 2.

2.6 Arithmetic Circuits

An *arithmetic circuit* is a finite acyclic vertex-labelled directed multigraph whose vertices, called *gates*, have indegree 0 or 2. Vertices of indegree 0 are called *input gates* and are labelled with a constant 0 or 1, or a variable from the set $\{x_i : i \in \mathbb{N}\}$. Vertices of indegree

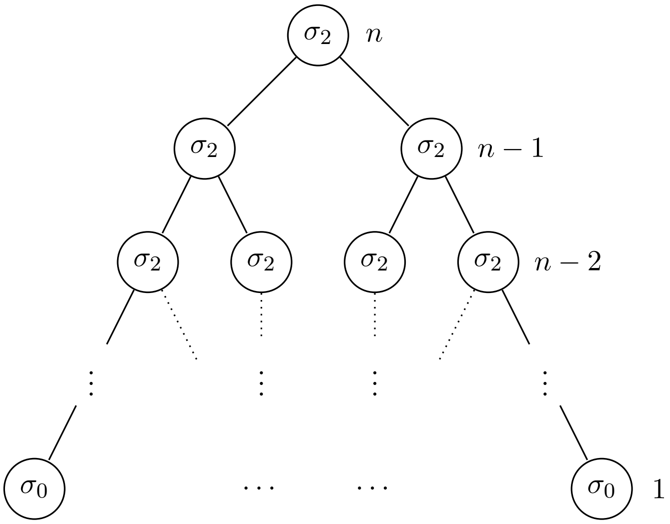


Figure 1: Tree t_n

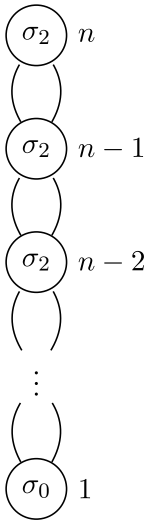


Figure 2: DAG G_n

2 are called *internal gates* and are labelled with an arithmetic operation $+$, $\times$, or $-$. We assume that there is a unique gate with outdegree 0 called the *output gate*. An arithmetic circuit is called *variable-free* if all input gates are labelled with 0 or 1.

Given two gates u and v of an arithmetic circuit C, we call u a *child* of v if (u, v) is a directed edge in C. The *size* of C is the number of gates in C. The *height* of a gate v in C, written as $\mathit{height}(v)$, is the length of a longest directed path from an input gate to v. The *height* of C is the maximal height of a gate in C.

An arithmetic circuit C computes a polynomial over the integers as follows: An input gate of C labelled with $\alpha \in \{0, 1\} \cup \{x_i : i \in \mathbb{N}\}$ computes the polynomial α. An internal gate of C labelled with $* \in \{+, \times, -\}$ computes the polynomial $p_1 * p_2$ where p_1 and p_2 are the polynomials computed by its children. For any gate v in C, we write f_v for the polynomial computed by v. The *output* of C, written as f_C, is the polynomial computed by the output gate of C. The *arithmetic circuit identity testing (**ACIT**)* problem asks whether the output of a given arithmetic circuit is equal to the zero polynomial.

Remark 8 *Any variable-free arithmetic circuit C can be seen as a Σ-DAG with the ranked alphabet $\Sigma = \{0, 1, +, \times, -\}$ where $0, 1$ are nullary symbols and $+, \times, -$ are binary symbols. Let $A = (2, \Sigma, \mu, \gamma)$ be the multiplicity tree automaton from Example 1. Then, for any gate v in C it holds that $\mu(C|_v) = \begin{bmatrix} 1 & f_v \end{bmatrix}$, where $C|_v$ is the sub-DAG of C rooted at v and f_v is the number computed at gate v. (This result can be easily proved using induction on $\mathit{height}(C|_v)$.) In particular, when v is the output gate of C we get that*

$$\|A\|(C) = \mu(C) \cdot \gamma = \begin{bmatrix} 1 & f_C \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \end{bmatrix}^\top = f_C.$$

Hence, automaton A evaluates the circuit C.

2.7 The Learning Model

In this paper we work with the *exact learning model* of Angluin (1988): Let f be a *target function.* A *Learner* (*learning algorithm*) may, in each step, propose a hypothesis function h by making an *equivalence query* to a *Teacher.* If h is equivalent to f, then the Teacher returns YES and the Learner succeeds and halts. Otherwise, the Teacher returns NO with a *counterexample*, which is an assignment x such that $h(x) \neq f(x)$. Moreover, the Learner may query the Teacher for the value of the function f on a particular assignment x by making a *membership query* on x. The Teacher returns the value $f(x)$ to such a query.

We say that a class of functions $\mathscr{C}$ is *exactly learnable* if there is a Learner that for any target function $f \in \mathscr{C}$, outputs a hypothesis $h \in \mathscr{C}$ such that $h(x) = f(x)$ for all assignments x, and does so in time polynomial in the size of a shortest representation of f and the size of a largest counterexample returned by the Teacher. We moreover say that the class $\mathscr{C}$ is *exactly learnable in (randomised) polynomial time* if the learning algorithm can be implemented to run in (randomised) polynomial time in the Turing model.

3. Equivalence Queries

In the exact learning model, one of the principal algorithmic questions from the point of view of the Teacher is the computational complexity of equivalence testing. In this section we characterise the computational complexity of equivalence testing for multiplicity tree automata, showing that this problem is logspace equivalent to polynomial identity testing. The latter is a well-studied problem for which there are numerous randomised polynomial-time algorithms, with the existence of a deterministic polynomial-time algorithm being a longstanding open problem. Moreover in this section, we explain why it is natural to expect the Teacher to return succinct DAG counterexamples in the case of inequivalence.

3.1 Computational Complexity of MTA Equivalence

A key algorithmic component of the exact learning framework is checking the equivalence of the hypothesis and the target function: a task for the Teacher rather than the Learner. The existence of efficient algorithms to perform such equivalence checks is important for several applications of the exact learning framework (see, e.g., Feng et al., 2011). With this in mind, in this subsection we characterise the computational complexity of the equivalence problem for $\mathbb{Q}$-multiplicity tree automata. Here we specialise the weight field to be $\mathbb{Q}$ since we want to work within the classical Turing model of computation. Parts of this section also exploit the fact that $\mathbb{Q}$ is an ordered field.

Our main result is:

Theorem 9 *The equivalence problem for $\mathbb{Q}$-multiplicity tree automata is logspace interreducible with **ACIT**.*

A related result, characterising equivalence of probabilistic visibly pushdown automata on words in terms of polynomial identity testing, was shown by Kiefer et al. (2013). On several occasions in this section, we will implicitly make use of the fact that a composition of two logspace reductions is again a logspace reduction (Arora and Barak, 2009, Lemma 4.17).

3.1.1 From MTA Equivalence to ACIT

First, we present a logspace reduction from the equivalence problem for $\mathbb{Q}$-MTAs to **ACIT**. We start with the following lemma.

In unary!

Lemma 10 *Given an integer $n \in \mathbb{N}$ and a $\mathbb{Q}$-multiplicity tree automaton A over a ranked alphabet Σ, one can compute, in logarithmic space in $|A|$ and n, a variable-free arithmetic circuit that has output $\sum_{t \in T_\Sigma^{<n}} \|A\|(t)$.*

Proof Let $A = (r, \Sigma, \mu, \gamma)$, and let m be the rank of Σ. By definition, it holds that

$$\sum_{t \in T_\Sigma^{<n}} \|A\|(t) = \left(\sum_{t \in T_\Sigma^{<n}} \mu(t) \right) \cdot \gamma. \tag{7}$$

We have $\sum_{t \in T_\Sigma^{<1}} \mu(t) = \sum_{\sigma \in \Sigma_0} \mu(\sigma)$. Furthermore for every $i \in \mathbb{N}$, it holds that

$$T_\Sigma^{<i+1} = \{\sigma(t_1, \ldots, t_k) : k \in \{0, \ldots, m\}, \sigma \in \Sigma_k, t_1, \ldots, t_k \in T_\Sigma^{<i}\}$$

and thus by bilinearity of Kronecker product,

$$\begin{aligned}
\sum_{t \in T_\Sigma^{<i+1}} \mu(t) &= \sum_{k=0}^{m} \sum_{\sigma \in \Sigma_k} \sum_{t_1 \in T_\Sigma^{<i}} \cdots \sum_{t_k \in T_\Sigma^{<i}} (\mu(t_1) \otimes \cdots \otimes \mu(t_k)) \cdot \mu(\sigma) \\
&= \sum_{k=0}^{m} \sum_{\sigma \in \Sigma_k} \left(\left(\sum_{t_1 \in T_\Sigma^{<i}} \mu(t_1) \right) \otimes \cdots \otimes \left(\sum_{t_k \in T_\Sigma^{<i}} \mu(t_k) \right) \right) \cdot \mu(\sigma) \\
&= \sum_{k=0}^{m} \left(\sum_{t \in T_\Sigma^{<i}} \mu(t) \right)^{\otimes k} \sum_{\sigma \in \Sigma_k} \mu(\sigma).
\end{aligned} \tag{8}$$

In the following we define a variable-free arithmetic circuit Φ that has output $\sum_{t \in T_\Sigma^{<n}} \|A\|(t)$. First, let us denote $G(i) := \sum_{t \in T_\Sigma^{<i}} \mu(t)$ for every $i \in \mathbb{N}$. Then by Equation (8) we have $G(i+1) = \sum_{k=0}^{m} G(i)^{\otimes k} \cdot S(k)$ where $S(k) := \sum_{\sigma \in \Sigma_k} \mu(\sigma)$ for every $k \in \{0, \ldots, m\}$. In coordinate notation, for every $j \in [r]$ we have by Equation (1) that

$$G(i+1)_j = \sum_{k=0}^{m} \sum_{(l_1, \ldots, l_k) \in [r]^k} \prod_{a=1}^{k} G(i)_{l_a} \cdot S(k)_{(l_1, \ldots, l_k), j}. \tag{9}$$

We present Φ as a straight-line program, with built-in constants

$$\{\mu^\sigma_{(l_1, \ldots, l_k), j}, \gamma_j : k \in \{0, \ldots, m\}, \sigma \in \Sigma_k, (l_1, \ldots, l_k) \in [r]^k, j \in [r]\}$$

representing the entries of the transition matrices and the final weight vector of A, internal variables $\{s^k_{(l_1, \ldots, l_k), j} : k \in \{0, \ldots, m\}, (l_1, \ldots, l_k) \in [r]^k, j \in [r]\}$ and $\{g_{i,j} : i \in [n], j \in [r]\}$ evaluating the entries of matrices $S(k)$ and vectors $G(i)$ respectively, and the final internal variable f computing the value of Φ.

```
1. For $j \in [r]$ do $g_{1,j} \leftarrow \sum_{\sigma \in \Sigma_0} \mu^{\sigma}_{j}$
2. For $k \in \{0, \ldots, m\}$, $(l_1, \ldots, l_k) \in [r]^k$, $j \in [r]$ do $s^{k}_{(l_1,\ldots,l_k),j} \leftarrow \sum_{\sigma \in \Sigma_k} \mu^{\sigma}_{(l_1,\ldots,l_k),j}$
3. For $i = 1$ to $n - 1$ do
    3.1. For $k \in \{0, \ldots, m\}$, $(l_1, \ldots, l_k) \in [r]^k$, $j \in [r]$ do
             $h^{i,k}_{(l_1,\ldots,l_k),j} \leftarrow \prod_{a=1}^{k} g_{i,l_a} \cdot s^{k}_{(l_1,\ldots,l_k),j}$
    3.2. For $j \in [r]$ do
             $g_{i+1,j} \leftarrow \sum_{k=0}^{m} \sum_{(l_1,\ldots,l_k) \in [r]^k} h^{i,k}_{(l_1,\ldots,l_k),j}$
4. For $j \in [r]$ do $f_j \leftarrow g_{n,j} \cdot \gamma_j$
5. $f \leftarrow \sum_{j \in [r]} f_j$.
```

Table 1: Straight-line program Φ

Formally, the straight-line program Φ is given in Table 1. Here the statements are given in indexed-sum and indexed-product notation, which can easily be expanded in terms of the corresponding binary operations. It follows from Equations (7) and (9) that the output of Φ is $G(n) \cdot \gamma = \sum_{t \in T_{\Sigma}^{<n}} \|A\|(t)$.

The input gates of Φ are labelled with rational numbers. By separately encoding numerators and denominators, we can in logarithmic space reduce Φ to an arithmetic circuit where all input gates are labelled with integers. Moreover, without loss of generality we can assume that every input gate of Φ is labelled with 0 or 1. Any other integer label given in binary can be encoded as an arithmetic circuit.

Recalling that a composition of two logspace reductions is again a logspace reduction, we conclude that the entire computation takes logarithmic space in $|A|$ and n. ■

Before presenting the reduction in Proposition 12, we recall the following characterisation (Seidl, 1990, Theorem 4.2) of equivalence of two multiplicity tree automata over an arbitrary field.

Proposition 11 (Seidl, 1990) *Suppose A and B are multiplicity tree automata of dimension n_1 and n_2, respectively, and over a ranked alphabet Σ. Then, A and B are equivalent if and only if $\|A\|(t) = \|B\|(t)$ for every $t \in T_\Sigma^{<n_1+n_2}$.*

We now turn to the reduction: SQUARING TRICK

Proposition 12 *The equivalence problem for $\mathbb{Q}$-multiplicity tree automata is logspace reducible to **ACIT**.*

Proof Let A and B be $\mathbb{Q}$-multiplicity tree automata over a ranked alphabet Σ, and let n be the sum of their dimensions. Proposition 2 implies that

$$\begin{aligned}\sum_{t\in T_\Sigma^{<n}} (\|A\|(t) - \|B\|(t))^2 &= \sum_{t\in T_\Sigma^{<n}} \left(\|A\|(t)^2 + \|B\|(t)^2 - 2\|A\|(t)\|B\|(t)\right) \\ &= \sum_{t\in T_\Sigma^{<n}} \left(\|A\times A\|(t) + \|B\times B\|(t) - 2\|A\times B\|(t)\right).\end{aligned}$$

Thus by Proposition 11, automata A and B are equivalent if and only if

$$\sum_{t\in T_\Sigma^{<n}} \|A\times A\|(t) + \sum_{t\in T_\Sigma^{<n}} \|B\times B\|(t) - 2\sum_{t\in T_\Sigma^{<n}} \|A\times B\|(t) = 0. \tag{10}$$

We know from Proposition 2 that automata $A\times A$, $B\times B$, and $A\times B$ can be computed in logarithmic space. Thus by Lemma 10 one can compute, in logarithmic space in $|A|$ and $|B|$, variable-free arithmetic circuits that have outputs $\sum_{t\in T_\Sigma^{<n}} \|A\times A\|(t)$, $\sum_{t\in T_\Sigma^{<n}} \|B\times B\|(t)$, and $\sum_{t\in T_\Sigma^{<n}} \|A\times B\|(t)$ respectively. Using Equation (10), we can now easily construct a variable-free arithmetic circuit that has output 0 if and only if A and B are equivalent. ■

3.1.2 From **ACIT** to MTA Equivalence

We now present a converse reduction: from **ACIT** to the equivalence problem for $\mathbb{Q}$-MTAs.

Allender et al. (2009, Proposition 2.2) give a logspace reduction of the general **ACIT** problem to the special case of **ACIT** for variable-free circuits. The latter can, by representing arbitrary integers as differences of two nonnegative integers, be reformulated as the problem of deciding whether two variable-free arithmetic circuits with only $+$ and $\times$-internal gates compute the same number. With this result at hand, we turn to the reduction:

Proposition 13 ***ACIT** is logspace reducible to the equivalence problem for $\mathbb{Q}$-multiplicity tree automata.*

Proof Let C_1 and C_2 be two variable-free arithmetic circuits whose internal gates are labelled with $+$ or $\times$. By padding with extra gates, without loss of generality we can assume that in each circuit the children of a height-i gate both have height $i-1$, $+$-gates have even height, $\times$-gates have odd height, and the output gate has an even height h.

In the following we define two $\mathbb{Q}$-MTAs, A_1 and A_2, that are equivalent if and only if circuits C_1 and C_2 have the same output. Automata A_1 and A_2 are both defined over a ranked alphabet $\Sigma = \{\sigma_0, \sigma_1, \sigma_2\}$ where σ_0 is a nullary, σ_1 is a unary, and σ_2 is a binary symbol. Intuitively, automata A_1 and A_2 both recognise the common 'tree unfolding' of circuits C_1 and C_2.

We now derive A_1 from C_1; A_2 is analogously derived from C_2. Let $\{v_1, \ldots, v_r\}$ be the set of gates of C_1 where v_r is the output gate. Automaton A_1 has a state q_i for every gate v_i of C_1. Formally, $A_1 = (r, \Sigma, \mu, e_r^\top)$ where for every $i \in [r]$:

- If v_i is an input gate with label 1 then $\mu(\sigma_0)_i = 1$, otherwise $\mu(\sigma_0)_i = 0$.
- If v_i is a $+$-gate with children v_{j_1} and v_{j_2} then $\mu(\sigma_1)_{j_1,i} = \mu(\sigma_1)_{j_2,i} = 1$ if $j_1 \neq j_2$, $\mu(\sigma_1)_{j_1,i} = 2$ if $j_1 = j_2$, and $\mu(\sigma_1)_{l,i} = 0$ for every $l \notin \{j_1, j_2\}$. If v_i is an input gate or a $\times$-gate then $\mu(\sigma_1)^i = 0_{r\times 1}$.
- If v_i is a $\times$-gate with children v_{j_1} and v_{j_2} then $\mu(\sigma_2)_{(j_1,j_2),i} = 1$, and $\mu(\sigma_2)_{(l_1,l_2),i} = 0$ for every $(l_1, l_2) \neq (j_1, j_2)$. If v_i is an input gate or a $+$-gate then $\mu(\sigma_2)^i = 0_{r^2\times 1}$.

We define a sequence of trees $(t_n)_{n\in\mathbb{N}_0} \subseteq T_\Sigma$ by $t_0 = \sigma_0$, $t_{n+1} = \sigma_1(t_n)$ for n odd, and $t_{n+1} = \sigma_2(t_n, t_n)$ for n even. In the following we show that $\|A_1\|(t_h) = f_{C_1}$. For every gate v of C_1, by assumption it holds that all paths from v to the output gate have equal length. We now prove that for every $i \in [r]$,

$$\mu(t_{h_i})_i = f_{v_i} \tag{11}$$

where $h_i := \mathit{height}(v_i)$. The proof uses induction on $h_i \in \{0, \ldots, h\}$. For the base case, let $h_i = 0$. Then, v_i is an input gate and thus by definition of automaton A_1 we have

$$\mu(t_{h_i})_i = \mu(t_0)_i = \mu(\sigma_0)_i = f_{v_i}.$$

For the induction step, let $n \in [h]$ and assume that Equation (11) holds for every gate v_i of height less than n. Take an arbitrary gate v_i of C_1 such that $h_i = n$. Let gates v_{j_1} and v_{j_2} be the children of v_i. Then $h_{j_1} = h_{j_2} = h_i - 1 = n - 1$ by assumption. The induction hypothesis now implies that $\mu(t_{h_i-1})_{j_1} = f_{v_{j_1}}$ and $\mu(t_{h_i-1})_{j_2} = f_{v_{j_2}}$. Depending on the label of v_i, there are two possible cases as follows:

(i) If v_i is a $+$-gate, then h_i is even and thus by definition of A_1 we have

$$\begin{aligned}\mu(t_{h_i})_i &= \mu(\sigma_1(t_{h_i-1}))_i = \mu(t_{h_i-1}) \cdot \mu(\sigma_1)^i \\ &= \mu(t_{h_i-1})_{j_1} + \mu(t_{h_i-1})_{j_2} = f_{v_{j_1}} + f_{v_{j_2}} = f_{v_i}.\end{aligned}$$

(ii) If v_i is a $\times$-gate, then h_i is odd and thus by definition of A_1 and Equation (1) we have

$$\begin{aligned}\mu(t_{h_i})_i &= \mu(\sigma_2(t_{h_i-1}, t_{h_i-1}))_i = \mu(t_{h_i-1})^{\otimes 2} \cdot \mu(\sigma_2)^i \\ &= \mu(t_{h_i-1})_{j_1} \cdot \mu(t_{h_i-1})_{j_2} = f_{v_{j_1}} \cdot f_{v_{j_2}} = f_{v_i}.\end{aligned}$$

This completes the proof of Equation (11) by induction. Now for the output gate v_r of C_1, we get from Equation (11) that $\mu(t_h)_r = f_{v_r}$ since $h_r = h$. Therefore,

$$\|A_1\|(t_h) = \mu(t_h) \cdot e_r^\top = \mu(t_h)_r = f_{v_r} = f_{C_1}.$$

Analogously, it holds that $\|A_2\|(t_h) = f_{C_2}$. It is moreover clear by construction that $\|A_1\|(t) = 0$ and $\|A_2\|(t) = 0$ for every $t \in T_\Sigma \setminus \{t_h\}$. Therefore, automata A_1 and A_2 are equivalent if and only if arithmetic circuits C_1 and C_2 have the same output. ■

Propositions 12 and 13 together imply Theorem 9. On a positive note, it should be remarked that there are numerous efficient randomised algorithms for **ACIT**. Indeed, it was already known that there is a randomised polynomial-time algorithm for equivalence of multiplicity tree automata (Seidl, 1990). On the other hand, we have shown that obtaining a deterministic polynomial-time algorithm for multiplicity tree automaton equivalence would imply also a deterministic polynomial-time algorithm for **ACIT**.

3.2 DAG Counterexamples

In the exact learning model, when answering an equivalence query the Teacher not only checks equivalence but also provides a counterexample in case of inequivalence. As mentioned before, there is a randomised polynomial-time algorithm for checking MTA equivalence (Seidl, 1990). In this subsection, we explain why a Teacher using this algorithm would naturally give succinct DAG counterexamples.

Although the paper of Seidl (1990) does not mention counterexamples, they can be easily extracted from the algorithm presented therein. Indeed the correctness proof of the algorithm shows, *inter alia*, that for any two inequivalent MTAs $A_1 = (n_1, \Sigma, \mu_1, \gamma_1)$ and $A_2 = (n_2, \Sigma, \mu_2, \gamma_2)$, there exists a tree t such that $\|A_1\|(t) \neq \|A_2\|(t)$ and t can be represented by a DAG with at most $n_1 + n_2$ vertices. To see this, we now briefly describe the main idea behind the procedure: Given MTAs A_1 and A_2 as above, a prefix-closed set of trees $S \subseteq T_\Sigma$ is maintained such that $\{\begin{bmatrix}\mu_1(t) & \mu_2(t)\end{bmatrix} : t \in S\}$ is a linearly independent set of vectors. Note that since this set of vectors lies in $\mathbb{F}^{n_1+n_2}$, it necessarily holds that $|S| \leq n_1 + n_2$. The algorithm terminates when

$$\mathit{span}\left\{\begin{bmatrix}\mu_1(t) & \mu_2(t)\end{bmatrix} : t \in S\right\} = \mathit{span}\left\{\begin{bmatrix}\mu_1(t) & \mu_2(t)\end{bmatrix} : t \in T_\Sigma\right\}$$

and reports that A_1 and A_2 are inequivalent just in case a tree $t \in S$ is found such that

$$\begin{bmatrix}\mu_1(t) & \mu_2(t)\end{bmatrix} \cdot \begin{bmatrix}\gamma_1 \\ -\gamma_2\end{bmatrix} \neq 0,$$

i.e., $\|A_1\|(t) \neq \|A_2\|(t)$. Such a tree t, if one exists, has at most $n_1 + n_2$ subtrees and thus has a DAG representation of size at most $n_1 + n_2$. As we have seen in Example 3, the number of vertices of tree t may be exponential in $n_1 + n_2$, thus it is very natural that a Teacher that resolves equivalence queries using the algorithm of Seidl (1990) would return counterexamples represented succinctly as DAGs.

4. The Learning Algorithm

In this section, we give an exact learning algorithm for multiplicity tree automata. Our algorithm is polynomial in the size of a minimal automaton equivalent to the target and the size of a largest counterexample given as a DAG. As seen in Example 3, DAG counterexamples can be exponentially more succinct than tree counterexamples. Therefore, achieving a polynomial bound in the context of DAG representations is a more exacting criterion.

Over an arbitrary field $\mathbb{F}$, the algorithm can be seen as running on a Blum-Shub-Smale machine that can write and read field elements to and from its memory at unit cost and that can also perform arithmetic operations and equality tests on field elements at unit cost (see Arora and Barak, 2009). Over $\mathbb{Q}$, the algorithm can be implemented in randomised polynomial time by representing rationals as arithmetic circuits and using a coRP algorithm for equality testing of such circuits (see Allender et al., 2009).

This section is organised as follows: In Section 4.1 we present the learning algorithm. In Section 4.2 we prove correctness on trees, and then argue in Section 4.3 that the algorithm can be faithfully implemented using a DAG representation of trees. Finally, in Section 4.4 we give a complexity analysis of the algorithm assuming the DAG representation.

4.1 The Algorithm

Let $f \in \text{Rec}(\Sigma, \mathbb{F})$ be the target function. The algorithm learns an MTA-representation of f using its Hankel matrix H, which has finite rank over $\mathbb{F}$ by Theorem 3.

The algorithm iteratively constructs a full row-rank submatrix of the Hankel matrix H. At each stage, the algorithm maintains the following data:

- An integer $n \in \mathbb{N}$.
- A set of n 'rows' $X = \{t_1, \ldots, t_n\} \subseteq T_\Sigma$.
- A finite set of 'columns' $Y \subseteq C_\Sigma$ such that $\Box \in Y$.
- A submatrix $H_{X,Y}$ of H that has full row rank.

These data determine a *hypothesis automaton* A of dimension n, whose states correspond to the rows of $H_{X,Y}$, with the i^{th} row corresponding to the state reached after reading tree t_i. The Learner makes an equivalence query on the hypothesis A. In case the Teacher answers NO, the Learner receives a counterexample z. The Learner then parses z bottom-up to find a minimal subtree of z that is also a counterexample, and uses this subtree to augment the row set X and the column set Y in a way that increases the rank of the submatrix $H_{X,Y}$.

Formally, the algorithm LMTA is given in Table 2. Here for any k-ary symbol $\sigma \in \Sigma$ we define $\sigma(X, \ldots, X) := \{\sigma(t_{i_1}, \ldots, t_{i_k}) : (i_1, \ldots, i_k) \in [n]^k\}$.

Algorithm LMTA follows a classical scheme: it generalises the procedure of Beimel et al. (2000) by working with a more general notion of a Hankel matrix that is appropriate for tree series. Moreover, LMTA differs from the procedure of Habrard and Oncina (2006) in the way counterexamples are treated and the hypothesis automaton updated; we provide more details on this point at the end of this section.

4.2 Correctness Proof

In this subsection, we prove the correctness of the exact learning algorithm LMTA. Specifically, we show that, given a target $f \in \text{Rec}(\Sigma, \mathbb{F})$, algorithm LMTA outputs a minimal MTA-representation of f after at most $rank(H)$ iterations of the main loop.

The correctness proof naturally breaks down into several lemmas. First, we show that matrix $H_{X,Y}$ has full row rank.

```
Algorithm LMTA
Target: $f \in \mathrm{Rec}(\Sigma, \mathbb{F})$, where $\Sigma$ has rank $m$ and $\mathbb{F}$ is a field

  1. Make an equivalence query on the 0-dimensional $\mathbb{F}$-MTA over $\Sigma$.
     If the answer is YES then output the 0-dimensional $\mathbb{F}$-MTA over $\Sigma$ and halt.
     Otherwise the answer is NO and $z$ is a counterexample. Initialise:
     $n \leftarrow 1$, $t_n \leftarrow z$, $X \leftarrow \{t_n\}$, $Y \leftarrow \{\Box\}$.

  2. 2.1. For every $k \in \{0, \dots, m\}$, $\sigma \in \Sigma_k$, and $(i_1, \dots, i_k) \in [n]^k$:
          If $H_{\sigma(t_{i_1}, \dots, t_{i_k}), Y}$ is not a linear combination of $H_{t_1,Y}, \dots, H_{t_n,Y}$ then
          $n \leftarrow n+1$, $t_n \leftarrow \sigma(t_{i_1}, \dots, t_{i_k})$, $X \leftarrow X \cup \{t_n\}$.
     2.2. Define an $\mathbb{F}$-MTA $A = (n, \Sigma, \mu, \gamma)$ as follows:
          • $\gamma = H_{X,\Box}$.
          • For every $k \in \{0, \dots, m\}$ and $\sigma \in \Sigma_k$:
            Define matrix $\mu(\sigma) \in \mathbb{F}^{n^k \times n}$ by the equation

                    $\mu(\sigma) \cdot H_{X,Y} = H_{\sigma(X,\dots,X),Y}$.        (12)

  3. 3.1. Make an equivalence query on $A$.
          If the answer is YES then output $A$ and halt.
          Otherwise the answer is NO and $z$ is a counterexample. Searching bottom-up,
          find a subtree $\sigma(\tau_1, \dots, \tau_k)$ of $z$ that satisfies the following two conditions:
           (i) For every $j \in [k]$, $H_{\tau_j,Y} = \mu(\tau_j) \cdot H_{X,Y}$.
          (ii) For some $c \in Y$, $H_{\sigma(\tau_1,\dots,\tau_k),c} \neq \mu(\sigma(\tau_1, \dots, \tau_k)) \cdot H_{X,c}$.
     3.2. For every $j \in [k]$ and $(i_1, \dots, i_{j-1}) \in [n]^{j-1}$:
          $Y \leftarrow Y \cup \{c[\sigma(t_{i_1}, \dots, t_{i_{j-1}}, \Box, \tau_{j+1}, \dots, \tau_k)]\}$.
     3.3. For every $j \in [k]$:
          If $H_{\tau_j,Y}$ is not a linear combination of $H_{t_1,Y}, \dots, H_{t_n,Y}$ then
          $n \leftarrow n+1$, $t_n \leftarrow \tau_j$, $X \leftarrow X \cup \{t_n\}$.
     3.4. Go to 2.
```

Table 2: Exact learning algorithm LMTA for the class of multiplicity tree automata

Lemma 14 *Linear independence of the set of vectors $\{H_{t_1,Y}, \dots, H_{t_n,Y}\}$ is an invariant of the loop consisting of Step 2 and Step 3.*

Proof We argue inductively on the number of iterations of the loop. The base case $n = 1$ clearly holds since $f(z) \neq 0$.

For the induction step, suppose that the set $\{H_{t_1,Y}, \dots, H_{t_n,Y}\}$ is linearly independent at the start of an iteration of the loop. If a tree $t \in T_\Sigma$ is added to X during Step 2.1, then $H_{t,Y}$ is not a linear combination of $H_{t_1,Y}, \dots, H_{t_n,Y}$, and therefore $\{H_{t_1,Y}, \dots, H_{t_n,Y}, H_{t,Y}\}$ is a linearly independent set of vectors. Hence, the set $\{H_{t_1,Y}, \dots, H_{t_n,Y}\}$ is linearly independent at the start of Step 3.

Unless the algorithm halts in Step 3.1, it proceeds to Step 3.2 where the set of columns Y is increased, which clearly preserves linear independence of vectors $H_{t_1,Y}, \ldots, H_{t_n,Y}$. If a tree τ_j is added to X in Step 3.3, then $H_{\tau_j,Y}$ is not a linear combination of $H_{t_1,Y}, \ldots, H_{t_n,Y}$ which implies that the vectors $H_{t_1,Y}, \ldots, H_{t_n,Y}, H_{\tau_j,Y}$ are linearly independent. Hence, the set $\{H_{t_1,Y}, \ldots, H_{t_n,Y}\}$ is linearly independent at the start of the next iteration of the loop. This completes the induction step. ■

Secondly, we show that Step 2.2 of LMTA can always be performed.

Lemma 15 *Whenever Step 2.2 starts, for every $k \in \{0, \ldots, m\}$ and $\sigma \in \Sigma_k$ there exists a unique matrix $\mu(\sigma) \in \mathbb{F}^{n^k \times n}$ satisfying Equation (12).*

Proof Take any $(i_1, \ldots, i_k) \in [n]^k$. Step 2.1 ensures that $H_{\sigma(t_{i_1},\ldots,t_{i_k}),Y}$ can be represented as a linear combination of vectors $H_{t_1,Y}, \ldots, H_{t_n,Y}$. This representation is unique since $H_{t_1,Y}, \ldots, H_{t_n,Y}$ are linearly independent vectors by Lemma 14. Row $\mu(\sigma)_{(i_1,\ldots,i_k)} \in \mathbb{F}^{1\times n}$ is, therefore, uniquely defined by the equation $\mu(\sigma)_{(i_1,\ldots,i_k)} \cdot H_{X,Y} = H_{\sigma(t_{i_1},\ldots,t_{i_k}),Y}$. ■

Thirdly, we show that Step 3.1 of LMTA can always be performed.

Lemma 16 *Suppose that upon making an equivalence query on A in Step 3.1, the Learner receives the answer NO and a counterexample z. Then, there exists a subtree $\sigma(\tau_1, \ldots, \tau_k)$ of z that satisfies the following two conditions:*

(i) For every $j \in [k]$, $H_{\tau_j,Y} = \mu(\tau_j) \cdot H_{X,Y}$.

(ii) For some $c \in Y$, $H_{\sigma(\tau_1,\ldots,\tau_k),c} \neq \mu(\sigma(\tau_1, \ldots, \tau_k)) \cdot H_{X,c}$.

Proof Towards a contradiction, assume that there exists no subtree $\sigma(\tau_1, \ldots, \tau_k)$ of z that satisfies conditions (i) and (ii). We claim that then for every subtree τ of z, it holds that

$$H_{\tau,Y} = \mu(\tau) \cdot H_{X,Y}. \tag{13}$$

In the following we prove this claim using induction on *height*(τ). The base case $\tau \in \Sigma_0$ follows immediately from Equation (12). For the induction step, let $0 \le h < \textit{height}(z)$ and assume that Equation (13) holds for every subtree $\tau \in T_\Sigma^{\le h}$ of z. Take an arbitrary subtree $\tau \in T_\Sigma^{h+1}$ of z. Then $\tau = \sigma(\tau_1, \ldots, \tau_k)$ for some $k \in [m]$, $\sigma \in \Sigma_k$, and $\tau_1, \ldots, \tau_k \in T_\Sigma^{\le h}$, where $\tau_1, \ldots, \tau_k$ are subtrees of z. The induction hypothesis implies that $H_{\tau_j,Y} = \mu(\tau_j) \cdot H_{X,Y}$ holds for every $j \in [k]$. Hence, subtree τ satisfies condition (i). By assumption, no subtree of z satisfies both conditions (i) and (ii). Thus τ does not satisfy condition (ii), i.e., it holds that $H_{\tau,Y} = \mu(\tau) \cdot H_{X,Y}$. This completes the proof by induction.

Equation (13) for $\tau = z$ gives $H_{z,Y} = \mu(z) \cdot H_{X,Y}$. Since $\Box \in Y$, this in particular implies that

$$f(z) = H_{z,\Box} = \mu(z) \cdot H_{X,\Box} = \mu(z) \cdot \gamma = \|A\|(z),$$

which yields a contradiction since z is a counterexample for the hypothesis A. ■

Finally, we show that the row set X is augmented with at least one element in each iteration of the main loop.

Lemma 17 *Every complete iteration of the Step 2 - 3 loop strictly increases the cardinality of the row set X.*

Proof It suffices to show that in Step 3.3 at least one of the trees $\tau_1, \ldots, \tau_k$ is added to X. By Lemma 14, at the start of Step 3.2 vectors $H_{t_1,Y}, \ldots, H_{t_n,Y}$ are linearly independent. Thus by condition (i) of Step 3.1, for every $j \in [k]$ it holds that

$$H_{\tau_j,Y} = \mu(\tau_j) \cdot H_{X,Y} \tag{14}$$

and, moreover, Equation (14) is the unique representation of vector $H_{\tau_j,Y}$ as a linear combination of vectors $H_{t_1,Y}, \ldots, H_{t_n,Y}$. Clearly, vectors $H_{t_1,Y}, \ldots, H_{t_n,Y}$ remain linearly independent when Step 3.2 ends.

Towards a contradiction, assume that in Step 3.3 none of the trees $\tau_1, \ldots, \tau_k$ is added to X. This means that for every $j \in [k]$, vector $H_{\tau_j,Y}$ can be represented as a linear combination of $H_{t_1,Y}, \ldots, H_{t_n,Y}$. The latter representation is unique, since vectors $H_{t_1,Y}, \ldots, H_{t_n,Y}$ are linearly independent, and is given by Equation (14). By condition (ii) of Step 3.1 and Equations (12) and (1), we now have that

$$\begin{aligned}
H_{\sigma(\tau_1,\ldots,\tau_k),c} &\neq \mu(\sigma(\tau_1, \ldots, \tau_k)) \cdot H_{X,c} \\
&= (\mu(\tau_1) \otimes \cdots \otimes \mu(\tau_k)) \cdot \mu(\sigma) \cdot H_{X,c} \\
&= (\mu(\tau_1) \otimes \ldots \otimes \mu(\tau_k)) \cdot H_{\sigma(X,\ldots,X),c} \\
&= \sum_{(i_1,\ldots,i_k)\in[n]^k} \left(\prod_{j=1}^{k} \mu(\tau_j)_{i_j} \right) \cdot H_{\sigma(t_{i_1},\ldots,t_{i_k}),c}.
\end{aligned} \tag{15}$$

By Step 3.2, it holds that $c[\sigma(t_{i_1}, \ldots, t_{i_{j-1}}, \Box, \tau_{j+1}, \ldots, \tau_k)] \in Y$ for every $j \in [k]$ and every $(i_1, \ldots, i_{j-1}) \in [n]^{j-1}$. Thus by Equation (14) for $j = k$, we have

$$\begin{aligned}
&\sum_{(i_1,\ldots,i_k)\in[n]^k} \left(\prod_{j=1}^{k} \mu(\tau_j)_{i_j} \right) \cdot H_{\sigma(t_{i_1},\ldots,t_{i_k}),c} \\
&= \sum_{(i_1,\ldots,i_{k-1})\in[n]^{k-1}} \left(\prod_{j=1}^{k-1} \mu(\tau_j)_{i_j} \right) \cdot \sum_{i\in[n]} \mu(\tau_k)_i \cdot H_{t_i,c[\sigma(t_{i_1},\ldots,t_{i_{k-1}},\Box)]} \\
&= \sum_{(i_1,\ldots,i_{k-1})\in[n]^{k-1}} \left(\prod_{j=1}^{k-1} \mu(\tau_j)_{i_j} \right) \cdot \mu(\tau_k) \cdot H_{X,c[\sigma(t_{i_1},\ldots,t_{i_{k-1}},\Box)]} \\
&= \sum_{(i_1,\ldots,i_{k-1})\in[n]^{k-1}} \left(\prod_{j=1}^{k-1} \mu(\tau_j)_{i_j} \right) \cdot H_{\tau_k,c[\sigma(t_{i_1},\ldots,t_{i_{k-1}},\Box)]}.
\end{aligned} \tag{16}$$

Proceeding inductively as above and applying Equation (14) for every $j \in \{k-1, \ldots, 1\}$, we get that the expression of (16) is equal to $H_{\tau_1,c[\sigma(\Box,\tau_2,\ldots,\tau_k)]}$. However, this contradicts Equation (15). The result follows. ■

Putting together Lemmas 14 - 17, we conclude the following:

Proposition 18 *Let Σ be a ranked alphabet and $\mathbb{F}$ be a field. Let $f \in Rec(\Sigma, \mathbb{F})$, let H be the Hankel matrix of f, and let r be the rank (over $\mathbb{F}$) of H. On target f, algorithm* LMTA *outputs a minimal MTA-representation of f after at most r iterations of the loop consisting of Step 2 and Step 3.*

Proof Lemmas 15 and 16 show that every step of algorithm LMTA can be performed.

Theorem 3 implies that r is finite. From Lemma 14 we know that, whenever Step 2 starts, matrix $H_{X,Y}$ has full row rank and thus $n = |X| \leq r$. Lemma 17 implies that n increases by at least one in each iteration of the Step 2 - 3 loop. Therefore, the number of iterations of the loop is at most r.

The proof follows by observing that LMTA halts only upon receiving the answer YES to an equivalence query. ■

4.3 Succinct Representations

In this subsection, we explain how algorithm LMTA can be correctly implemented using a DAG representation of trees. In particular, we assume that membership queries are made on Σ-DAGs, that the counterexamples are given as Σ-DAGs, the elements of X are Σ-DAGs, and the elements of Y are DAG representations of Σ-contexts, i.e., $(\{\Box\} \cup \Sigma)$-DAGs.

As shown in Section 2.5, multiplicity tree automata can run directly on DAGs and, moreover, they assign equal weight to a DAG and to its tree unfolding. Crucially also, as explained in the proof of Theorem 19, Step 3.1 can be run directly on a DAG representation of the counterexample, without unfolding. Specifically, Step 3.1 involves multiple executions of the hypothesis automaton on trees. By Proposition 7, we can faithfully carry out these executions on DAG representations of trees. Step 3.1 also involves considering all the subtrees of a given counterexample. However, by Proposition 5, this is equivalent to looking at all the sub-DAGs of a DAG representation of the counterexample.

At various points in the algorithm, we take $c \in Y$, $t \in X$ and compute their concatenation $c[t]$ in order to determine the corresponding entry $H_{t,c}$ of the Hankel matrix by making a membership query. Proposition 6 implies that this can be done faithfully using DAG representations of Σ-trees and Σ-contexts.

4.4 Complexity Analysis

In this subsection, we give a query and computational complexity analysis of our algorithm and compare it to the best previously-known exact learning algorithm for multiplicity tree automata (Habrard and Oncina, 2006) showing in particular an exponential improvement on the query complexity and the running time in the worst case.

Theorem 19 *Let $f \in Rec(\Sigma, \mathbb{F})$ where Σ has rank m and $\mathbb{F}$ is a field. Let A be a minimal MTA-representation of f, and let r be the dimension of A. Then, f is learnable by the algorithm* LMTA*, making $r + 1$ equivalence queries, $|A|^2 + |A| \cdot s$ membership queries, and $O(|A|^2 + |A| \cdot r \cdot s)$ arithmetic operations, where s denotes the size of a largest counterexample z, represented as a DAG, that is obtained during the execution of the algorithm.*

Proof Let H be the Hankel matrix of f. Note that, by Theorem 3, the rank of H is equal to r. Proposition 18 implies that on target f, algorithm LMTA outputs a minimal MTA-representation of f after at most r iterations of the Step 2 - 3 loop, thereby making at most $r + 1$ equivalence queries.

From Lemma 14 we know that matrix $H_{X,Y}$ has full row rank, which implies that $|X| \le r$. As for the cardinality of the column set Y, at the end of Step 1 we have $|Y| = 1$. Furthermore, in each iteration of Step 3.2 the number of columns added to Y is at most

$$\sum_{j=1}^{k} n^{j-1} \le \sum_{j=1}^{k} r^{j-1} = \frac{r^k - 1}{r-1} \le \frac{r^m - 1}{r-1},$$

where k and n are as defined in Step 3.2. Since the number of iterations of Step 3.2 is at most $r - 1$, we have $|Y| \le r^m$.

The number of membership queries made in Step 2 over the whole algorithm is

$$\left(\sum_{\sigma \in \Sigma} |\sigma(X, \ldots, X)| + |X| \right) \cdot |Y|$$

because the Learner needs to ask for the values of the entries of matrices $H_{X,Y}$ and $H_{\sigma(X,\ldots,X),Y}$ for every $\sigma \in \Sigma$.

To analyse the number of membership queries made in Step 3, we now detail the procedure by which an appropriate sub-DAG of the counterexample z is found in Step 3.1. By Lemma 16, there exists a sub-DAG τ of z such that $H_{\tau,Y} \ne \mu(\tau) \cdot H_{X,Y}$. Thus given a counterexample z in Step 3.1, the procedure for finding a required sub-DAG of z is as follows: Check if $H_{\tau,Y} = \mu(\tau) \cdot H_{X,Y}$ for every sub-DAG τ of z in a nondecreasing order of height; stop when a sub-DAG τ is found such that $H_{\tau,Y} \ne \mu(\tau) \cdot H_{X,Y}$.

In each iteration of Step 3, the Learner makes $\mathit{size}(z) \cdot |Y| \le s \cdot |Y|$ membership queries because, for every sub-DAG τ of z, the Learner needs to ask for the values of the entries of vector $H_{\tau,Y}$. All together, the number of membership queries made during the execution of the algorithm is at most

$$\begin{aligned} &\left(\sum_{\sigma \in \Sigma} |\sigma(X, \ldots, X)| + |X| \right) \cdot |Y| + (r-1) \cdot s \cdot |Y| \\ \le &\left(\sum_{\sigma \in \Sigma} r^{rk(\sigma)} + r \right) \cdot r^m + (r-1) \cdot s \cdot r^m \le |A|^2 + |A| \cdot s. \end{aligned}$$

As for the arithmetic complexity, in Step 2.1 one can determine if a vector $H_{\sigma(t_{i_1},\ldots,t_{i_k}),Y}$ is a linear combination of $H_{t_1,Y}, \ldots, H_{t_n,Y}$ via Gaussian elimination using $O(n^2 \cdot |Y|)$ arithmetic operations (see Cohen, 1993, Section 2.3). Analogously, in Step 3.3 one can determine if $H_{\tau_j,Y}$ is a linear combination of $H_{t_1,Y}, \ldots, H_{t_n,Y}$ via Gaussian elimination using $O(n^2 \cdot |Y|)$ arithmetic operations. Since $|X| \le r$ and $|Y| \le r^m$, all together Step 2.1 and Step 3.3 require at most $O(|A|^2)$ arithmetic operations.

Lemma 15 implies that in each iteration of Step 2.2, for every $\sigma \in \Sigma$ there exists a unique matrix $\mu(\sigma) \in \mathbb{F}^{n^{rk(\sigma)} \times n}$ that satisfies Equation (12). To perform an iteration of Step 2.2, we first put matrix $H_{X,Y}$ in echelon form and then, for each $\sigma \in \Sigma$, solve Equation (12) for $\mu(\sigma)$ by back substitution. It follows from standard complexity bounds on the conversion of matrices to echelon form (Cohen, 1993, Section 2.3) that the total operation count for Step 2.2 can be bounded above by $O(|A|^2)$.

Finally, let us consider the arithmetic complexity of Step 3.1. In every iteration, for each sub-DAG τ of the counterexample z the Learner needs to compute the vector $\mu(\tau)$ and the product $\mu(\tau) \cdot H_{X,Y}$. Note that $\mu(\tau)$ can be computed bottom-up from the sub-DAGs of τ. Since z has at most s sub-DAGs, Step 3.1 requires at most $O(|A| \cdot r \cdot s)$ arithmetic operations. All together, the algorithm requires at most $O(|A|^2 + |A| \cdot r \cdot s)$ arithmetic operations. ■

Algorithm LMTA can be used to show that over $\mathbb{Q}$, multiplicity tree automata are exactly learnable in randomised polynomial time. The key idea is to represent numbers as arithmetic circuits. In executing LMTA, the Learner need only perform arithmetic operations on circuits (addition, subtraction, multiplication, and division), which can be done in constant time, and equality testing, which can be done in coRP (see Arora and Barak, 2009). These suffice for all the operations detailed in the proof of Theorem 19; in particular they suffice for Gaussian elimination, which can be used to implement the linear-independence checks in LMTA.

The complexity of algorithm LMTA should be compared to the complexity of the algorithm of Habrard and Oncina (2006), which learns multiplicity tree automata by making $r+1$ equivalence queries, $|A| \cdot s$ membership queries, and a number of arithmetic operations polynomial in $|A|$ and s, where s is the size of a largest counterexample given as a tree. Note that the algorithm of Habrard and Oncina (2006) cannot be straightforwardly adapted to work directly with DAG representations of trees since when given a counterexample z, every suffix of z is added to the set of columns. However, the tree unfolding of a DAG can have exponentially many different suffixes in the size of the DAG. For example, the DAG in Figure 2 has size n, and its tree unfolding, shown in Figure 1, has $O(2^n)$ different suffixes.

5. Lower Bounds on Query Complexity of Learning MTA

In this section, we study lower bounds on the query complexity of learning multiplicity tree automata in the exact learning model. Our results generalise the corresponding lower bounds for learning multiplicity word automata by Bisht et al. (2006), and make no assumption about the computational model of the learning algorithm.

First, we give a lower bound on the total number of queries required by an exact learning algorithm that works over any field, which is the situation of our algorithm in Section 4. Note that when we say that an algorithm works over any field, we mean that it just uses field arithmetic, equality testing, and the ability to store and communicate field elements to the Teacher, and its correctness depends only on these operations satisfying the field axioms.

Theorem 20 *Any exact learning algorithm that learns the class of multiplicity tree automata of dimension at most r, over a ranked alphabet (Σ, rk) and any field, must make at least $\sum_{\sigma \in \Sigma} r^{rk(\sigma)+1} - r^2$ queries.*

Proof Take an arbitrary exact learning algorithm L that learns the class of multiplicity tree automata of dimension at most r, over a ranked alphabet (Σ, rk) and over any field.

Let $\mathbb{F}$ be any field. Let $\mathbb{K} := \mathbb{F}(\{z^\sigma_{i,j} : \sigma \in \Sigma, i \in [r^{rk(\sigma)}], j \in [r]\})$ be an extension field of $\mathbb{F}$, where the set $\{z^\sigma_{i,j} : \sigma \in \Sigma, i \in [r^{rk(\sigma)}], j \in [r]\}$ is algebraically independent over $\mathbb{F}$. We define a 'generic' $\mathbb{K}$-multiplicity tree automaton $A := (r, \Sigma, \mu, \gamma)$ where $\gamma = e_1^\top \in \mathbb{F}^{r\times 1}$ and $\mu(\sigma) = [z^\sigma_{i,j}]_{i,j} \in \mathbb{K}^{r^{rk(\sigma)}\times r}$ for every $\sigma \in \Sigma$. We define a tree series $f := \|A\|$. Observe that every r-dimensional $\mathbb{F}$-MTA over Σ can be obtained from A by substituting values from the field $\mathbb{F}$ for the variables $z^\sigma_{i,j}$. Thus if the Hankel matrix of f had rank less than r, then every r-dimensional $\mathbb{F}$-MTA over Σ would have Hankel matrix of rank less than r. Therefore, the Hankel matrix of f has rank r.

We run algorithm L on the target function f. By assumption, the output of L is an MTA $A' = (r, \Sigma, \mu', \gamma')$ such that $\|A'\| \equiv f$. Let n be the number of queries made by L on target f. Let $t_1, \ldots, t_n \in T_\Sigma$ be the trees on which L either made a membership query, or which were received as the counterexample to an equivalence query. Then for every $l \in [n]$, there exists a multivariate polynomial $p_l \in \mathbb{F}[(z^\sigma_{i,j})_{i,j,\sigma}]$ such that $f(t_l) = p_l$.

Note that both A and A' are minimal MTA-representations of f. Thus by Theorem 4, there exists an invertible matrix $U \in \mathbb{K}^{r\times r}$ such that $\gamma = U\cdot\gamma'$ and $\mu(\sigma) = U^{\otimes rk(\sigma)}\cdot\mu'(\sigma)\cdot U^{-1}$ for every $\sigma \in \Sigma$. This implies that the entries of matrices $\mu(\sigma)$, $\sigma \in \Sigma$, lie in an extension of $\mathbb{F}$ generated by the entries of U and $\{p_l : l \in [n]\}$, i.e., by at most $r^2 + n$ elements. But since the entries of matrices $\mu(\sigma)$, $\sigma \in \Sigma$, form an algebraically independent set over $\mathbb{F}$, the total number $\sum_{\sigma\in\Sigma} r^{rk(\sigma)+1}$ of such entries is at most $r^2 + n$. Therefore, the number of queries n is at least $\sum_{\sigma\in\Sigma} r^{rk(\sigma)+1} - r^2$. ■

One may wonder whether a learning algorithm could do better over a specific field $\mathbb{F}$ by exploiting particular features of that field such as having zero characteristic, being ordered, or being algebraically closed. In this setting, we have the following lower bound.

Theorem 21 *Let $\mathbb{F}$ be a fixed but arbitrary field. Any exact learning algorithm that learns the class of $\mathbb{F}$-multiplicity tree automata of dimension at most r, over a ranked alphabet (Σ, rk) that has rank m and contains at least one unary symbol, must make number of queries at least*

$$\frac{1}{2^{m+1}} \cdot \left(\sum_{\sigma\in\Sigma} r^{rk(\sigma)+1} - r^2 - r\right).$$

Proof Without loss of generality, we can assume that r is even and can, therefore, define a natural number $n := r/2$. Let L be an exact learning algorithm for the class of $\mathbb{F}$-multiplicity tree automata of dimension at most r, over a ranked alphabet (Σ, rk) of rank m such that $rk^{-1}(\{1\}) \neq \emptyset$. We will identify a class of functions $\mathcal{C}$ such that L has to make at least $\sum_{\sigma\in\Sigma} n^{rk(\sigma)+1} - n^2 - n$ queries to distinguish between the members of $\mathcal{C}$.

Let $\sigma_0, \sigma_1 \in \Sigma$ be a nullary and a unary symbol, respectively. Let $P \in \mathbb{F}^{n\times n}$ be the permutation matrix corresponding to the cycle $(1, 2, \ldots, n)$. Define $\mathcal{A}$ to be the set of all $\mathbb{F}$-multiplicity tree automata $(2n, \Sigma, \mu, \gamma)$ where:

- $\mu(\sigma_0) = \begin{bmatrix}1 & 0\end{bmatrix} \otimes e_1$ and $\mu(\sigma_1) = I_2 \otimes P$;

- For each k-ary symbol $\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}$, there exists $B(\sigma) \in \mathbb{F}^{n^k \times n}$ such that

$$\mu(\sigma) = \begin{bmatrix} 1 & 1 \end{bmatrix} \otimes \left(\begin{bmatrix} I_n \\ -I_n \end{bmatrix}^{\otimes k} \cdot B(\sigma) \right);$$

- $\gamma = \begin{bmatrix} 1 & 0 \end{bmatrix}^\top \otimes e_1^\top$.

We define a set of recognisable tree series $\mathcal{C} := \{\|A\| : A \in \mathcal{A}\}$.

In Lemma 22 we state some properties of the functions in $\mathcal{C}$. Specifically, we show that the coefficient of a tree $t \in T_\Sigma$ in any series $f \in \mathcal{C}$ fundamentally depends on whether t has zero, one, or at least two nodes whose label is not σ_0 or σ_1. Here for every $i \in \mathbb{N}_0$ and $t \in T_\Sigma$, we use $\sigma_1^i(t)$ to denote the tree $\underbrace{\sigma_1(\sigma_1(\ldots\sigma_1(}_{i} t) \ldots))$.

Lemma 22 *The following properties hold for every $f \in \mathcal{C}$ and $t \in T_\Sigma$:*

(i) If $t = \sigma_1^j(\sigma_0)$ where $j \in \{0, 1, \ldots, n-1\}$, then $f(\sigma_0) = 1$ and $f(\sigma_1^j(\sigma_0)) = 0$ for $j > 0$.

(ii) If $t = \sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0), \ldots, \sigma_1^{i_k}(\sigma_0)))$ where $k \in \{0, 1, ..., m\}$, $\sigma \in \Sigma_k \setminus \{\sigma_0, \sigma_1\}$, and $j, i_1, \ldots, i_k \in \{0, 1, \ldots, n-1\}$, then $f(t) = B(\sigma)_{(1+i_1, \ldots, 1+i_k), (1+n-j) \bmod n}$.

(iii) If $\sum_{\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}} \#_\sigma(t) \geq 2$, then $f(t) = 0$.

Proof Let $A = (2n, \Sigma, \mu, \gamma) \in \mathcal{A}$ be such that $\|A\| \equiv f$. First, we prove property *(i)*. Using Equation (2) and the mixed-product property of Kronecker product, we get that

$$\mu(\sigma_1^j(\sigma_0)) \ = \mu(\sigma_0) \cdot \mu(\sigma_1)^j = (\begin{bmatrix} 1 & 0 \end{bmatrix} \otimes e_1) \cdot (I_2 \otimes P^j) = \begin{bmatrix} 1 & 0 \end{bmatrix} \otimes e_1 P^j \tag{17}$$

and therefore

$$\begin{aligned} f(\sigma_1^j(\sigma_0)) &= \mu(\sigma_1^j(\sigma_0)) \cdot \gamma = (\begin{bmatrix} 1 & 0 \end{bmatrix} \otimes e_1 P^j) \cdot (\begin{bmatrix} 1 & 0 \end{bmatrix}^\top \otimes e_1^\top) \\ &= (\begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \end{bmatrix}^\top) \otimes (e_1 P^j \cdot e_1^\top) = e_{j+1} \cdot e_1^\top. \end{aligned} \tag{18}$$

If $j = 0$ then the expression of (18) is equal to 1, otherwise the expression of (18) is equal to 0. This completes the proof of property *(i)*.

Next, we prove property *(ii)*. By the mixed-product property of Kronecker product and Equations (2), (3), and (17), we have

$$\begin{aligned} &\mu(\sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0), \ldots, \sigma_1^{i_k}(\sigma_0)))) \\ &= \left(\bigotimes_{l=1}^{k} \mu(\sigma_1^{i_l}(\sigma_0)) \right) \cdot \mu(\sigma) \cdot \mu(\sigma_1)^j \\ &= \left(\begin{bmatrix} 1 \end{bmatrix} \otimes \bigotimes_{l=1}^{k} \mu(\sigma_1^{i_l}(\sigma_0)) \right) \cdot \left(\begin{bmatrix} 1 & 1 \end{bmatrix} \otimes \left(\begin{bmatrix} I_n \\ -I_n \end{bmatrix}^{\otimes k} \cdot B(\sigma) \right) \right) \cdot (I_2 \otimes P)^j \\ &= \left((\begin{bmatrix} 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \end{bmatrix}) \otimes \left(\bigotimes_{l=1}^{k} \mu(\sigma_1^{i_l}(\sigma_0)) \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix}^{\otimes k} \cdot B(\sigma) \right) \right) \cdot (I_2 \otimes P^j) \end{aligned}$$

$$\begin{aligned}
&= \left(\begin{bmatrix}1 & 1\end{bmatrix} \otimes \left(\bigotimes_{l=1}^{k}\left(\left(\begin{bmatrix}1 & 0\end{bmatrix} \otimes e_1 P^{i_l}\right) \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix}\right) \cdot B(\sigma)\right)\right) \cdot (I_2 \otimes P^j) \\
&= \left(\begin{bmatrix}1 & 1\end{bmatrix} \otimes \left(\bigotimes_{l=1}^{k} e_1 P^{i_l} \cdot B(\sigma)\right)\right) \cdot (I_2 \otimes P^j) \\
&= \left(\begin{bmatrix}1 & 1\end{bmatrix} \cdot I_2\right) \otimes \left(\bigotimes_{l=1}^{k} e_{1+i_l} \cdot B(\sigma) \cdot P^j\right) \\
&= \begin{bmatrix}1 & 1\end{bmatrix} \otimes (B(\sigma)_{(1+i_1,\ldots,1+i_k)} \cdot P^j) \qquad (19)
\end{aligned}$$

and therefore, using the fact that $P^n = I_n$, we get that

$$\begin{aligned}
f(\sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0),\ldots,\sigma_1^{i_k}(\sigma_0)))) &= \mu(\sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0),\ldots,\sigma_1^{i_k}(\sigma_0)))) \cdot \gamma \\
&= (\begin{bmatrix}1 & 1\end{bmatrix} \otimes (B(\sigma)_{(1+i_1,\ldots,1+i_k)} \cdot P^j)) \cdot (\begin{bmatrix}1 & 0\end{bmatrix}^\top \otimes e_1^\top) \\
&= (\begin{bmatrix}1 & 1\end{bmatrix} \cdot \begin{bmatrix}1 & 0\end{bmatrix}^\top) \otimes (B(\sigma)_{(1+i_1,\ldots,1+i_k)} \cdot P^j \cdot e_1^\top) \\
&= B(\sigma)_{(1+i_1,\ldots,1+i_k)} \cdot (e_1 P^{n-j})^\top \\
&= B(\sigma)_{(1+i_1,\ldots,1+i_k),(1+n-j) \bmod n}.
\end{aligned}$$

Finally, we prove property *(iii)*. If $\sum_{\sigma \in \Sigma \setminus \{\sigma_0,\sigma_1\}} \#_\sigma(t) \geq 2$, then there exists a subtree $\sigma'(t_1,\ldots,t_k)$ of t where $k \geq 1$, $\sigma' \in \Sigma_k \setminus \{\sigma_1\}$, and $\sum_{\sigma \in \Sigma \setminus \{\sigma_0,\sigma_1\}} \#_\sigma(t_i) = 1$ for some $i \in [k]$. It follows from Equation (19) that $\mu(t_i) = \begin{bmatrix}1 & 1\end{bmatrix} \otimes \alpha$ for some $\alpha \in \mathbb{F}^{1\times n}$. By the mixed-product property of Kronecker product and Equation (3), we have

$$\begin{aligned}
\mu(\sigma'(t_1,\ldots,t_k)) &= \left(\bigotimes_{j=1}^{k} \mu(t_j)\right) \cdot \left(\begin{bmatrix}1 & 1\end{bmatrix} \otimes \left(\begin{bmatrix} I_n \\ -I_n \end{bmatrix}^{\otimes k} \cdot B(\sigma')\right)\right) \\
&= \begin{bmatrix}1 & 1\end{bmatrix} \otimes \left(\bigotimes_{j=1}^{k} \mu(t_j) \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix}^{\otimes k} \cdot B(\sigma')\right) \\
&= \begin{bmatrix}1 & 1\end{bmatrix} \otimes \left(\bigotimes_{j=1}^{k} \left(\mu(t_j) \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix}\right) \cdot B(\sigma')\right) = 0_{1\times 2n}
\end{aligned}$$

where the last equality holds because

$$\mu(t_i) \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix} = \begin{bmatrix}\alpha & \alpha\end{bmatrix} \cdot \begin{bmatrix} I_n \\ -I_n \end{bmatrix} = 0_{1\times n}.$$

Since $\sigma'(t_1,\ldots,t_k)$ is a subtree of t, we now have that $\mu(t) = 0_{1\times 2n}$ and thus $f(t) = 0$. ■

Remark 23 *As $P^n = I_n$, we have $\mu(\sigma_1)^n = I_{2n}$. Thus for every $f \in \mathcal{C}$, $k \in \{0, 1, ..., m\}$, $\sigma \in \Sigma_k \setminus \{\sigma_0, \sigma_1\}$, and $j, i_1, \ldots, i_k \in \mathbb{N}_0$, it holds that $f(\sigma_1^j(\sigma_0)) = f(\sigma_1^{j \bmod n}(\sigma_0))$ and*

$$f(\sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0),\ldots,\sigma_1^{i_k}(\sigma_0)))) = f(\sigma_1^{j \bmod n}(\sigma(\sigma_1^{i_1 \bmod n}(\sigma_0),\ldots,\sigma_1^{i_k \bmod n}(\sigma_0)))).$$

Returning to the proof of Theorem 21, let us run the learning algorithm L on a target $f \in \mathcal{C}$. Lemma 22 (i) and Remark 23 imply that when L makes a membership query on $t = \sigma_1^j(\sigma_0)$ where $j \in \mathbb{N}_0$, the Teacher returns 1 if $j \bmod n = 0$ and returns 0 otherwise. Furthermore, by Lemma 22 (iii), when L makes a membership query on $t \in T_\Sigma$ such that $\sum_{\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}} \#_\sigma(t) \geq 2$, the Teacher returns 0. In these cases, L does not gain any new information about f since every function in $\mathcal{C}$ is consistent with the values returned by the Teacher.

When L makes a membership query on a tree $t = \sigma_1^j(\sigma(\sigma_1^{i_1}(\sigma_0), \ldots, \sigma_1^{i_k}(\sigma_0)))$, where $k \in \{0, 1, ..., m\}$, $\sigma \in \Sigma_k \setminus \{\sigma_0, \sigma_1\}$, and $j, i_1, \ldots, i_k \in \mathbb{N}_0$, the Teacher returns an arbitrary number from the field $\mathbb{F}$ if the value $f(t)$ is not already known from an earlier query. It follows from Lemma 22 (ii) and Remark 23 that L thereby learns the entry

$$B(\sigma)_{(1+(i_1 \bmod n), \ldots, 1+(i_k \bmod n)), (1+n-j) \bmod n}.$$

When L makes an equivalence query on a hypothesis $h \in \mathcal{C}$, the Teacher finds some entry $B(\sigma)_{(i_1, \ldots, i_k), j}$ that L does not already know from previous queries and returns the tree $\sigma_1^{1+n-j}(\sigma(\sigma_1^{i_1-1}(\sigma_0), \ldots, \sigma_1^{i_k-1}(\sigma_0)))$ as the counterexample.

With each query, the Learner L learns at most one entry of $B(\sigma)$ where $\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}$. The number of queries made by L on target f is, therefore, at least the total number of entries of matrices $B(\sigma)$ for all $\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}$. The latter number is equal to

$$\begin{aligned}\sum_{\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}} n^{rk(\sigma)+1} &\geq \frac{1}{2^{m+1}} \cdot \sum_{\sigma \in \Sigma \setminus \{\sigma_0, \sigma_1\}} r^{rk(\sigma)+1} \\ &= \frac{1}{2^{m+1}} \cdot \left(\sum_{\sigma \in \Sigma} r^{rk(\sigma)+1} - r^2 - r \right).\end{aligned}$$

This completes the proof. ■

The lower bounds of Theorem 20 and Theorem 21 are both linear in the target automaton size. Note that when the alphabet rank is fixed, the lower bound for learning over a fixed field (Theorem 21) is the same, up to a constant factor, as for learning over an arbitrary field (Theorem 20).

Assuming a Teacher that represents counterexamples as succinctly as possible (see Section 3.2 for details), the upper bound of algorithm LMTA from Theorem 19 is quadratic in the target automaton size and, therefore, also quadratic in the lower bound of Theorem 20.

6. Conclusions and Future Work

In this work, we have characterised the query and computational complexity of learning multiplicity tree automata in the exact learning model. We gave the first-known lower bound on the number of queries needed by any exact learning algorithm to learn a target recognisable tree series. This bound is linear in the size of a smallest multiplicity tree automaton recognising the series. We also gave a new learning algorithm whose query complexity is quadratic in the size of a smallest automaton recognising the target tree series and linear in the size of a largest DAG counterexample provided by the Teacher. With regard to computational complexity, we show that the problem of deciding equivalence of multiplicity tree automata is logspace equivalent to polynomial identity testing.

The algebraic theory of recognisable word series, notably the connection to finite-rank Hankel matrices, generalises naturally to recognisable tree series and underlies many of the approaches to learning tree automata, including the present paper (see Section 1.1 for more details). In the case of trees, however, the issue of succinctness of automaton and counterexample representations comes to the fore. As we have noted, the smallest counterexample to the equivalence of two tree automata may be exponential in their total size. Therefore, in order to obtain even a polynomial query complexity, our learning algorithm works with a succinct representation of trees in terms of DAGs. The assumption of a Teacher that provides succinct DAG counterexamples is reasonable in light of the fact that the algorithm of Seidl (1990) for deciding equivalence of multiplicity tree automata can easily be modified to produce DAG counterexamples of minimal size in case of inequivalence.

The issue of succinctness of automaton representations seems to be more subtle and has not been addressed in the present paper. Here we have used the standard definition of automaton size, in which an automaton with n states and maximum alphabet rank m necessarily has size at least n^{m+1}. Adopting a sparse encoding of the transition matrices may result in an exponentially more succinct automaton representation. However, it seems a difficult problem to efficiently learn an automaton of minimal size under a sparse representation of transition matrices. In this regard, note that two different MTAs recognising the same tree series, both with a minimal number of states, can have considerably different sizes under a sparse representation since minimal MTAs are only unique up to change of basis.

One route to obtaining succinct automaton representations in the case of alphabets of unbounded rank is to use the encoding of unranked alphabets into binary alphabets presented by Comon et al. (2007) and Bailly et al. (2010). Such an encoding would potentially allow to use our learning algorithm to learn recognisable tree series over an arbitrary alphabet Σ (including even unranked alphabets) while maintaining hypothesis automaton and Hankel matrix over a binary alphabet. Note though that if the algorithm were required to present its hypotheses to the Teacher as automata over the original alphabet Σ, then it would need to translate automata over the binary encoding to corresponding automata over Σ—potentially leading to an exponential blow-up.

With regard to applications of tree-automaton learning algorithms to other problems, we recall that Beimel et al. (2000) apply their exact learning algorithm for multiplicity word automata to show exact learnability of certain classes of polynomials over both finite and infinite fields. Beimel et al. (2000) also prove the learnability of disjoint DNF formulae (i.e., DNF formulae in which each assignment satisfies at most one term) and, more generally, disjoint unions of geometric boxes over finite domains.

The learning framework considered in this paper concerns multiplicity tree automata, which are strictly more expressive than multiplicity word automata. Moreover, our result on the computational complexity of equivalence testing for multiplicity tree automata shows that, through equivalence queries, the Learner essentially has an oracle for polynomial identity testing. Thus a natural direction for future work is to seek to apply our algorithm to derive new results on exact learning of other concept classes, such as propositional formulae and polynomials (both in the commutative and noncommutative cases). In this direction, we plan to examine the relationship of our work with that of Klivans and Shpilka (2006) on exact learning of algebraic branching programs and arithmetic circuits and formulae. The latter paper relies on rank bounds for Hankel matrices of polynomials in noncommuting variables, obtained by considering a generalised notion of partial derivative. Here we would like to determine whether the extra expressiveness of tree series can be used to show learnability of more general classes of formulae and circuits than have hitherto been handled using learnability of word series.

Sakakibara (1990) showed that context-free grammars (CFGs) can be learned efficiently from their structural descriptions in the exact learning model, using structural membership queries and structural equivalence queries. Specifically, Sakakibara, *op. cit.*, notes that the set of structural descriptions of a context-free grammar constitutes a rational tree language, and thereby reduces the problem of learning a context-free grammar from its structural descriptions to the problem of learning a tree automaton. Given the important role of weighted and probabilistic CFGs across a range of applications including linguistics, a natural next step would be to apply our algorithm to learn weighted CFGs. The idea is to reduce the problem of learning a weighted context-free grammar using structural membership queries and structural equivalence queries to the problem of learning a multiplicity tree automaton in the exact learning model. The basis for applying our algorithm in this setting is the fact that the tree series that maps unlabelled derivation trees to their total weights under a given weighted context-free grammar is recognisable.

Acknowledgments

The authors would like to thank Michael Benedikt for stimulating discussions and helpful advice. We would also like to thank the referees for providing detailed and constructive reports. Both authors gratefully acknowledge the support of the EPSRC.

References

A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. *SIAM Journal on Computing*, 38(5):1987–2006, 2009.

S. Anantharaman, P. Narendran, and M. Rusinowitch. Closure properties and decision problems of DAG automata. *Information Processing Letters*, 94(5):231–240, 2005.

D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, 2009.

R. Bailly, F. Denis, and L. Ralaivola. Grammatical inference as a principal component analysis problem. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 33–40, 2009.

R. Bailly, A. Habrard, and F. Denis. A spectral approach for probabilistic grammatical inference on trees. In *Proceedings of the 21st International Conference on Algorithmic Learning Theory (ALT)*, volume 6331 of *LNCS*, pages 74–88, 2010.

B. Balle and M. Mohri. Spectral learning of general weighted automata via constrained matrix completion. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2168–2176, 2012.

A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *Journal of the ACM*, 47(3):506–530, 2000.

J. Berstel and C. Reutenauer. Recognizable formal power series on trees. *Theoretical Computer Science*, 18(2):115–148, 1982.

L. Bisht, N. H. Bshouty, and H. Mazzawi. On optimal learning algorithms for multiplicity automata. In *Proceedings of the 19th Annual Conference on Learning Theory (COLT)*, volume 4005 of *LNCS*, pages 184–198. Springer, 2006.

S. Bozapalidis and A. Alexandrakis. Représentations matricielles des séries d'arbre reconnaissables. *Informatique Théorique et Applications (ITA)*, 23(4):449–459, 1989.

S. Bozapalidis and O. Louscou-Bozapalidou. The rank of a formal tree power series. *Theoretical Computer Science*, 27(1):211–215, 1983.

P. Buneman, M. Grohe, and C. Koch. Path queries on compressed XML. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pages 141–152, 2003.

J. W. Carlyle and A. Paz. Realizations by stochastic finite automata. *Journal of Computer and System Sciences*, 5(1):26–40, 1971.

W. Charatonik. Automata on DAG representations of finite trees. Research Report MPI-I-1999-2-001, Max-Planck-Institut für Informatik, Saarbrücken, 1999.

H. Cohen. *A Course in Computational Algebraic Number Theory.* Springer-Verlag, Berlin, 1993.

H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available at `http://tata.gforge.inria.fr/`, 2007.

R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978.

F. Denis and A. Habrard. Learning rational stochastic tree languages. In *Proceedings of the 18th International Conference on Algorithmic Learning Theory (ALT)*, volume 4754 of *LNAI*, pages 242–256, 2007.

F. Denis, M. Gybels, and A. Habrard. Dimension-free concentration bounds on Hankel matrices for spectral learning. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pages 449–457, 2014.

F. Drewes and J. Högberg. Query learning of regular tree languages: How to avoid dead states. *Theory of Computing Systems*, 40(2):163–185, 2007.

F. Drewes and H. Vogler. Learning deterministically recognizable tree series. *Journal of Automata, Languages and Combinatorics*, 12(3):332–354, 2007.

F. Drewes, J. Högberg, and A. Maletti. MAT learners for tree series: an abstract data type and two realizations. *Acta Informatica*, 48(3):165–189, 2011.

L. Feng, T. Han, M. Z. Kwiatkowska, and D. Parker. Learning-based compositional verification for synchronous probabilistic systems. In *Proceedings of the 9th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 511–521, 2011.

B. Fila and S. Anantharaman. Automata for analyzing and querying compressed documents. Research Report RR-2006-03, Laboratoire d'Informatique Fondamentale d'Orléans (LIFO), Université d'Orléans, France, 2006.

M. Fliess. Matrices de Hankel. *Journal de Mathématiques Pures et Appliquées*, 53(9): 197–222, 1974.

M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees (extended abstract). In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 188–197, 2003.

E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

M. Gybels, F. Denis, and A. Habrard. Some improvements of the spectral learning approach for probabilistic grammatical inference. In *Proceedings of the 12th International Conference on Grammatical Inference (ICGI)*, pages 64–78, 2014.

A. Habrard and J. Oncina. Learning multiplicity tree automata. In *Proceedings of the 8th International Colloquium on Grammatical Inference: Algorithms and Applications (ICGI)*, volume 4201 of *LNCS*, pages 268–280. Springer, 2006.

D. Hsu, S. M. Kakade, and T. Zhang. A spectral algorithm for learning hidden Markov models. *Journal of Computer and System Sciences*, 78(5):1460–1480, 2012.

A. Kasprzik. Four one-shot learners for regular tree languages and their polynomial characterizability. *Theoretical Computer Science*, 485:85–106, 2013.

M. J. Kearns and L. G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994.

M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, Cambridge, MA, 1994.

S. Kiefer, A. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. On the complexity of equivalence and minimisation for $\mathbb{Q}$-weighted automata. *Logical Methods in Computer Science*, 9(1), 2013.

A. R. Klivans and A. Shpilka. Learning restricted models of arithmetic circuits. *Theory of Computing*, 2(1):185–206, 2006.

A. Maletti. Learning deterministically recognizable tree series - revisited. In *Proceedings of the 2nd International Conference on Algebraic Informatics (CAI)*, volume 4728 of *LNCS*, pages 218–235, 2007.

I. Marušić and J. Worrell. Complexity of equivalence and learning for multiplicity tree automata. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS), Part I*, volume 8634 of *LNCS*, pages 414–425. Springer, 2014.

Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76(2-3):223–242, 1990.

M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2–3):245–270, 1961.

J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19 (3):424–437, 1990.

A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI), Volume 1*, pages 560–566. Morgan Kaufmann, 1985.

R. E. Zippel. Probabilistic algorithms for sparse polynominals. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (EUROSAM)*, volume 72 of *LNCS*, pages 216–226. Springer, 1979.