LEARNING OF STRUCTURALLY UNAMBIGUOUS PROBABILISTIC GRAMMARS

DANA FISMAN, DOLAV NITAY, AND MICHAL ZIV-UKELSON

Ben-Gurion University, Israel
e-mail address: dana@cs.bgu.ac.il

Ben-Gurion University, Israel
e-mail address: dolavn@post.bgu.ac.il

Ben-Gurion University, Israel
e-mail address: michaluz@cs.bgu.ac.il

Abstract. The problem of identifying a probabilistic context free grammar has two aspects: the first is determining the grammar's topology (the rules of the grammar) and the second is estimating probabilistic weights for each rule. Given the hardness results for learning context-free grammars in general, and probabilistic grammars in particular, most of the literature has concentrated on the second problem. In this work we address the first problem. We restrict attention to *structurally unambiguous weighted context-free grammars* (SUWCFG) and provide a query learning algorithm for *structurally unambiguous probabilistic context-free grammars* (SUPCFG). We show that SUWCFG can be represented using *co-linear multiplicity tree automata* (CMTA), and provide a polynomial learning algorithm that learns CMTAs. We show that the learned CMTA can be converted into a probabilistic grammar, thus providing a complete algorithm for learning a structurally unambiguous probabilistic context free grammar (both the grammar topology and the probabilistic weights) using structured membership queries and structured equivalence queries. A summarized version of this work was published at AAAI 21 [NFZ21].

1. Introduction

Probabilistic context free grammars (PCFGs) constitute a computational model suitable for probabilistic systems which observe non-regular (yet context-free) behavior. They are vastly used in computational linguistics [Cho56], natural language processing [Chu88] and biological modeling, for instance, in probabilistic modeling of RNA structures [Gra95]. Methods for learning PCFGs from experimental data have been studied for over half a century. Unfortunately, there are various hardness results regarding learning context-free grammars in general and probabilistic grammars in particular. It follows from [Gol78] that context-free grammars (CFGs) cannot be identified in the limit from positive examples, and from [Ang90] that CFGs cannot be identified in polynomial time using equivalence queries only. Both results are not surprising for those familiar with learning regular languages, as

Key words and phrases: Active Learning, Learning Theory, Grammatical Inference, Multiplicity Automata, Interpretability & Analysis of NLP Models.

Preprint submitted to
Logical Methods in Computer Science

they hold for the class of regular languages as well. However, while regular languages can be learned using both membership queries and equivalence queries [Ang87], it was shown that learning CFGs using both membership queries and equivalence queries is computationally as hard as key cryptographic problems for which there is currently no known polynomial-time algorithm [AK95]. de la Higuera elaborates more on the difficulties of learning context-free grammars in his book [dlH10, Chapter 15]. Hardness results for the probabilistic setting have also been established. Abe and Warmuth have shown a computational hardness result for the inference of probabilistic automata, in particular, that an exponential blowup with respect to the alphabet size is inevitable unless $\mathbf{RP} = \mathbf{NP}$ [AW92].

The problem of identifying a probabilistic grammar from examples has two aspects: the first is determining the rules of the grammar up to variable renaming and the second is estimating probabilistic weights for each rule. Given the hardness results mentioned above, most of the literature has concentrated on the second problem. Two dominant approaches for solving the second problem are the forward-backward algorithm for HMMs [Rab89] and the inside-outside algorithm for PCFGs [Bak79, LY90].

In this work we address the first problem. Due to the hardness results regarding learning probabilistic grammars using *membership queries* and *equivalence queries* (MQ and EQ) we use *structured membership queries* and *structured equivalence queries* (SMQ and SEQ), as was done by [Sak88] for learning context-free grammars. *Structured strings*, proposed by [LJ78a], are strings over the given alphabet that includes parentheses that indicate the structure of a possible derivation tree for the string. One can equivalently think about a structured string as a derivation tree in which all nodes but the leaves are marked with ?, namely an *unlabeled derivation tree.*

It is known that the set of derivation trees of a given CFG constitutes a *regular tree-language*, where a regular tree-language is a tree-language that can be recognized by a *tree automaton* [LJ78b]. Sakakibara has generalized Angluin's $\mathbf{L}^*$ algorithm (for learning regular languages using MQ and EQ) to learning a tree automaton, and provided a polynomial learning algorithm for CFGs using SMQ and SEQ [Sak88]. Let $\mathsf{T}(\mathcal{G})$ denote the set of derivation trees of a CFG $\mathcal{G}$, and $\mathsf{S}(\mathsf{T}(\mathcal{G}))$ the set of unlabeled derivation trees (namely the structured strings of $\mathcal{G}$). While a membership query (MQ) asks whether a given string w is in the unknown grammar $\mathcal{G}$, a structured membership query (SMQ) asks whether a structured string s is in $\mathsf{S}(\mathsf{T}(\mathcal{G}))$ and a structured equivalence query (SEQ) answers whether the queried CFG $\mathcal{G}'$ is structurally equivalent to the unknown grammar $\mathcal{G}$, and accompanies a negative answer with a structured string s' in the symmetric difference of $\mathsf{S}(\mathsf{T}(\mathcal{G}'))$ and $\mathsf{S}(\mathsf{T}(\mathcal{G}))$.

In our setting, since we are interested in learning probabilistic grammars, an SMQ on a structured string s is answered by a weight $p \in [0,1]$ standing for the probability for $\mathcal{G}$ to generate s, and a negative answer to an SEQ is accompanied by a structured string s such that $\mathcal{G}$ and $\mathcal{G}'$ generate s with different probabilities, along with the probability p with which the unknown grammar $\mathcal{G}$ generates s.

Sakakibara works with tree automata to model the derivation trees of the unknown grammars [Sak88]. In our case the automaton needs to associate a weight with every tree (representing a structured string). We choose to work with the model of *multiplicity tree automata*. A multiplicity tree automaton (MTA) associates with every tree a value from a given field $\mathbb{K}$. An algorithm for learning multiplicity tree automata, to which we refer as $\mathbf{M}^*$, was developed in [HO06, DH07].[1]

[1]Following a learning algorithm developed for multiplicity word automata [BV96, BBB$^+$00].

A probabilistic grammar is a special case of a weighted grammar and [AMP99, SJ07] have shown that convergent weighted CFGs (WCFGs) where all weights are non-negative and probabilistic CFGs (PCFGs) are equally expressive.[2] We thus might expect to be able to use the learning algorithm $\mathbf{M}^*$ to learn an MTA corresponding to a WCFG, and apply this conversion to the result, in order to obtain the desired PCFG. However, as we show in Proposition 4.3, there are probabilistic languages for which applying the $\mathbf{M}^*$ algorithm results in an MTA with negative weights. Trying to adjust the algorithm to learn a positive basis may encounter the issue that for some PCFGs, no finite subset of the infinite Hankel Matrix spans the entire space of the function, as we show in Proposition 4.5.[3] To overcome these issues we restrict attention to structurally unambiguous grammars (SUCFG, see section 4.2), which as we show, can be modeled using co-linear multiplicity automata (defined next).

We develop a polynomial learning algorithm, which we term $\mathbf{C}^*$, that learns a restriction of MTA, which we term *co-linear multiplicity tree automata* (CMTA). We then show that a CMTA for a probabilistic language can be converted into a PCFG, thus yielding a complete algorithm for learning SUPCFGs using SMQs and SEQs as desired.

A summarized version of this work was published at AAAI'21 [NFZ21].

2. Preliminaries

This section provides the definitions required for *probabilistic grammars* – the object we design a learning algorithm for, and *multiplicity tree automata*, the object we use in the learning algorithm.

2.1. Probabilistic Grammars.

Probabilistic grammars are a special case of context free grammars where each production rule has a weight in the range $[0, 1]$ and for each non-terminal, the sum of weights of its productions is one.

A *context free grammar* (CFG) is a quadruple $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S\rangle$, where $\mathcal{V}$ is a finite non-empty set of symbols called *variables* or *non-terminals*, Σ is a finite non-empty set of symbols called the *alphabet* or the *terminals*, $R \subseteq \mathcal{V} \times (\mathcal{V} \cup \Sigma)^*$ is a relation between variables and strings over $\mathcal{V} \cup \Sigma$, called the *production rules*, and $S \in \mathcal{V}$ is a special variable called the *start variable*. We assume the reader is familiar with the standard definition of CFGs and of derivation trees.

We say that $S \Rightarrow w$ for a string $w \in \Sigma^*$ if there exists a derivation tree t such that all leaves are in Σ and when concatenated from left to right they form w. That is, w is the *yield* of the tree t. In this case we also use the notation $S \Rightarrow_t w$. A CFG $\mathcal{G}$ defines a set of words over Σ, the *language generated by* $\mathcal{G}$, which is the set of words $w \in \Sigma^*$ such that $S \Rightarrow w$, and is denoted $[\![\mathcal{G}]\!]$. For simplicity, we assume the grammar does not derive the empty word.

Weighted grammars. A *weighted grammar* (WCFG) is a pair $\langle \mathcal{G}, \theta\rangle$ where $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S\rangle$ is a CFG and $\theta : R \to \mathbb{R}$ is a function mapping each production rule to a weight in $\mathbb{R}$. A WCFG $\mathcal{W} = \langle \mathcal{G}, \theta\rangle$ defines a function from words over Σ to weights in $\mathbb{R}$. The WCFG associates with a derivation tree t its weight, which is defined as

$$\mathcal{W}(t) = \prod_{(V \to \alpha) \in R} \theta(V \to \alpha)^{\sharp_t(V \to \alpha)}$$

[2]The definition of *convergent* is deferred to the preliminaries.

[3]The definition of the Hankel Matrix and its role in learning algorithms appears in the sequel.

where $\sharp_t(V \to \alpha)$ is the number of occurrences of the production $V \to \alpha$ in the derivation tree t. We abuse notation and treat $\mathcal{W}$ also as a function from Σ^* to $\mathbb{R}$ defined as $\mathcal{W}(w) = \sum_{S \Rightarrow_t w} \mathcal{W}(t)$. That is, the weight of w is the sum of weights of the derivation trees yielding w, and if $w \notin [\![\mathcal{G}]\!]$ then $\mathcal{W}(w) = 0$. If the sum of all derivation trees in $[\![\mathcal{G}]\!]$, namely $\sum_{w \in [\![\mathcal{G}]\!]} \mathcal{W}(w)$, is finite we say that $\mathcal{W}$ is *convergent*. Otherwise, we say that $\mathcal{W}$ is *divergent*.

Probabilistic grammars. A *probabilistic grammar* (PCFG) is a WCFG $\mathcal{P} = \langle \mathcal{G}, \theta \rangle$ where $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S \rangle$ is a CFG and $\theta : R \to [0,1]$ is a function mapping each production rule of $\mathcal{G}$ to a weight in the range $[0,1]$ that satisfies

$$1 = \sum_{(V \to \alpha_i) \in R} \theta(V \to \alpha_i)$$

for every $V \in \mathcal{V}$.[4] One can see that if $\mathcal{P}$ is a PCFG then the sum of all derivations equals 1, thus $\mathcal{P}$ is convergent.

Word and Tree Series. While *words* are defined as sequences over a given alphabet, trees are defined using a *ranked alphabet*, an alphabet $\Sigma = \{\Sigma_0, \Sigma_1, \ldots, \Sigma_p\}$ which is a tuple of alphabets Σ_k where Σ_0 is non-empty. Let $\mathit{Trees}(\Sigma)$ be the set of trees over Σ, where a node labeled $\sigma \in \Sigma_k$ for $0 \leq k \leq p$ has exactly k children. While a *word language* is a function mapping all possible words (elements of Σ^*) to $\{0,1\}$, a *tree language* is a function from all possible trees (elements of $\mathit{Trees}(\Sigma)$) to $\{0,1\}$. We are interested in assigning each word or tree a non-Boolean value, usually a weight $p \in [0,1]$. More generally, let $\mathbb{K}$ be an arbitrary domain, e.g. the real numbers. We are interested in functions mapping words or trees to values in $\mathbb{K}$. A function from Σ^* to $\mathbb{K}$ is called a *word series*, and a function from $\mathit{Trees}(\Sigma)$ to $\mathbb{K}$ is referred to as a *tree series*. CFGs define word languages, and induce tree languages (the parse trees deriving the words defined by the grammars). WCFGs and PCFGs define word series, where in the latter case the map is from Σ^* to $[0,1]$.

2.2. Multiplicity Tree Automata.

Word and Tree Automata. *Word automata* are machines that recognize word languages, i.e. they define a function from Σ^* to $\{0,1\}$. *Tree automata* are machines that recognize tree languages, i.e. they define a function from $\mathit{Trees}(\Sigma)$ to $\{0,1\}$. *Multiplicity word automata* (MA) are machines to implement word series $f : \Sigma^* \to \mathbb{K}$ where $\mathbb{K}$ is a field. *Multiplicity tree automata* (MTA) are machines to implement tree series $f : \mathit{Trees}(\Sigma) \to \mathbb{K}$ where $\mathbb{K}$ is a field.

Multiplicity Automata. Multiplicity automata can be thought of as an algebraic extension of automata, in which reading an input letter is implemented by matrix multiplication. In a multiplicity word automaton with dimension m over alphabet Σ, for each $\sigma \in \Sigma$ there is an m by m matrix, μ_σ, whose entries are values in $\mathbb{K}$ where intuitively the value of entry $\mu_\sigma(i,j)$ is the weight of the passage from state i to state j. The definition of multiplicity tree automata is a bit more involved; it makes use of multilinear functions as defined next.

[4]Probabilistic grammars are sometimes called *stochastic grammars (SCFGs).*

$$M_\eta = \begin{pmatrix} c^1_{111} & c^1_{112} & c^1_{121} & c^1_{122} & c^1_{211} & c^1_{212} & c^1_{221} & c^1_{222} \\ c^2_{111} & c^2_{112} & c^2_{121} & c^2_{122} & c^2_{211} & c^2_{212} & c^2_{221} & c^2_{222} \end{pmatrix} \qquad P_{xyz} = \begin{pmatrix} x_1y_1z_1 \\ x_1y_1z_2 \\ x_1y_2z_1 \\ \cdots \\ x_2y_2z_2 \end{pmatrix}$$

Figure 1. A matrix M_η for a multi-linear function η and a vector P_{xyz} for the respective 3 parameters.

Multilinear functions. Let $\mathbb{V} = \mathbb{K}^d$ be the d dimensional vector space over $\mathbb{K}$. Let $\eta : \mathbb{V}^k \to \mathbb{V}$ be a k-linear function. We can represent η by a d by d^k matrix over $\mathbb{K}$.

Example 2.1. For instance, if $\eta : \mathbb{V}^3 \to \mathbb{V}$ and $d = 2$ (i.e. $\mathbb{V} = \mathbb{K}^2$) then η can be represented by the 2×2^3 matrix M_η provided in Fig 1 where $c^i_{j_1j_2j_3} \in \mathbb{K}$ for $i, j_1, j_2, j_3 \in \{1, 2\}$. Then η, a function taking k parameters in $\mathbb{V} = \mathbb{K}^d$, can be computed by multiplying the matrix M_η with a vector for the parameters for η. Continuing this example, given the parameters $\mathbf{x} = (x_1 \;\; x_2)$, $\mathbf{y} = (y_1 \;\; y_2)$, $\mathbf{z} = (z_1 \;\; z_2)$ the value $\eta(\mathbf{x}, \mathbf{y}, \mathbf{z})$ can be calculated using the multiplication $M_\eta P_{xyz}$ where the vector P_{xyz} of size 2^3 is provided in Fig 1.

In general, if $\eta : \mathbb{V}^k \to \mathbb{V}$ is such that $\eta(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k) = \mathbf{y}$ and M_η, the matrix representation of η, is defined using the constants $c^i_{j_1j_2\cdots j_k}$ then

$$\mathbf{y}[i] = \sum_{\{(j_1,j_2,\ldots,j_k)\in\{1,2,\ldots,d\}^k\}} c^i_{j_1j_2\cdots j_k}\, \mathbf{x}_1[j_1]\, \mathbf{x}_2[j_2] \cdots \mathbf{x}_k[j_k]$$

Multiplicity tree automata. A *multiplicity tree automaton* (MTA) is a tuple $\mathcal{M} = (\Sigma, \mathbb{K}, d, \mu, \lambda)$ where $\Sigma = \{\Sigma_0, \Sigma_1, \ldots, \Sigma_p\}$ is the given ranked alphabet, $\mathbb{K}$ is a field corresponding to the range of the tree-series, d is a non-negative integer called the automaton *dimension*, μ and λ are the transition and output function, respectively, whose types are defined next. Let $\mathbb{V} = \mathbb{K}^d$. Then λ is an element of $\mathbb{V}$, namely a d-vector over $\mathbb{K}$. Intuitively, λ corresponds to the final values of the "states" of $\mathcal{M}$. The transition function μ maps each element σ of Σ to a dedicated transition function μ_σ such that given $\sigma \in \Sigma_k$ for $0 \le k \le p$ then μ_σ is a k-linear function from $\mathbb{V}^k$ to $\mathbb{V}$. The transition function μ induces a function from *Trees*(Σ) to $\mathbb{V}$, defined as follows. If $t = \sigma$ for some $\sigma \in \Sigma_0$, namely t is a tree with one node which is a leaf, then $\mu(t) = \mu_\sigma$ (note that μ_σ is a vector in $\mathbb{K}^d$ when $\sigma \in \Sigma_0$). If $t = \sigma(t_1, \ldots, t_k)$, namely t is a tree with root $\sigma \in \Sigma_k$ and children $t_1, \ldots, t_k$ then $\mu(t) = \mu_\sigma(\mu(t_1), \ldots, \mu(t_k))$. The automaton $\mathcal{M}$ induces a total function from *Trees*(Σ) to $\mathbb{K}$ defined as follows: $\mathcal{M}(t) = \lambda \cdot \mu(t)$.

Example 2.2. Fig. 2 on the left provides an example of an MTA $\mathcal{M} = ((\Sigma_0, \Sigma_2), \mathbb{R}, 2, \mu, \lambda)$ where $\Sigma_0 = \{a\}$ and $\Sigma_2 = \{b\}$ implementing a tree series that returns the number of leaves in the tree. Fig. 2 on the right provides a tree where a node t is annotated by $\mu(t)$. Since $\mu(t_\epsilon) = \left(\begin{smallmatrix}3\\1\end{smallmatrix}\right)$, where t_ϵ is the root, the value of the entire tree is $\lambda \cdot \left(\begin{smallmatrix}3\\1\end{smallmatrix}\right) = 3$.

2.3. **Contexts and Structured Trees.**

$$\lambda = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \mu_a = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\mu_b = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

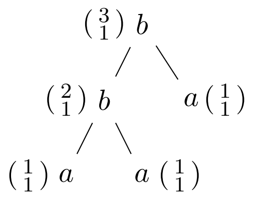


Figure 2. (Left) An MTA implementing a tree series that returns the number of leaves in the tree. (Right) a tree where a node t is annotated by $\mu(t)$.

Contexts. When learning word languages, learning algorithms typically distinguish words u and v if there exists a suffix z such that uz is accepted but vz is rejected or vice versa. When learning tree languages, in a similar way we would like to distinguish between two trees t_u and t_v, if there exists a tree t_z whose composition with t_u and t_v is accepted in one case and rejected in the other. To define this formally we need some notion to compose trees, more accurately we compose trees with *contexts* as defined next. Let $\Sigma = \{\Sigma_0, \Sigma_1, \ldots, \Sigma_p\}$ be a ranked alphabet. Let $\diamond$ be a symbol not in Σ. We use $\mathit{Trees}_\diamond(\Sigma)$ to denote all non-empty trees over $\Sigma' = \{\Sigma_0 \cup \{\diamond\}, \Sigma_1, \ldots, \Sigma_p\}$ in which $\diamond$ appears exactly once. Intuitively $\diamond$ indicates the place where a tree t_u can be composed with a context t_z yielding a tree that resembles t_z but has t_u as a sub-tree instead of the leaf $\diamond$. We refer to an element of $\mathit{Trees}_\diamond(\Sigma)$ as a *context*. Note that at most one child of any node in a context c is a context; the other ones are pure trees (i.e. elements of $\mathit{Trees}(\Sigma)$). Given a tree $t \in \mathit{Trees}(\Sigma)$ and context $c \in \mathit{Trees}_\diamond(\Sigma)$ we use $c[\![t]\!]$ for the tree $t' \in \mathit{Trees}(\Sigma)$ obtained from c by replacing $\diamond$ with t.

Structured tree languages/series. Recall that our motivation is to learn a word (string) series rather than a tree series, and due to hardness results on learning CFGs and PCFGs we resort to using *structured strings.* A *structured string* is a string with parentheses exposing the structure of a derivation tree for the corresponding trees, as exemplified in Fig. 3.

A *skeletal alphabet* is a ranked alphabet in which we use a special symbol $? \notin \Sigma_0$ and for every $0 < k \leq p$ the set Σ_k consists only of the symbol ?. For $t \in \mathit{Trees}(\Sigma)$, the skeletal description of t, denoted by $\mathsf{S}(t)$, is a tree with the same topology as t, in which the symbol in all internal nodes is ?, and the symbols in all leaves are the same as in t. Let T be a set of trees. The corresponding skeletal set, denoted $\mathsf{S}(T)$ is $\{\mathsf{S}(t) \mid t \in T\}$. Going from the other direction, given a skeletal tree s we use $\mathsf{T}(s)$ for the set $\{t \in \mathit{Trees}(\Sigma) \mid \mathsf{S}(t) = s\}$.

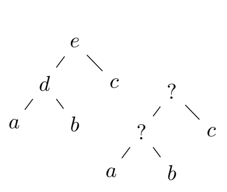


Figure 3. A derivation tree and its corresponding skeletal tree, which can be written as the structured string $((ab)c)$.

A tree language over a skeletal alphabet is called a *skeletal tree language.* And a mapping from skeletal trees to $\mathbb{K}$ is called a *skeletal tree series.* Let $\mathcal{T}$ denote a tree series mapping trees in $\mathit{Trees}(\Sigma)$ to $\mathbb{K}$. By abuse of notations, given a skeletal tree s, we use $\mathcal{T}(s)$ for the sum of values $\mathcal{T}(t)$ for every tree t of which $s = \mathsf{S}(t)$. That is, $\mathcal{T}(s) = \sum_{t \in \mathsf{T}(s)} \mathcal{T}(t)$. Thus, given a tree series $\mathcal{T}$ (possibly generated by a WCFG or an MTA) we can treat $\mathcal{T}$ as a skeletal tree series.

3. From Positive MTAs to PCFGs

Our learning algorithm for probabilistic grammars builds on the relation between WCFGs with positive weights (henceforth PWCFGs) and PCFGs [AMP99, SJ07]. In particular, we first establish that a *positive multiplicity tree automaton* (PMTA), which is a multiplicity tree automaton (MTA) where all weights of both μ and λ are positive, can be transformed into an equivalent WCFG $\mathcal{W}$. That is, we show that a given PMTA $\mathcal{A}$ over a skeletal alphabet can be converted into a WCFG $\mathcal{W}$ such that for every structured string s we have that $\mathcal{A}(s) = \mathcal{W}(s)$. If the PMTA defines a convergent tree series (namely the sum of weights of all trees is finite) then so will the constructed WCFG. Therefore, given that the WCFG describes a probability distribution, we can apply the transformation of WCFG to a PCFG [AMP99, SJ07] to yield a PCFG $\mathcal{P}$ such that $\mathcal{W}(s) = \mathcal{P}(s)$, obtaining the desired PCFG for the unknown tree series.

3.1. **Transforming a PMTA into a PWCFG.** Let $\mathcal{A} = (\Sigma, \mathbb{R}_+, d, \mu, \lambda)$ be a PMTA over the skeletal alphabet $\Sigma = \{\Sigma_0, \Sigma_1, \ldots, \Sigma_p\}$. We define a PWCFG $\mathcal{W}_\mathcal{A} = (\mathcal{G}_\mathcal{A}, \theta)$ for $\mathcal{G}_\mathcal{A} = (\mathcal{V}, \Sigma_0, R, S)$ as provided in Fig. 4 where $c^i_{i_1,i_2,\ldots,i_k}$ is the respective coefficient in the matrix corresponding to $\mu_?$ for $? \in \Sigma_k$, $1 \le k \le p$.

$$
\begin{array}{rll}
\mathcal{V} = & \{S\} \cup \{V_i \mid 1 \le i \le d\} & \\
R = & \{S \to V_i \mid 1 \le i \le d\} \cup & \theta(S \to V_i) = \lambda[i] \\
& \{V_i \to \sigma \mid 1 \le i \le d,\ \sigma \in \Sigma_0\} \cup & \theta(V_i \to \sigma) = \mu_\sigma[i] \\
& \{V_i \to V_{i_1} V_{i_2} ... V_{i_k} \mid 1 \le i, i_1, \ldots, i_k \le d,\ 1 \le k \le p\} & \theta(V_i \to V_{i_1} V_{i_2} ... V_{i_k}) = c^i_{i_1,i_2,\ldots,i_k}
\end{array}
$$

Figure 4. Transforming a PMTA into a PCFG

Example 3.1. Let $\mathcal{A} = \langle \Sigma, \mathbb{R}_+, 2, \mu, \lambda \rangle$ where $\Sigma = \{\Sigma_0 = \{a, b\} \cup \Sigma_2 = \{?\}\}$, and

$$
\lambda = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \mu_a = \begin{pmatrix} 0.25 \\ 0.25 \end{pmatrix} \quad \mu_b = \begin{pmatrix} 0.25 \\ 0.0 \end{pmatrix} \quad \mu_? = \begin{pmatrix} 0.25 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0.75 \end{pmatrix}.
$$

After applying the transformation we obtain the following PCFG:

$$
\begin{aligned}
S &\longrightarrow N_1\ [1.0] \\
N_1 &\longrightarrow N_1 N_1\ [0.25] \mid N_1 N_2\ [0.25] \mid a\ [0.25] \mid b\ [0.25] \\
N_2 &\longrightarrow N_2 N_2\ [0.75] \mid a\ [0.25]
\end{aligned}
$$

Proposition 3.1 states that the transformation preserves the weights.

Proposition 3.1. $\mathcal{W}(t) = \mathcal{A}(t)$ *for every* $t \in$ *Trees*(Σ).

Recall that given a WCFG $\langle \mathcal{G}, \theta \rangle$, and a tree that can be derived from $\mathcal{G}$, namely some $t \in \mathsf{T}(\mathcal{G})$, the weight of t is given by $\theta(t)$. Recall also that we are working with skeletal trees $s \in \mathsf{S}(\mathsf{T}(\mathcal{G}))$ and the weight of a skeletal tree s is given by the sum of all derivation trees t such that s is the skeletal tree obtained from t by replacing all non-terminals with ?.

The following two lemmas and the following notations are used in the proof of Proposition 3.1. For a skeletal tree s and a non-terminal V we use $\mathcal{W}_V(s)$ for the weight of all derivation trees t in which the root is labeled by non-terminal V and s is their skeletal form.

Assume $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S \rangle$. Lemma 3.2 follows in a straightforward manner from the definition of $\mathcal{W}(\cdot)$ given in \$2.1.

Lemma 3.2. *Let $s = ?(s_1, s_2, \ldots, s_k)$. The following holds for every non-terminal $V \in \mathcal{V}$:*

$$\mathcal{W}_V(s) = \sum_{(X_1,X_2,\ldots,X_k)\in\mathcal{V}^k} \theta(V \to X_1X_2\cdots X_k) \cdot \mathcal{W}_{X_1}(s_1)\mathcal{W}_{X_2}(s_2)\cdots\mathcal{W}_{X_k}(s_k)$$

Consider now the transformation from a PMTA to a WCFG (provided in Fig. 4). It associates with every dimension i of the PMTA $\mathcal{A} = (\Sigma, \mathbb{R}_+, d, \mu, \lambda)$ a variable (i.e. non-terminal) V_i. The next lemma considers the d-dimensional vector $\mu(s)$ computed by $\mathcal{A}$ and states that its i-th coordinate holds the value $\mathcal{W}_{V_i}(s)$.

Lemma 3.3. *Let s be a skeletal tree, and let $\mu(s) = (v_1, v_2, \ldots, v_d)$. Then $v_i = \mathcal{W}_{V_i}(s)$ for every $1 \leq i \leq d$.*

Proof. The proof is by induction on the height of s. For the base case $h = 1$. Then s is a leaf, thus $s \in \Sigma$. Then for each i we have that $v[i] = \mu(\sigma)[i]$ by definition of MTA computation. On the other hand, by the definition of the transformation in Fig. 4, we have $\theta(V_i \to \sigma) = \mu(\sigma)[i]$. Thus, $\mathcal{W}_{V_i}(s) = \mathcal{W}_{V_i}(\sigma) = \mu(\sigma)[i]$, so the claim holds.

For the induction step, assume $s = ?(s_1, s_2, ..., s_k)$. By the definition of a multi-linear map, for each i we have:

$$v_i = \sum_{\{(j_1,j_2,\ldots,j_k)\in\{1,2,\ldots,d\}^k\}} c^i_{j_1,\ldots,j_k} v_i[j_1] \cdot \ldots \cdot v_k[j_k]$$

where $c^i_{j_1,\ldots,j_k}$ are the coefficients of the $d \times d^k$ matrix of $\mu_?$ for $? \in \Sigma_k$. By the definition of the transformation in Fig. 4 we have that $c^i_{j_1,\ldots,j_k} = \theta(V_i \to V_{j_1}V_{j_2}...V_{j_k})$. Also, from our induction hypothesis, we have that for each j_i, $v_i[j_i] = \mathcal{W}_{V_{j_i}}(s_i)$. Therefore, we have that:

$$v_i = \sum_{V_{j_1}V_{j_2}...V_{j_p}\in\mathcal{V}^k} \theta(V_i \to V_{j_1}V_{j_2}...V_{j_k}) \cdot \mathcal{W}_{V_{j_1}}(s_1) \cdot \ldots \cdot \mathcal{W}_{V_{j_k}}(s_k)$$

which according to Lemma 3.2 is equal to $\mathcal{W}_{V_i}(s)$ as required. □

We are now ready to prove Proposition 3.1.

Proof of Prop.3.1. Let $\mu(t) = v = (v_1, v_2, ..., v_n)$ be the vector calculated by $\mathcal{A}$ for t. The value calculated by $\mathcal{A}$ is $\lambda \cdot v$, which is:

$$\sum_{i=1}^{n} v_i \cdot \lambda[i]$$

By the transformation in Fig. 4 we have that $\lambda[i] = \theta(S \to V_i)$ for each i. So we have:

$$\sum_{i=1}^{n} v_i \cdot \lambda[i] = \theta(S \to V_i) \cdot v_i$$

By Lemma 3.3, for each i, v_i is equal to the probability of deriving t starting from the non-terminal V_i, so we have that the value calculated by $\mathcal{A}$ is the probability of deriving the tree starting from the start symbol S. That is, $\mathcal{W}(t) = \mathcal{W}_S(t) = \mathcal{A}(t)$. □

3.2. **Structurally Unambiguous WCFGs and PCFGs.** In this paper we consider *structurally unambiguous* WCFGs and PCFGs (in short SUWCFGs and SUPCFGs, resp.) as defined in the sequel in §4.2. In Thm. 5.1, given in §5, we show that we can learn a PMTA for a SUWCFG in polynomial time using a polynomial number of queries (see exact bounds there), thus obtaining the following result.

Corrolary 3.4. *SUWCFGs can be learned in polynomial time using* SMQ*s and* SEQ*s, where the number of* SEQ*s is bounded by the number of non-terminal symbols.*

The overall learning time for SUPCFG relies, on top of Corollary 3.4, on the complexity of converting a WCFG into a PCFG [AMP99], for which an exact bound is not provided, but the method is reported to converge quickly [SJ07, §2.1].

4. Learning of Structurally Unambiguous Probabilistic Grammars

In this section we discuss the setting of the algorithm, the ideas behind Angluin-style learning algorithms, and the issues with using current algorithms to learn PCFGs. As in $\mathbf{G}^*$(the algorithm for CFGs [Sak88]), we assume an oracle that can answer two types of queries: *structured membership queries* (SMQ) and *structured equivalence queries* (SEQ) regarding the unknown regular tree series $\mathcal{T}$ (over a given ranked alphabet Σ). Given a structured string s, the query SMQ(s) is answered with the value $\mathcal{T}(s)$. Given an automaton $\mathcal{A}$ the query SEQ($\mathcal{A}$) is answered "yes" if $\mathcal{A}$ implements the skeletal tree series $\mathcal{T}$ and otherwise the answer is a pair $(s, \mathcal{T}(s))$ where s is a structured string for which $\mathcal{T}(s) \neq \mathcal{A}(s)$ (up to a predefined error).

Our starting point is the learning algorithm $\mathbf{M}^*$ [HO06] which learns MTA using SMQs and SEQs. First we explain the idea behind this and similar algorithms, next the issues with applying it as is for learning PCFGs, then the idea behind restricting attention to strucutrally unambiguous grammars, and finally our algorithm itself.

Hankel Matrix. The *Hankel Matrix* is a key concept in learning of languages and formal series. A word or tree language as well as a word or tree series can be represented by its Hankel Matrix. The Hankel Matrix has infinitely many rows and infinitely many columns. In the case of word series both rows and columns correspond to an infinite enumeration $w_0, w_1, w_2, \ldots$ of words over the given alphabet. In the case of tree series, the rows correspond to an infinite enumeration of trees $t_0, t_1, t_2, \ldots$ (where $t_i \in \mathit{Trees}(\Sigma)$) and the columns to an infinite enumeration of contexts $c_0, c_1, c_2, \ldots$ (where $c_i \in \mathit{Trees}_\diamond(\Sigma)$). In the case of words, the entry $H(i, j)$ holds the value for the word $w_i \cdot w_j$, and in the case of trees it holds the value of the tree $c_j[\![t_i]\!]$. If the series is *regular* there should exists a finite number of rows in this infinite matrix, which we term *basis*, such that all other rows can be represented using rows in the basis. In the case of $\mathbf{L}^*$, and $\mathbf{G}^*$(that learn word-languages and tree-languages, resp.) rows that are not in the basis should be equal to rows in the basis. The rows of the basis correspond to the automaton states, and the equalities to other rows determines the transition relation. In the case of $\mathbf{M}^*$(that learns tree-series by means of multiplicity tree automata) and its predecessor (that learns word-series using multiplicity word automata) rows not in the basis should be expressible as a linear combination of rows in the basis, and the linear combinations determines the weights of the extracted automaton.

Example 4.1. Fig. 5 depicts the Hankel Matrix for the tree-series that returns the number of leaves in a tree over the ranked alphabet $\Sigma = \{\Sigma_0 = \{a\}, \Sigma_2 = \{b\}\}$. The first two rows

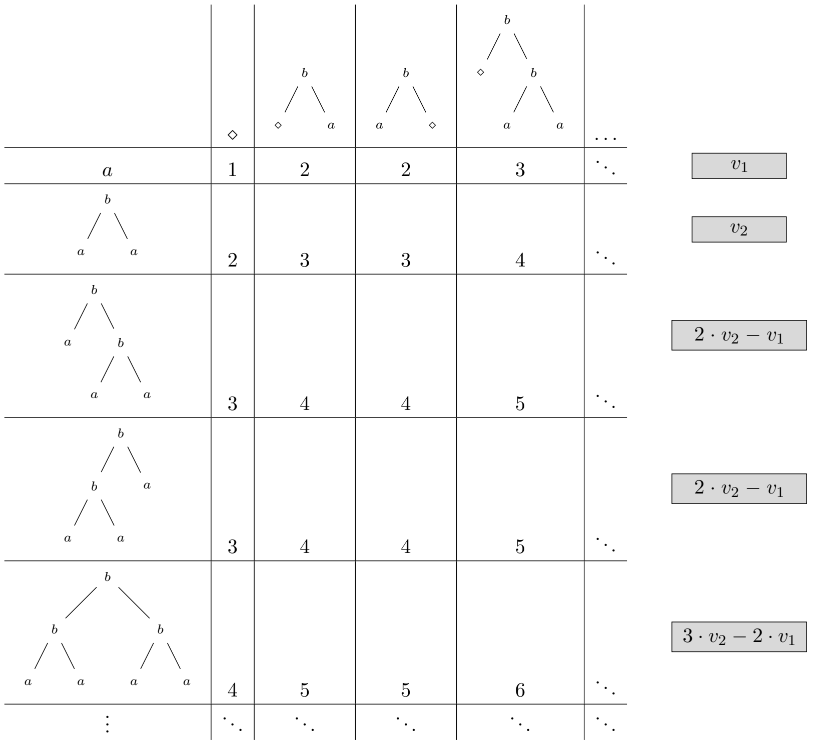
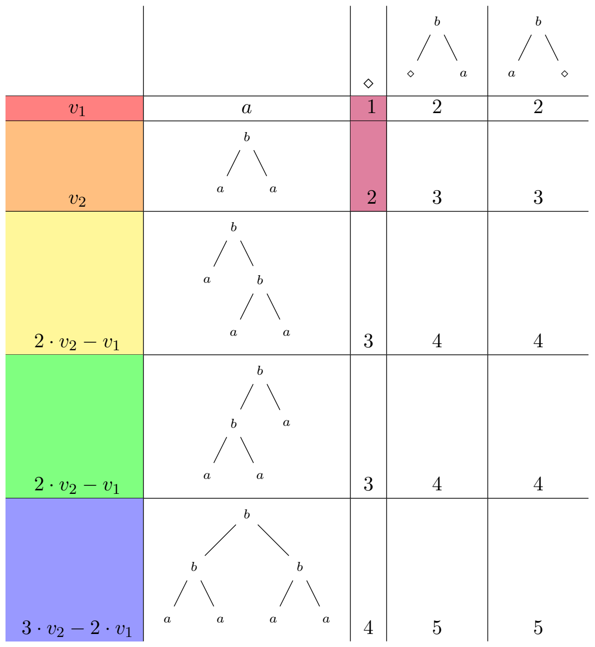

v_1
v_2
$2 \cdot v_2 - v_1$
$2 \cdot v_2 - v_1$
$3 \cdot v_2 - 2 \cdot v_1$

Figure 5. The Hankel Matrix for the tree-series that returns the number of leaves in a tree over the ranked alphabet $\Sigma = \{\Sigma_0, \Sigma_2\}$ where $\Sigma_0 = \{a\}$ and $\Sigma_2 = \{b\}$.

linearly span the entire matrix. Let $H[a] = v_1$ and $H[b(a, a)] = v_2$. The rest of the rows can be described as a linear combination of v_1 and v_2, as shown on the right in gray.

L*-style query learning algorithms. All the generalizations of $\mathbf{L}^*$ share a general idea that can be explained as follows. The algorithm maintains an *observation table*, a finite sub-matrix of the Hankel Matrix, whose entries are filled by asking membership queries. Once the table meets certain criteria, namely is *closed* with respect to a *basis*, which is a subset of the rows, an automaton can be extracted from it. In the case of $\mathbf{L}^*$ (that learns regular word-languages) a table is closed if the row of the empty string is in the basis, for every row s in the basis, its one letter extension, $s\sigma$ is in the rows of the table, and every row in the table is equivalent to some row in the basis. In the case of $\mathbf{C}^*$(that learns tree automata) instead of one letter extensions, we need for every letter $\sigma \in \Sigma_k$, a row with root σ and for the k-children all options of trees from the basis. The criterion for a row not in the basis to be covered, is the same; it should be equivalent to a row in the basis. In the case of $\mathbf{M}^*$, the extension requirement is as for $\mathbf{C}^*$, since we are dealing with trees. As for covering the rows of the table that are not in the basis, the requirement is that a rows that is not in the basis be expressible as a linear combination of rows in the basis.

In the case of $\mathbf{L}^*$, the extracted automaton will have one state for every row in the basis, where ϵ is the initial state, and a state s is determined to be final if the observation table entry corresponding to row s and column ϵ is labeled 1 (i.e. the word s is in the language).



$$\mu_a = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\mu_b = \begin{bmatrix} 0 & -1 & -1 & -2 \\ 1 & 2 & 2 & 3 \end{bmatrix}$$

$$\lambda = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Figure 6. A closed observation table (on the left) and the MTA extracted from it (on the right).

Then, the transition from s on σ is determined to be the row s' of the basis that is equivalent to $s\sigma$.

In the case of $\mathbf{M}^*$ the size of the basis determines the dimension of the extracted multiplicity tree automaton. Suppose the size of the basis is d. The output vector λ is set to $(c_1, \ldots, c_d)$ where c_i is the value of the entry corresponding to the row i of the basis and the column $\diamond$ (i.e the value of the i-th row of the basis). Consider a letter $\sigma \in \Sigma_k$, its corresponding transition matrix μ_σ is a $d \times d^k$ matrix. Think of μ_σ as d^k d-vectors $(v_{11\ldots1}\ v_{11\ldots2}\ \ldots\ v_{dd\ldots d})$; then the vector $v_{j_1 j_2 \ldots j_k}$ which corresponds to the row extending σ with the trees in the base rows $r_{j_1}, r_{j_2} \ldots r_{j_k}$ gets its coefficient from the linear combination for this row, see Example 4.2.

Note that in all cases we would like the basis to be *minimal* in the sense that no row in the basis is expressible using other rows in the basis. This is since the size of the basis derives the dimension of the automaton, and obviously we prefer smaller automata.

Example 4.2. Fig. 6 shows (on the left) a closed observation table which is a sub-matrix of the Hankel Matrix of Fig. 5, and the MTA extracted from it (on the right). The base here consists of the first two rows (thus the dimension d is 2) and as can be seen they linearly span the other rows of the table. The coefficients with which they span the table appear on the left-most column. The final vector λ is set to $(\begin{smallmatrix} 1 \\ 2 \end{smallmatrix})$ since these are the values of the row basis in column $\diamond$. The matrix μ_a is a 2×2^0 matrix since $a \in \Sigma_0$. We view it as a single vector that corresponds to the row of the letter a, it is thus $(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix})$. The matrix μ_b is a 2×2^2 matrix since $b \in \Sigma_2$. We view it as the vectors $(v_{11}\ v_{12}\ v_{21}\ v_{22})$. Consider for instance v_{21}. It corresponds to the tree with root b whose left child is the second base row, namely the tree on the second row of the observation table, and whose right child is the first base row, namely a. It thus corresponds to the 4th row of the table and is therefore set to $\left(\begin{smallmatrix}-1\\2\end{smallmatrix}\right)$. The other vectors of μ_b are set similarly (follow the color coding in Fig. 6).

Back to the problem at hand, since we are interested in probabilistic languages, we would like to apply an $\mathbf{L}^*$-style algorithm to obtain a PMTA, so that we can convert it into a PCFG. Since the coefficients of the linear combination in rows that are not in the basis determine the values of the entries of the transitions matrix, we cannot have negative coefficients. That is, we need the algorithm to find a subset of the rows (a basis), so that all other rows can be obtained by a *positive linear combination* of the rows in the basis, so that the extracted automaton will be a PMTA.

Positive linear spans. An interest in *positive linear combinations* occurs also in the research community studying convex cones and derivative-free optimizations and a theory of positive linear combinations has been developed [CR93, Reg16].[5] We need the following definitions and results.

The *positive span* of a finite set of vectors $S = \{v_1, v_2, ..., v_k\} \subseteq \mathbb{R}^n$ is defined as follows:

$$span_+(S) = \{\lambda_1 \cdot v_1 + \lambda_2 \cdot v_2 + ... + \lambda_k \cdot v_k \mid \lambda_i \geq 0, \forall 1 \leq i \leq k\}$$

A set of vectors $S = \{v_1, v_2, ..., v_k\} \subseteq \mathbb{R}^n$ is *positively dependent* if some v_i is a positive combination of the other vectors; otherwise, it is *positively independent*. Let $A \in \mathbb{R}^{m\times n}$. We say that A is *nonnegative* if all of its elements are nonnegative. The *nonnegative column (resp. row) rank* of A, denoted *c-rank*$_+(A)$ (resp. *r-rank*$_+(A)$), is defined as the smallest nonnegative integer q for which there exist a set of column- (resp. row-) vectors $V = \{v_1, v_2, ..., v_q\}$ in $\mathbb{R}^m$ such that every column (resp. row) of A can be represented as a positive combination of V. It was shown that *c-rank*$_+(A) =$ *r-rank*$_+(A)$ for any matrix A [CR93]. Thus one can freely use *rank*$_+(A)$ for *positive rank*, to refer to either one of these.

4.1. Trying to Accommodate M* for learning PMTA.

The first question that comes to mind, is whether we can use the $\mathbf{M}^*$ algorithm as is in order to learn a positive tree series. We show that this is not the case. We prove two propositions, which show that one cannot use the $\mathbf{M}^*$algorithm to learn PMTA. Proposition 4.3 shows a grammar, for which there exist a PMTA, and a positive base in the Hankel Matrix, but applying the $\mathbf{M}^*$algorithm as is may choose a non-negative base.

Proposition 4.3. *There exists a probabilistic word series for which the* $\boldsymbol{M}^*$ *algorithm may return an MTA with negative weights.*

Proof. Let $\mathcal{G}$ be the grammar that assigns to the words *aa*,*ab*,*ac*,*ba*,*cb*,*cc* a probability of $\frac{1}{6}$ and to all other words a probability of 0. For simplicity we assume that the grammar is regular. The first rows of Hankel Matrix for this word series are given in Fig.7 (all entries not in the figure are 0). One can see that the rows ϵ, b, c, ba (highlighted in light gray) are a positive span of the entire Hankel Matrix. However, the $\mathbf{M}^*$ algorithm may return the MTA spanned by the basis ϵ, a, b, aa (highlighted in dark gray). Since the row of c is obtained by subtracting the row of b from the row of a, the resulting MTA will contain negative weights. □

Proposition 4.5 strengthens this and shows that there are grammars which have PMTA, but $\mathbf{M}^*$ would *always* return a non-positive MTA. This is since the Hankel Matrix $H_{\mathcal{G}}$

[5]Throughout the paper we use the terms *positive* and *nonnegative* interchangeably.

	ε	a	b	c	aa	ab	ac	ba	bb	bc	ca	cb	cc
ε	0	0	0	0	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	0	$\frac{1}{6}$	$\frac{1}{6}$
a	0	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	0	0	0	0	0	0	0
b	0	$\frac{1}{6}$	0	0	0	0	0	0	0	0	0	0	0
c	0	0	$\frac{1}{6}$	$\frac{1}{6}$	0	0	0	0	0	0	0	0	0
aa	$\frac{1}{6}$	0	0	0	0	0	0	0	0	0	0	0	0
ab	$\frac{1}{6}$	0	0	0	0	0	0	0	0	0	0	0	0
ac	$\frac{1}{6}$	0	0	0	0	0	0	0	0	0	0	0	0
ba	$\frac{1}{6}$	0	0	0	0	0	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0	0	0	0	0	0	0
bc	0	0	0	0	0	0	0	0	0	0	0	0	0
ca	0	0	0	0	0	0	0	0	0	0	0	0	0
cb	$\frac{1}{6}$	0	0	0	0	0	0	0	0	0	0	0	0
cc	$\frac{1}{6}$	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7. The observation table for the grammar $\mathcal{G}$. Two of the possible bases are highlighted. The dark gray one results in a negative MTA, while the light gray one results in a positive one, that can be converted to a PCFG. For simplicity, the trees are presented as words, since we assume that the grammar is regular.

corresponding to the tree-series $\mathcal{T}_G$ of the respective PCFG $\mathcal{G}$ has the property that no finite number of rows positively spans the entire matrix.

Before proving Proposition 4.5, we prove the following lemma:

Lemma 4.4. *Let $B = \{b_1, b_2, ..., b_p\}$ be a set of positively independent vectors. Let $\hat{B}$ be a matrix whose columns are the elements of B, and let α be a positive vector. Then if $\hat{B}\alpha = b_i$ then $\alpha[i] = 1$ and $\alpha[j] = 0$ for every $j \neq i$.*

Proof. Assume $b_i = \hat{B}\alpha$. Then we have:

$$b_i = \hat{B}\alpha = \sum_{j=1}^{p} \alpha[j]b_j = \alpha[i]b_i + \sum_{j\neq i} \alpha[j]b_j$$

If $\alpha[i] < 1$ we obtain:

$$b_i(1 - \alpha[i]) = \sum_{j\neq i} \alpha[j]b_j$$

Which is a contradiction, since $b_i \in B$ and thus can't be described as a positive combination of the other elements.

If $\alpha[i] > 1$ we obtain:

$$\sum_{j\neq i} \alpha[j]b_j + (\alpha[i] - 1)b_i = 0$$

This is a contradiction, since all b's are positive, all $\alpha[j] \geq 0$ and $\alpha[i] > 1$.

Hence $\alpha[i] = 1$. Therefore we have:

$$\sum_{j\neq i} \alpha[j]b_j = 0$$

Since $\alpha[j] \geq 0$, and b_j is positive the only solution is that $\alpha[j] = 0$ for every $j \neq i$. □

We turn to ask whether we can adjust the algorithm $\mathbf{M}^*$ to learn a positive basis. We note first that working with positive spans is much trickier than working with general spans, since for $d \geq 3$ there is no bound on the size of a positively independent set in $\mathbb{R}_+^d$ [Reg16]. To apply the ideas of the Angluin-style query learning algorihtms we need the Hankel Matrix (which is infinite) to contain a finite sub-matrix with the same rank. Unfortunately, as we show next, there exists a probabilistic (thus positive) tree series $\mathcal{T}$ that can be recognized by a PMTA, but none of its finite-sub-matrices span the entire space of $H_{\mathcal{T}}$.

Proposition 4.5. *There exists a PCFG $\mathcal{G}$ s.t. for the Hankel Matrix $H_{\mathcal{G}}$ corresponding to its tree-series $\mathcal{T}_G$ no finite number of rows positively spans the entire matrix.*

Proof. Let $\mathcal{G} = (\{a\}, \{N_1, N_2\}, R, N_1)$ be the following PCFG:

$$N_1 \longrightarrow aN_1\ [\tfrac{1}{2}]\ \mid\ aN_2\ [\tfrac{1}{3}]\ \mid\ aa\ [\tfrac{1}{6}]$$
$$N_2 \longrightarrow aN_1\ [\tfrac{1}{4}]\ \mid\ aN_2\ [\tfrac{1}{4}]\ \mid\ aa\ [\tfrac{1}{2}]$$

We say that a tree has a *chain structure* if every inner node is of branching-degree 2 and has one child which is a terminal. We say that a tree has a *right-chain structure* (resp. *left-chain structure*) if the non-terminal is always the right (resp. left) child. Note that all trees in $[\![\mathcal{G}]\!]$ have a right-chain structure, and the terminals are always the letter a.

Let us denote by p_n the total probability of all trees with n non-terminals s.t. the lowest non terminal is N_1, and similarly, let us denote by q_n the total probability of all trees with n non-terminals s.t. the lowest non-terminal is N_2.

We have that $p_0 = 0$, $p_1 = \frac{1}{6}$, and $p_2 = \frac{1}{12}$. We also have that $q_0 = 0$, $q_1 = 0$ and $q_2 = \frac{1}{6}$.

Now, to create a tree with n non-terminals, we should take a tree with $n-1$ non-terminals, ending with either N_1 or N_2, and use the last derivation. So we have:

$$p_n = \frac{1}{2} \cdot p_{n-1} + \frac{1}{4} \cdot q_{n-1}$$
$$q_n = \frac{1}{3} \cdot p_{n-1} + \frac{1}{4} \cdot q_{n-1}$$

We want to express p_n only as a function of p_i for $i < n$, and similarly for q_n. Starting with the first equation we obtain:

$$\begin{aligned} p_n &= \frac{1}{2} \cdot p_{n-1} + \frac{1}{4} \cdot q_{n-1} \\ 4 \cdot p_{n+1} - 2 \cdot p_n &= q_n \\ 4 \cdot p_n - 2 \cdot p_{n-1} &= q_{n-1} \end{aligned}$$

And from the second equation we obtain:

$$\begin{aligned} q_n &= \frac{1}{3} \cdot p_{n-1} + \frac{1}{4} \cdot q_{n-1} \\ 3 \cdot q_{n+1} - \frac{3}{4} \cdot q_n &= p_n \\ 3 \cdot q_n - \frac{3}{4} \cdot q_{n-1} &= p_{n-1} \end{aligned}$$

Now setting these values in each of the equation, we obtain:

$$\begin{aligned}
4 \cdot p_{n+1} - 2 \cdot p_n &= \frac{1}{3} \cdot p_{n-1} + \frac{1}{4}(4 \cdot p_n - 2 \cdot p_{n-1}) \\
p_{n+1} &= \frac{1}{2} \cdot p_n + \frac{1}{12} \cdot p_{n-1} + \frac{1}{16}(4 \cdot p_n - 2 \cdot p_{n-1}) \\
p_{n+1} &= \frac{1}{2} \cdot p_n + \frac{1}{12} \cdot p_{n-1} + \frac{1}{4} \cdot p_n - \frac{1}{8} \cdot p_{n-1} \\
p_{n+1} &= \frac{3}{4} \cdot p_n - \frac{1}{24} \cdot p_{n-1}
\end{aligned}$$

And:

$$\begin{aligned}
3 \cdot q_{n+1} - \frac{3}{4} \cdot q_n &= \frac{1}{2} \cdot (3 \cdot q_n - \frac{3}{4} \cdot q_{n-1}) + \frac{1}{4} \cdot q_{n-1} \\
3 \cdot q_{n+1} &= \frac{9}{4} \cdot q_n - \frac{1}{8} \cdot q_{n-1} \\
q_{n+1} &= \frac{9}{12} \cdot q_n - \frac{1}{24} \cdot q_{n-1} = \frac{3}{4} \cdot q_n - \frac{1}{24} \cdot q_{n-1}
\end{aligned}$$

let us denote by t_n the probability that $\mathcal{G}$ assigns to a^n. This probability is:

$$t_n = \frac{1}{6} \cdot p_{n-1} + \frac{1}{2} \cdot q_{n-1}$$

Since

$$p_{n-1} = \frac{3}{4} \cdot p_{n-2} - \frac{1}{24} \cdot p_{n-3}$$

and

$$q_{n-1} = \frac{3}{4} \cdot q_{n-2} - \frac{1}{24} \cdot q_{n-3}$$

we obtain:

$$\begin{aligned}
t_n =& \frac{1}{6} \cdot (\frac{3}{4} \cdot p_{n-2} - \frac{1}{24} \cdot p_{n-3}) + \frac{1}{2} \cdot (\frac{3}{4} \cdot q_{n-2} - \frac{1}{24} \cdot q_{n-3}) \\
t_n =& \frac{1}{6} \cdot \frac{3}{4} \cdot p_{n-2} - \frac{1}{6} \cdot \frac{1}{24} \cdot p_{n-3} + \frac{1}{2} \cdot \frac{3}{4} \cdot q_{n-2} - \frac{1}{2} \cdot \frac{1}{24} \cdot q_{n-3} \\
t_n =& \frac{3}{4} \cdot (\frac{1}{6} \cdot p_{n-2} + \frac{1}{2} \cdot q_{n-2}) - \frac{1}{24} \cdot (\frac{1}{6} \cdot p_{n-3} + \frac{1}{2} \cdot q_{n-3}) = \\
=& \frac{3}{4} \cdot t_{n-1} - \frac{1}{24} \cdot t_{n-2}
\end{aligned}$$

Hence, overall, we obtain:

$$t_n = \frac{3}{4} \cdot t_{n-1} - \frac{1}{24} \cdot t_{n-2}$$

Now, let L be the skeletal-tree-language of the grammar $\mathcal{G}$, and let H be the Hankel Matrix for this tree set. Note, that any tree t whose structure is not a right-chain, would have $L(t) = 0$, and also for every context c, $L(c[\![t]\!]) = 0$. Similarly, every context c who violates the right-chain structure, would have $L(c[\![t]\!]) = 0$ for every t.

Let T_n be the skeletal tree for the tree of right-chain structure, with n leaves. We have that $L(T_1) = 0$, $L(T_2) = \frac{1}{6}$, $L(T_3) = \frac{1}{4}$, and for every $i > 3$ we have

$$L(T_i) = \frac{3}{4} \cdot L(T_{i-1}) - \frac{1}{24} \cdot L(T_{i-2}).$$

Let v_i be the infinite row-vector of the Hankel matrix corresponding to T_i. We have that for every $i > 3$,

$$v_i = \frac{3}{4} \cdot v_{i-1} - \frac{1}{24} \cdot v_{i-2}.$$

Assume towards contradiction that there exists a subset of rows that is a positive base and spans the entire matrix H.

Let B be the positive base, whose highest member (in the lexicographic order) is the lowest among all the positive bases. Let v_r be the row vector for the highest member in this base. Thus, $v_{r+1} \in \mathit{span}_+(B)$. Hence:

$$v_{r+1} = \alpha \hat{B}$$

Also, $v_{r+1} = \frac{3}{4} \cdot v_r - \frac{1}{24} \cdot v_{r-1}$. Therefore,

$$\frac{3}{4} \cdot v_r - \frac{1}{24} \cdot v_{r-1} = \alpha \hat{B}$$

$$v_r = \frac{4}{3} \cdot \alpha \hat{B} + \frac{1}{18} \cdot v_{r-1}$$

We will next show that v_{r-1} and v_r are co-linear, which contradicts our choice of v_r. Since $v_{r-1} \in \mathit{span}_+(B)$ we know $v_{r-1} = \alpha' \hat{B}$ for some α'. Therefore,

$$v_r = (\frac{4}{3} \cdot \alpha + \frac{1}{18} \cdot \alpha')\hat{B}$$

Let $\beta = \frac{4}{3} \cdot \alpha + \frac{1}{18} \cdot \alpha'$. Since α and α' are non-negative vectors, so is β. And by Lemma 4.4 it follows that for every $i \neq r$:

$$\beta_i = \frac{4 \cdot \alpha_i}{3} + \frac{\alpha'_i}{18} = 0$$

Since α_i and α'_i are non-negative, we have that $\alpha_i = \alpha'_i = 0$.

Since for every $i \neq r$, $\alpha'_i = 0$, it follows that $v_{r-1} = m \cdot v_r$ for some $m \in \mathbb{R}$. Now, m can't be zero since our language is strictly positive and all entries in the matrix are non-negative. Thus, $v_r = \frac{1}{m} \cdot v_{r-1}$, and v_r and v_{r-1} are co-linear. We can replace v_r by v_{r-1}, contradicting the fact that we chose the base whose highest member is as low as possible. □

We note that a similar negative result was independently obtained in [vHKRS20] which studies the learnability of weighted automata, using an $\mathbf{L}^*$ algorithm, over algebraic structures different than fields. It was shown that weighted automata over the semiring of natural numbers are not learnable in the $\mathbf{L}^*$ framework. The formal series used in the proof is $f(a^n) = 2^n - 1$. Since this series is divergent it does not characterize a probabilistic automata/grammars. Proposition 4.5 strengthens this result as it provides a convergent series that is not $\mathbf{L}^*$ learnable.

4.2. **Focusing on Structurally Unambiguous CFGs.** To overcome these obstacles we restrict attention to structurally unambiguous CFGs (SUCFGs) and their weighted/probabilistic versions (SUWCFGs/SUPCFGs). A context-free grammar is termed *ambiguous* if there exists more than one derivation tree for the same word. We term a CFG *structurally ambiguous* if there exists more than one derivation tree with the same structure for the same word. A context-free language is termed *inherently ambiguous* if it cannot be derived by an unambiguous CFG. Note that a CFG which is unambiguous is also structurally unambiguous, while the other direction is not necessarily true. For instance, the language

$\{a^nb^nc^md^m \mid n \geq 1, m \geq 1\} \cup \{a^nb^mc^md^n \mid n \geq 1, m \geq 1\}$ which is inherently ambiguous [HU79, Thm. 4.7] is not inherently structurally ambiguous. Therefore we have relaxed the classical unambiguity requirement.

The Hankel Matrix and MTA for SUPCFG. Recall that the Hankel Matrix considers skeletal trees. Therefore if a word has more than one derivation tree with the same structure, the respective entry in the matrix holds the sum of weights for all derivations. This makes it harder for the learning algorithm to infer the weight of each tree separately. By choosing to work with structurally unambiguous grammars, we overcome this difficulty as an entry corresponds to a single derivation tree.

To discuss properties of the Hankel Matrix for an SUPCFG we need the following definitions. Let H be a matrix, t a tree (or row index) c a context (or column index), T a set of trees (or row indices) and C a set of contexts (or column indices). We use $H[t]$ (resp. $H[c]$) for the row (resp. column) of H corresponding to t (resp. c). Similarly we use $H[T]$ and $H[C]$ for the corresponding sets of rows or columns. Finally, we use $H[t][C]$ for the restriction of H to row t and columns $[C]$.

Two vectors, $v_1, v_2 \in \mathbb{R}^n$ are *co-linear* with a scalar $\alpha \in \mathbb{R}$ for some $\alpha \neq 0$ iff $v_1 = \alpha \cdot v_2$. Given a matrix H, and two trees t_1 and t_2, we say that $t_1 \ltimes_H^\alpha t_2$ iff $H[t_1]$ and $H[t_2]$ are co-linear, with scalar $\alpha \neq 0$. That is, $H[t_1] = \alpha \cdot H[t_2]$. Note that if $H[t_1] = H[t_2] = \bar{0}$, then $t_1 \ltimes_H^\alpha t_2$ for every $\alpha > 0$. We say that $t_1 \equiv_H t_2$ if $t_1 \ltimes_H^\alpha t_2$ for some $\alpha \neq 0$. It is not hard to see that $\equiv_H$ is an equivalence relation.

The following proposition states that in the Hankel Matrix of an SUPCFG, the rows of trees that are rooted by the same non-terminal are co-linear.

Proposition 4.6. *Let H be the Hankel Matrix of an SUPCFG. Let t_1, t_2 be derivation trees rooted by the same non-terminal. Assume $\mathbb{P}(t_1), \mathbb{P}(t_2) > 0$. Then $t_1 \ltimes_H^\alpha t_2$ for some $\alpha \neq 0$.*

Proof. Let c be a context. Let $u \diamond v$ be the yield of the context; that is, the letters with which the leaves of the context are tagged, in a left to right order (note that u and v might be ε). We denote by $\mathbb{P}_i(c)$ the probability of deriving the given context c, while setting the context location to be N_i. That is:

$$\mathbb{P}_i(c) = \mathbb{P}(S \xrightarrow[\mathcal{G}]{*} uN_iv)$$

Let $\mathbb{P}(t_1)$ and $\mathbb{P}(t_2)$ be the probabilities for deriving the trees t_1 and t_2 respectively. Since the grammar is structurally unambiguous, we are guaranteed there is a single derivation tree for uwv where w is the yield of t_1 with the structure of $c[\![t_1]\!]$ (and similarly for t_2), thus we obtain

$$\mathbb{P}(c[\![t_1]\!]) = \mathbb{P}_i(c) \cdot \mathbb{P}(t_1)$$
$$\mathbb{P}(c[\![t_2]\!]) = \mathbb{P}_i(c) \cdot \mathbb{P}(t_2)$$

Hence for every context c, assuming that $P_i(c) \neq 0$, we have

$$\frac{\mathbb{P}(c[\![t_1]\!])}{\mathbb{P}(c[\![t_2]\!])} = \frac{\mathbb{P}(t_1)}{\mathbb{P}(t_2)}$$

For a context c for which $\mathbb{P}_i(c) = 0$ we obtain that $\mathbb{P}(c[\![t_1]\!]) = \mathbb{P}(c[\![t_2]\!]) = 0$. Therefore, for every context

$$\mathbb{P}(c[\![t_1]\!]) = \frac{\mathbb{P}(t_1)}{\mathbb{P}(t_2)} \mathbb{P}(c[\![t_2]\!])$$

Thus $H[t_1] = \alpha \cdot H[t_2]$ for $\alpha = \frac{\mathbb{P}(t_1)}{\mathbb{P}(t_2)}$. Hence $H[t_1]$ and $H[t_2]$ are co-linear, and $t_1 \ltimes_H^{\alpha} t_2$. ☐

We can thus conclude that the number of equivalence classes of $\equiv_H$ for an SUPCFG is finite and bounded by the number of non-terminals plus one (for the zero vector).

Corrolary 4.1. *The skeletal tree-set for an SUPCFG has a finite number of equivalence classes under* $\equiv_H$.

Proof. Since the PCFG is structurally unambiguous, it follows that for every skeletal tree s there is a single tagged parse tree t such that $\mathsf{S}(t) = s$. Hence, for every s there is a single possible tagging, and a single possible non-terminal in the root. By Proposition 4.6 every pair of trees s_1, s_2 which are tagged by the same non-terminal, and in which $\mathbb{P}(s_1), \mathbb{P}(s_2) > 0$ are in the same equivalence class under $\equiv_H$. There is another equivalence class for all the trees $t \in \mathit{Trees}$ for which $\mathbb{P}(t) = 0$. Since there is a finite number of non-terminals, there is a finite number of equivalence classes under $\equiv_H$. ☐

Next we would like to reveal the restrictions that can be imposed on a PMTA that corresponds to an SUPCFG. We term a PMTA *co-linear*, and denote it CMTA, if in every column of every transition matrix μ_σ there is at most one entry which is non-negative. That is, in every transition matrix μ_σ of a CMTA all weights are either zero or positive, and there is at most one non-zero entry in each column.

Proposition 4.7. *A CMTA can represent an SUPCFG.*

To prove Proposition 4.7 we first show how to convert a PWCFG into a PMTA. Then we claim, that in case the PWCFG is structurally unambiguous the resulting PMTA is a CMTA.

Converting a PWCFG into a PMTA. Let $\langle \mathcal{G}, \theta \rangle$ be a PWCFG where $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S \rangle$. Suppose w.l.o.g that $\mathcal{V} = \{N_0, N_1, ..., N_{|\mathcal{V}|-1}\}$, $\Sigma = \{\sigma_0, \sigma_1, ..., \sigma_{|\Sigma|-1}\}$ and that $S = N_0$. Let $n = |\mathcal{V}| + |\Sigma|$. We define a function $\iota : \mathcal{V} \cup \Sigma \to \mathbb{N}_{\leq n}$ in the following manner:

$$\iota(x) = \begin{cases} j & x = N_j \in \mathcal{V} \\ |\mathcal{V}| + j & x = \sigma_j \in \Sigma \end{cases}$$

Note that since $\mathcal{V} \cap \Sigma = \emptyset$, ι is well defined. It is also easy to observe that ι is a bijection, so $\iota^{-1} : \mathbb{N}_{\leq n} \to \mathcal{V} \cup \Sigma$ is also a function.
We define a PMTA $\mathcal{A}_{\mathcal{G}}$ in the following manner:

$$\mathcal{A}_{\mathcal{G}} = (\Sigma, \mathbb{R}_+, n, \mu, \lambda)$$

where $\lambda \in \mathbb{R}_*^n$ is defined as follows

$$\lambda = (1, 0, ..., 0)$$

and for each $\sigma \in \Sigma$ we define

$$\mu_\sigma[i] = \begin{cases} 1 & i = \iota(\sigma) \\ 0 & \text{otherwise} \end{cases}$$

For $i \in \{1, \ldots, |\mathcal{V}|\}$ and $(i_1, i_2, \ldots, i_j) \in \{1, 2 \ldots, n\}^{|j|}$, we define $R^{-1}(i, i_1, i_2, \ldots, i_j)$ to be the production rule

$$\iota^{-1}(i) \longrightarrow \iota^{-1}(i_1)\ \iota^{-1}(i_2)\ \cdots\ \iota^{-1}(i_j)$$

We define $\mu_?$ in the following way:

$$\mu_?{}^{i}{}_{i_1,\dots,i_j} = \begin{cases} \theta(R^{-1}(i, i_1, i_2, \dots, i_j)) & 1 \le i \le |\mathcal{V}| \\ 0 & \text{otherwise} \end{cases}$$

We claim that the weights computed by the constructed PMTA agree with the weights computed by the given grammar.

Proposition 4.8. *For each skeletal tree $t \in \mathsf{S}(\mathsf{T}(\mathcal{G}))$ we have that $\mathcal{W}_{\mathcal{G}}(t) = \mathcal{A}_{\mathcal{G}}(t)$.*

Proof. The proof is reminiscent of the proof in the other direction, namely that of Proposition 3.1. We first prove by induction that for each $t \in \mathsf{S}(\mathsf{T}(\mathcal{G})))$ the vector $\mu(t) = v = (v[1], v[2], ..., v[n])$ calculated by $\mathcal{A}_{\mathcal{G}}$ maintains that for each $i \le |\mathcal{V}|$, $v[i] = \mathcal{W}_{N_i}(t)$; and for $i > |\mathcal{V}|$ we have that $v[i] = 1$ iff $t = \iota^{-1}(i)$ and $v[i] = 0$ otherwise.

The proof is by induction on the height of t. For the base case $h = 1$, thus t is a leaf, therefore $t = \sigma \in \Sigma$. By definition $\mu_\sigma[i] = 1$ if $i = \iota(\sigma)$ and 0 otherwise. Hence $v[\iota(\sigma)] = 1$, and for every $i \neq \iota(\sigma)$ $v[i] = 0$. Since the root of the tree is in Σ, the root of the tree can't be a non-terminal, so $\mathcal{W}_{N_i}(t) = 0$ for every i. Thus, the claim holds.

For the induction step, $h > 1$, thus $t = (?(t_1, t_2, ..., t_k))$ for some skeletal trees $t_1, t_2, ..., t_k$ of depth at most h. Let $\mu(t) = v = (v[1], v[2], ..., v[n])$ be the vector calculated by $\mathcal{A}$ for t. By our definition of $\mu_?$, for every $i > |\mathcal{V}|$ $\mu_?{}^{i}{}_{i_1,\dots,i_j} = 0$ for all values of $i_1, i_2, ..., i_j$. So for every $i > |\mathcal{V}|$ we have that $v[i] = 0$ as required, since $t \notin \Sigma$. Now for $i \le |\mathcal{V}|$, by definition of a multi-linear map we have that:

$$v[i] = \sum_{(i_1,i_2,\dots,i_j)\in[|\mathcal{V}|]^j} \mu_?{}^{i}{}_{i_1,\dots,i_j}\, v_1[j_1] \cdot \dots \cdot v_j[i_j]$$

Since $i \le |\mathcal{V}|$, by our definition we have that:

$$\mu_?{}^{i}{}_{i_1,\dots,i_j} = \theta(\iota^{-1}(i) \longrightarrow \iota^{-1}(i_1)\ \iota^{-1}(i_2)\ \cdots\ \iota^{-1}(i_j))$$

For each i_k let $B_k = \iota^{-1}(i_k)$, also since $i \le |\mathcal{V}|$, $\iota^{-1}(i) = N_i$, hence

$$\mu_?{}^{i}{}_{i_1,\dots,i_j} = \theta(N_i \longrightarrow B_1 B_2 ... B_j)$$

For each i, by the induction hypothesis, if t_i is a leaf, $v_i[j_i] = 1$ only for $j_i = \iota(t_i)$, and otherwise $v_i[j_i] = 0$. If t_i is not a leaf, then $v_i[j_i] = 0$ for every $j_i > |\mathcal{V}|$; and for $j_i \le |\mathcal{V}|$, we have that $v_i[j_i] = \mathcal{W}_{N_{j_i}}(t_i)$. Therefore we have:

$$v[i] = \sum_{(i_1,i_2,\dots,i_j)\in[|\mathcal{V}|]^j} \theta(N_i \to B_1 B_2 ... B_j) \cdot \mathcal{W}_{N_{i_1}}(t_1) \cdots \mathcal{W}_{N_{i_j}}(t_j)$$

Thus by Lemma 3.2 we have that $v[i] = \mathcal{W}_{N_i}(t)$ as required.

Finally, since $S = N_0$ and since by our claim, for each $i \le |\mathcal{V}|$, $v_i = v[i] = \mathcal{W}_{N_i}(t)$, we get that $v[1] = \mathcal{W}_S(t)$. Also, since $\lambda = (1, 0, ..., 0)$ we have that $\mathcal{A}_{\mathcal{G}}(t)$ is $v[1]$, which is $\mathcal{W}_S(t)$. Thus, it follows that $\mathcal{W}_{\mathcal{G}}(t) = \mathcal{A}_{\mathcal{G}}(t)$ for every $t \in \mathsf{S}(\mathsf{T}(\mathcal{G}))$. □

To show that the resulting PMTA is a CMTA we need the following lemma, which makes use of the notion of *invertible* grammars [Sak92]. A CFG $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S \rangle$ is said to be *invertible* if and only if $A \to \alpha$ and $B \to \alpha$ in R implies $A = B$.

Lemma 4.2. *A CFG is invertible iff it is structurally unambiguous.*

Proof. Let $\mathcal{G}$ be a SUCFG. We show that $\mathcal{G}$ is invertible. Assume towards contradiction that there are derivations $N_1 \rightarrow \alpha$ and $N_2 \rightarrow \alpha$. Then the tree $?(\alpha)$ is structurally ambiguous since its root can be tagged by both N_1 and N_2.

For the other direction, let $\mathcal{G}$ be an invertible grammar. We show that $\mathcal{G}$ is an SUCFG. Let t be a skeletal tree. We show by induction on the height of t that there is a single tagging for t. For the base case, the height of t is 1. Therefore, t is a leaf; so obviously, it has a single tagging. For the induction step, we assume that the claim holds for all skeletal trees of height at most $h \geq 1$. Let t be a tree of height $h+1$. Then $t = ?(t_1, t_2, ..., t_p)$ for some trees $t_1, t_2, ..., t_p$ of smaller depth. By the induction hypothesis, for each of the trees $t_1, t_2, ..., t_p$ there is a single possible tagging. Hence we have established that all nodes of t, apart from the root, have a single tagging. Let $X_i \in \Sigma \cup N$ be the only possible tagging for the root of t_i. Let $\alpha = X_1 X_2 ... X_p$. Since the grammar is invertible, there is a single non-terminal N s.t. $N \rightarrow \alpha$. Hence, there is a single tagging for the root of t as well. Thus $\mathcal{G}$ is structurally unambiguous. $\square$

We are finally ready to prove Proposition 4.7.

Proof of Prop.4.7. By Proposition 4.8 a WCFG $\langle \mathcal{G}, \theta \rangle$ can be represented by a PMTA $\mathcal{A}_{\mathcal{G}}$, namely they provide the same weight for every skeletal tree. By Lemma 4.2 the fact that $\mathcal{G}$ is unambiguous implies it is invertible. We show that given $\mathcal{G}$ is invertible, the resulting PMTA is actually a CMTA. That is, in every column of the matrices of $\mathcal{A}_{\mathcal{G}}$, there is at most one non-zero coefficient. Let $\alpha \in (\Sigma \cup \mathcal{V})^p$, let $\iota(\alpha)$ be the extension of ι to α, e.g., $\iota(aN_7bb) = \iota(a)\iota(N_7)\iota(b)\iota(b)$. Since $\mathcal{G}$ is invertible, there is a single N_i from which α can be derived, namely for which $\mathcal{W}_{N_i}(t_\alpha^{N_i}) > 0$ where $t_\alpha^{N_i}$ is a tree deriving α with N_i in the root. If $\alpha \in \Sigma$, i.e. it is a leaf, then we have that $\mu_\sigma[j] = 0$ for every $j \neq i$, and $\mu_\sigma[i] > 0$. If $\alpha \notin \Sigma$, then we have that ${\mu_?}^{j}{}_{\iota(\alpha)} = 0$ for every $j \neq i$, and ${\mu_?}^{i}{}_{\iota(\alpha)} > 0$, as required. $\square$

5. The Learning Algorithm

We are now ready to present the learning algorithm. Let $\mathcal{T} : \mathit{Trees}(\Sigma) \rightarrow \mathbb{R}$ be an unknown tree series, and let $H_{\mathcal{T}}$ be its Hankel Matrix. The learning algorithm *LearnCMTA* (or $\mathbf{C}^*$, for short), provided in Alg. 1, maintains a data structure called an *observation table*. An observation table for $\mathcal{T}$ is a quadruple (T, C, H, B). Where $T \subseteq \mathit{Trees}(\Sigma)$ is a set of row titles, $C \subseteq \mathit{Trees}_{\diamond(\Sigma)}$ is a set of column titles, $H : T \times C \rightarrow \mathbb{R}$ is a sub-matrix of $H_{\mathcal{T}}$, and $B \subset T$, the so called *basis*, is a set of row titles corresponding to rows of H that are co-linearly independent.

The algorithm starts with an almost empty observation table, where $T = \emptyset$, $C = \diamond$, $B = \emptyset$ and uses procedure *Complete*(T, C, H, B, Σ_0) to add the nullary symbols of the alphabet to the row titles, uses SMQ queries to fill in the table until certain criteria hold on the observation, namely it is *closed* and *consistent*, as defined in the sequel. Once the table is closed and consistent, it is possible to extract from it a CMTA $\mathcal{A}$ (as we shortly explain). The algorithm then issues the query SEQ$(\mathcal{A})$. If the result is "yes" the algorithm returns $\mathcal{A}$ which was determined to be structurally equivalent to the unknown series. Otherwise, the algorithm gets in return a counterexample $(s, \mathcal{T}(s))$, a structured string in the symmetric difference of $\mathcal{A}$ and $\mathcal{T}$, and its value. It then uses *Complete* to add all prefixes of t to T and uses SMQs to fill in the entries of the table until the table is once again closed and consistent.

Given a set of trees T we use $\Sigma(T)$ for the set of trees $\{\sigma(t_1, \ldots, t_k) \mid \exists \Sigma_k \in \Sigma,\ \sigma \in \Sigma_k,\ t_i \in T,\ \forall 1 \leq i \leq k\}$. The procedure *Close*$(T, C, H, B)$, given in Alg. 2, checks if $H[t][C]$ is

Algorithm 1 *LearnCMTA*(T, C, H, B).

```
1 Initialize $B \leftarrow \emptyset,\ T \leftarrow \emptyset,\ C \leftarrow \{\diamond\}$
2 Complete$(T, C, H, B, \Sigma_0)$
3 while true do
4     $\mathcal{A} \leftarrow$ ExtractCMTA$(T, C, H, B)$
5     $t \leftarrow \textsc{seq}(\mathcal{A})$
6     if $t$ is null then
7         return $\mathcal{A}$
8     Complete$(T, C, H, B, Pref(t))$
```

Algorithm 2 *Close*(T, C, H, B).

```
1 while $\exists t \in \Sigma(T)$ s.t. $H[t]$ is co-linearly independent
      from $T$ do
2     $B \leftarrow B \cup \{t\}$
3     $T \leftarrow T \cup \{t\}$
```

Algorithm 3 *Consistent*(T, C, H, B)

```
1 for $t \in T$ s.t. $H[t] = \overline{0}$ do
2     for $c \in \Sigma(T, \diamond), c' \in C$ do
3         if $H[c'[\![c[t]]\!]] \neq \overline{0}$ then
4             $C \leftarrow C \cup \{c[\![c']\!]\}$
5 for $t_1, t_2 \in T$ s.t. $t_1 \ltimes_H^{\alpha} t_2$ do
6     for $c \in \Sigma(T, \diamond), c' \in C$ do
7         if $H[c'][c[\![t_1]\!]] \neq \alpha H[c'][c[\![t_2]\!]]$ then
8             $C \leftarrow C \cup \{c[\![c']\!]\}$
```

Algorithm 4 *Complete*(T, C, H, B, S).

```
1 $T \leftarrow T \cup S$
2 while $(T, C, H, B)$ is not closed or not consistent do
3     Close$(T, C, H, B)$
4     Consistent$(T, C, H, B)$
```

co-linearly independent from T for some tree $t \in \Sigma(T)$. If so it adds t to both T and B and loops back until no such trees are found, in which case the table is termed *closed*.

We use $\Sigma(T, t)$ for the set of trees in $\Sigma(T)$ satisfying that one of the children is the tree t. We use $\Sigma(T, \diamond)$ for the set of contexts all of whose children (but the context) are in T. An observation table (T, C, H, B) is said to be *zero-consistent* if for every tree $t \in T$ for which $H[t] = \overline{0}$ it holds that $H[c[\![t']\!]] = \overline{0}$ for every $t' \in \Sigma(T, t)$ and $c \in C$ (where $\overline{0}$ is a vector of all zeros). It is said to be *co-linear consistent* if for every $t_1, t_2 \in T$ such that $t_1 \ltimes_H^{\alpha} t_2$ and every context $c \in \Sigma(T, \diamond)$ we have that $c[\![t_1]\!] \ltimes_H^{\alpha} c[\![t_2]\!]$. The procedure *Consistent*, given in Alg. 3, looks for trees which violate the zero-consistency or co-linear consistency requirement, and for every violation, the respective context is added to the set of columns C.

The procedure *Complete*(T, C, H, B, S), given in Alg. 4, first adds the trees in S to T, then runs procedures *Close* and *Consistent* iteratively until the table is both closed and consistent. When the table is closed and consistent the algorithm extracts from it a CMTA as detailed in Alg. 5.

The procedure *ExtractCMTA*, given in Alg. 5, sets the output vector λ of the CMTA by setting its j-th coordinate to $H(b_j, \diamond)$ (lines 17-18). For each letter $\sigma \in \Sigma_k$ it builds the

Algorithm 5 Extract CMTA

```
d ← |B|
for 0 ≤ k ≤ p do
    for σ ∈ Σ_k do
        for (i_1, i_2, ..., i_k) ∈ {1, 2, ..., d}^k do
            Let t = σ(b_{i_1}, b_{i_2}, ..., b_{i_k})
            if H[t] = 0 then
                for 1 ≤ j ≤ d do
                    σ^j_{i_1,i_2,...,i_k} ← 0
            else
                Let b_i ∈ B, α ∈ ℝ be s.t. t ⋉^α_H b_i
                for 1 ≤ j ≤ d do
                    if j = i then
                        σ^j_{i_1,i_2,...,i_k} ← α
                    else
                        σ^j_{i_1,i_2,...,i_k} ← 0
        Let μ_σ be the d × d^k matrix obtained from the
            respective coefficients.
for 1 ≤ j ≤ d do
    λ[j] ← H(b_j, ⋄)
return A = (Σ, ℝ, d, μ, λ)
```

$d \times d^k$ matrix μ_σ (where d is the size of the basis B) by setting its coefficients as follows (lines 4-15). For each possible assignments of trees from the basis as children of σ, namely for each $(i_1, i_2, \ldots, i_k) \in \{1, 2, \ldots, d\}$ it inspects the row $H(t)$ for $t = \sigma(b_{i_1}, b_{i_2}, \ldots, b_{i_k})$. If this row is zero then the column of μ_σ corresponding to $(i_1, i_2, \ldots, i_k)$ is set to zero. Otherwise, since the basis consists of rows that are co-linearly independent, there exists a single row in the basis, b_i, such that the row $H(t)$ for $t = \sigma(b_{i_1}, b_{i_2}, \ldots, b_{i_k})$ is co-linear to b_i. Let $\alpha \in \mathbb{R}_+$ be such that $t \ltimes^{\alpha}_{H} b_i$. Then the entry $\mu^i_{i_1,i_2,\ldots,i_k}$ is set to α and the entries $\mu^j_{i_1,i_2,\ldots,i_k}$ for $j \neq i$ are set to zero.

Overall we can show that the algorithm *LearnCMTA* always terminates, returning a correct CMTA whose dimension is minimal, namely it equals the rank n of the Hankel matrix for the target language. It does so while asking at most n equivalence queries, and the number of membership queries is polynomial in n, and in the size of the largest counterexample m, but of course exponential in p, the highest rank of a symbol in Σ. Hence for a grammar in Chomsky Normal Form, where $p = 2$, it is polynomial in all parameters.

Theorem 5.1. *Let n be the rank of the target language, let m be the size of the largest counterexample given by the teacher, and let p be the highest rank of a symbol in Σ. Then the algorithm makes at most $n \cdot (n + m \cdot n + |\Sigma| \cdot (n + m \cdot n)^p)$ SMQs and at most n SEQs.*

We first give a running example in §5.1, and then prove this theorem in §5.2.

5.1. **Running Example.**

We will now demonstrate a running example of the learning algorithm. For the unknown target consider the series which gives probability $(\frac{1}{2})^n$ to strings of the form $a^n b^n$ for $n \geq 1$ and probability zero to all other strings. This series can be generated by the following SUPCFG $\mathcal{G} = \langle \mathcal{V}, \{a, b\}, R, S \rangle$ with $\mathcal{V} = \{S, S_2\}$, and the following derivation rules:

$$\begin{array}{lcl} S & \longrightarrow & aS_2\ [\tfrac{1}{2}] \ \mid\ ab\ [\tfrac{1}{2}] \\ S_2 & \longrightarrow & Sb\ [1] \end{array}$$

(a)

	$\diamond$
a	0
b	0
$?(a,a)$	0
$?(a,b)$	0.5
$?(b,a)$	0
$?(b,b)$	0

(b)

		$\diamond$	$?(a,\diamond)$
t_1	a	0	0
t_2	b	0	0.5
t_3	$?(a,b)$	0.5	0
	$?(?(a,b),b)$	0	0.25

(c)

		$\diamond$	$?(a,\diamond)$	$?(\diamond,b)$
t_1	a	0	0	0.5
t_2	b	0	0.5	0
t_3	$?(a,b)$	0.5	0	0
$t_4 \notin B$	$?(a,a)$	0	0	0
	$?(?(a,b),b)$	0	0.25	0

Table 1. Observation tables (a) (b) and (c)

The algorithm initializes $T = \{a, b\}$ and $C = \{\diamond\}$, fills in the entries of M using SMQs, first for the rows of T and then for their one letter extensions $\Sigma(T)$ (marked in gray), resulting in the observation table in Tab. 1 (a).

We can see that the table is not closed, since $?(a,b) \in \Sigma(T)$ but is not co-linearly spanned by T, so we add it to T. Also, the table is not consistent, since $a \ltimes_H^1 b$, but $\text{MQ}(\diamond[\![?(a,b)]\!]) \neq \text{MQ}(\diamond[\![?(a,a)]\!])$, so we add $?(a,\diamond)$ to C, and we obtain the observation table in Tab. 1 (b). From now on we omit 0 rows of $\Sigma(T)$ for brevity.

The table is now closed but it is not zero-consistent, since we have $H[a] = \overline{0}$, but there exists a context with children in T, specifically $?(\diamond, b)$, with which when a is extended the result is not zero, namely $H[?(a,b)] \neq 0$. So we add this context and we obtain the observation table in Tab. 1 (c).

Note that t_4 was added to T since it was not spanned by T, but it is not a member of B, since $H[t_4] = 0$. We can extract the following CMTA $\mathcal{A}_1 = (\Sigma, \mathbb{R}, d, \mu, \lambda)$ of dimension $d = 3$ since $|B| = |\{t_1, t_2, t_3\}| = 3$. Let $\mathbb{V} = \mathbb{R}^3$. For the letters $\sigma \in \Sigma_0 = \{a, b\}$ we have that $\mu_\sigma : \mathbb{V}^0 \to \mathbb{V}$, namely μ_a and μ_b are 3×3^0-matrices. Specifically, following Alg. 4 we get that $\mu_a = (1, 0, 0)$, $\mu_b = (0, 1, 0)$ as a is the first element of B and b is the second. For $? \in \Sigma^2$ we have that $\mu_? : \mathbb{V}^2 \to \mathbb{V}$, thus $\mu_?$ is a 3×3^2-matrix. We compute the entries of $\mu_?$ following Alg. 4. For this, we consider all pairs of indices $(j, k) \in \{1, 2, 3\}^2$. For each such entry we look for the row $t_{j,k} = ?(t_j, t_k)$ and search for the base row t_i and the scalar α for which $t_{j,k} \ltimes_H^\alpha t_i$. We get that $t_{1,2} \ltimes_H^1 t_3$, $t_{3,2} \ltimes_H^{0.5} t_2$ and for all other j, k we get $t_{j,k} \ltimes_H^1 t_4$, so we set $c_{j,k}^i$ to be 0 for every i. Thus, we obtain the following matrix for $\mu_?$

$$\eta_? = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The vector λ is also computed via Alg. 4, and we get $\lambda = (0, 0, 0.5)$.

The algorithm now asks an equivalence query and receives the tree p given in Fig. 8 (i) as a counterexample:

Indeed, while $\text{SMQ}(p) = 0$ we have that $\mathcal{A}(p) = 0.125$. To see why $\mathcal{A}(p) = 0.125$, let's look at the values $\mu(t)$ for every sub-tree t of p. For the leaves, we have $\mu(a) = (1, 0, 0)$ and $\mu(b) = (0, 1, 0)$.

Now, to calculate $\mu(?(a,b))$, we need to calculate $\mu_?(\mu(a), \mu(b))$. To do that, we first compose them as explained in the *multilinear functions* paragraph of Sec. 2.2, see also Fig. 2. The vector $P_{\mu(a),\mu(b)}$ is $(0, 1, 0, 0, 0, 0, 0, 0, 0)$. When multiplying this vector by the matrix $\eta_?$ we obtain $(0, 0, 1)$. So $\mu(?(a,b)) = (0, 0, 1)$. Similarly, to obtain $\mu(?(?(a,b),a))$

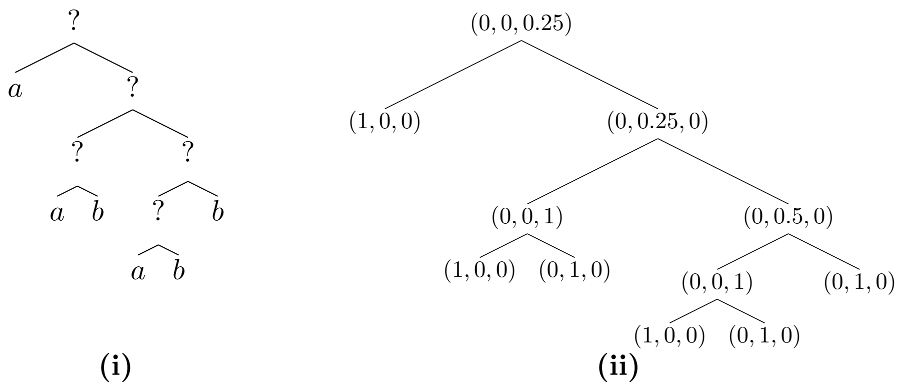


Figure 8. (i) The tree p and (ii) the tree p annotated with the value $\mu(t)$ for its of sub-trees t.

		$\diamond$	$?(a,\diamond)$	$?(\diamond,b)$
t_1	a	0	0	$\frac{1}{2}$
t_2	b	0	$\frac{1}{2}$	0
t_3	$?(a,b)$	$\frac{1}{2}$	0	0
t_4	$?(a,a)$	0	0	0
t_5	$?(?(a,a),a)$	0	0	0
t_6	$?(?(a,b),b)$	0	$\frac{1}{4}$	0
t_7	$?(?(a,b),?(?(a,b),b))$	0	0	0
t_8	$?(a,?(?(a,b),?(?(a,b),b)))$	0	0	0

(d)

		$\diamond$	$?(a,\diamond)$	$?(\diamond,b)$	$?(a,?(?(a,b),\diamond))$
t_1	a	0	0	$\frac{1}{2}$	0
t_2	b	0	$\frac{1}{2}$	0	$\frac{1}{4}$
t_3	$?(a,b)$	$\frac{1}{2}$	0	0	0
t_4	$?(a,a)$	0	0	0	0
	$?(?(a,a),a)$	0	0	0	0
t_5	$?(?(a,b),b)$	0	$\frac{1}{4}$	0	0
	$?(?(a,b),?(?(a,b),b))$	0	0	0	0
	$?(a,?(?(a,b),?(?(a,b),b)))$	0	0	0	0
t_6	$?(a,?(?(a,b),b))$	$\frac{1}{4}$	0	0	0

(e)

Table 2. Observation tables (d) and (e)

we first compose the value $(0,0,1)$ for $?(a,b)$ with the value $(0,1,0)$ for a and obtain $P_{\mu?(a,b),\mu(a)} = (0,0,0,0,0,0,0,1,0)$. Then we multiply $\eta_?$ by $P_{\mu?(a,b),\mu(a)}$ and obtain $(0,0.5,0)$. In other words,

$$\mu(?(?(a,b),a)) = \mu_?\left(\begin{bmatrix}0\\0\\1\end{bmatrix},\begin{bmatrix}0\\1\\0\end{bmatrix}\right) = \begin{bmatrix}0\\0.5\\0\end{bmatrix}.$$

The tree in Fig. 8 (ii) depicts the entire calculation by marking the values obtained for each sub-tree. We can see that $\mu(p) = (0,0,0.25)$, thus we get that $\mathcal{A} = \mu(p)\cdot\lambda = 0.125$.

We add all prefixes of this counterexample to T and we obtain the observation table in Tab. 2 (d). This table is not consistent since while $t_6 \ltimes_H^{0.5} t_2$ this co-linearity is not preserved when extended with $t_3 = ?(a,b)$ to the left, as evident from the context $?(a,\diamond)$. We thus add the context $?(a,\diamond)[\![?(?(a,b),\diamond)]\!] = ?(a,?(?(a,b),\diamond))$ to obtain the final observation table given in in Tab. 2 (e).

The table is now closed and consistent, and we extract the following CMTA from it: $\mathcal{A}_3 = (\Sigma,\mathbb{R},4,\mu,\lambda)$ with $\mu_a = (1,0,0,0)$, $\mu_b = (0,1,0,0)$. Now $\mu_?$ is a 4×4^2 matrix. Its non-zero entries are $c^3_{1,2} = 1$, $c^4_{3,2} = 1$ and $c^3_{1,4} = \frac{1}{2}$ since $t_{1,2} \ltimes^1_H t_3$, $t_{3,2} \ltimes^1_H t_5$, $t_{1,5} \ltimes^1_H t_6 \ltimes^{0.5}_H t_3$. And for every other combination of unit-basis vectors we have $t_{i,j} \ltimes^1_H t_4$. The final output vector is $\lambda = (0, 0, 0.5, 0)$.

The equivalence query on this CMTA returns true, hence the algorithm now terminates, and we can convert this CMTA into a WCFG. Applying the transformation provided in Fig. 4 we obtain the following WCFG:

$$\begin{aligned}
S &\longrightarrow N_3 \ [0.5] \\
N_1 &\longrightarrow a \ [1.0] \\
N_2 &\longrightarrow b \ [1.0] \\
N_3 &\longrightarrow N_1 N_2 \ [1.0] \ \mid \ N_1 N_4 \ [0.5] \\
N_4 &\longrightarrow N_3 N_2 \ [1.0]
\end{aligned}$$

Now, following [AMP99, SJ07] we can calculate the partition functions for each non-terminal. Let f_N be the sum of the weights of all trees whose root is N, we obtain:

$$f_S = 1, \quad f_{N_1} = 1, \quad f_{N_2} = 1, \quad f_{N_3} = 2, \quad f_{N_4} = 2.$$

Hence we obtain the PCFG

$$\begin{aligned}
S &\longrightarrow N_3 \ [1.0] \\
N_1 &\longrightarrow a \ [1.0] \\
N_2 &\longrightarrow b \ [1.0] \\
N_3 &\longrightarrow N_1 N_2 \ [0.5] \mid N_1 N_4 \ [0.5] \\
N_4 &\longrightarrow N_3 N_2 \ [1.0]
\end{aligned}$$

which is a correct grammar for the unknown probabilistic series.

The careful reader might notice that the produced grammar is not the most succinct one for this language. Indeed, the conversation of a CMTA to a PCFG could be optimized; currently it adds additional $O(\Sigma)$ non-terminals and derivation rules (for every terminal a a non-terminal N_a and a respective production rule $N_a \longrightarrow a$ are introduced). Algorithm 1 itself does return the minimal CMTA with respect to the given skeletal trees.

5.2. **Correctness Proof.**

To prove the main theorem we require a series of lemmas, which we state and prove here. We start with some additional notations. Let v be a row vector in a given matrix. Let C be a set of columns. We denote by $v[C]$ the restriction of v to the columns of C. For a set of row-vectors V in the given matrix, we denote by $V[C]$ the restriction of all vectors in V to the columns of C.

Lemma 5.2. *Let B be a set of row vectors in a matrix H, and let C be a set of columns. If a row $v[C]$ is co-linearly independent from $B[C]$ then v is co-linearly independent from B.*

Proof. Assume towards contradiction that there is a vector $b \in B$ and a scalar $\alpha \in \mathbb{R}$ s.t. $v = \alpha b$. Then for every column c we have $v[c] = \alpha b[c]$. In particular that holds for every $c \in C$. Thus, $v[C] = \alpha b[C]$ and so $v[C]$ is not co-linearly independent from $B[C]$, contradicting our assumption. □

Lemma 5.3. *Let $\mathcal{A} = (\Sigma, \mathbb{R}, d, \mu, \lambda)$ be a CMTA. Let t_1, t_2 s.t. $\mu(t_1) = \alpha \cdot \mu(t_2)$. Then for every context c*

$$\mu(c[\![t_1]\!]) = \alpha \cdot \mu(c[\![t_2]\!])$$

Proof. The proof is by induction on the depth of $\diamond$ in c. For the base case, the depth of $\diamond$ in c is 1. Hence, $c = \diamond$ and indeed we have

$$\mu(c[\![t_1]\!]) = \mu(t_1) = \alpha \cdot \mu(t_2) = \alpha \cdot \mu(c[\![t_2]\!])$$

as required.

For the induction step, assume the claim holds for all contexts where $\diamond$ is in depth at most h. Let c be a context s.t. $\diamond$ is in depth $h+1$. Hence, there exists contexts c_1 and c_2 s.t. $c = c_1[\![c_2]\!]$ where $c_2 = \sigma(s_1, s_2, ..., s_{i-1}, \diamond, s_{i+1}, ..., s_p)$ for some s_i's and the depth of $\diamond$ in c_1 is h. Let $t'_1 = c_2[\![t_1]\!]$ and let $t'_2 = c_2[\![t_2]\!]$. We have

$$\begin{aligned}\mu(t'_1) &= \mu(c_2[\![t_1]\!]) \\ &= \mu_\sigma(\mu(s_1), \mu(s_2), ..., \mu(s_{i-1}), \mu(t_1), \mu(s_{i+1}), ..., \mu(s_p)) \\ &= \mu_\sigma(\mu(s_1), \mu(s_2), ..., \mu(s_{i-1}), \alpha \cdot \mu(t_2), \mu(s_{i+1}), ..., \mu(s_p))\end{aligned}$$

Similarly for t_2 we obtain

$$\mu(t'_2) = \mu_\sigma(\mu(s_1), \mu(s_2), ..., \mu(s_{i-1}), \mu(t_2), \mu(s_{i+1}), ..., \mu(s_p)).$$

By properties of multi-linear functions we obtain:

$$\begin{aligned}\mu(\sigma(s_1, s_2, ..., s_{i-1}, t_1, s_{i+1}, ..., s_p)) &= \\ \alpha \cdot \mu(\sigma(s_1, s_2, ..., s_{i-1}, t_2, s_{i+1}, ..., s_p))\end{aligned}$$

Thus, $\mu(t'_1) = \alpha \cdot \mu(t'_2)$, and by the induction hypothesis on c_1 we have:

$$\mu(c_1[\![t'_1]\!]) = \alpha \cdot \mu(c_1[\![t'_2]\!])$$

Hence

$$\mu(c[\![t_1]\!]) = \mu(c_1[\![t'_1]\!]) = \alpha \cdot \mu(c_1[\![t'_2]\!]) = \alpha \cdot \mu(c[\![t_2]\!])$$

as required. □

Recall that a subset B of T is called a *basis* if for every $t \in T$, if $H[t] \neq 0$ then there is a unique $b \in B$ such that $t \ltimes^{\alpha}_{H} b$. Let (T, C, H, B) be an observation table. Then $B = \{b_1, b_2 \ldots, b_d\}$ is a basis for T, and if b_i is the unique element of B for which $t \ltimes^{\alpha}_{H} b_i$ for some α, we say that $[t] = b_i$, $\alpha[t] = \alpha$, and $\iota[t] = i$. The following lemma states that the value assigned to a tree $?(t_1, t_2, ..., t_p)$ all of whose children are in T, can be computed by multiplying the respective coefficients $\alpha[t_i]$ witnessing the co-linearity of t_i to its respective base vector $[t_i]$.

Lemma 5.4. *Let (T, C, H, B) be a closed consistent observation table. Let $t_1, t_2, ..., t_p \in T$, and let $t = ?(t_1, t_2, ..., t_p)$. Then*

$$H[?(t_1, t_2, ..., t_p)] = \prod_{i=1}^{p} \alpha[t_i] \cdot H[?([t_1], [t_2], ..., [t_p])]$$

Proof. Let k be the number of elements in $t_1, t_2, ..., t_p$ such that $t_i \neq [t_i]$. We proceed by induction on k. For the base case, we have $k = 0$, so for every t_i we have $t_i = [t_i]$ and $\alpha[t_i] = 1$. Hence, obviously we have

$$H[?(t_1, t_2, ..., t_p)] = \prod_{i=1}^{p} \alpha[t_i] \cdot H[?([t_1], [t_2], ..., [t_p])]$$

Assume now the claim holds for some $k \geq 0$. Since $k + 1 > 0$ there is at least one i such that $t_i \neq [t_i]$. Let $t' = ?(t_1, t_2, ..., t_{i-1}, [t_i], t_{i+1}, ..., t_p)$. Since the table is consistent, we have that $H[t] = \alpha[t_i] \cdot H[t']$. Now, t' has k children such that $t_i \neq [t_i]$, so from the induction hypothesis we have

$$H[t'] = \prod_{\substack{j=1 \\ j \neq i}}^{p} \alpha[t_j] \cdot H[?([t_1],[t_2],...,[t_p])$$

Hence we have

$$H[t] = \alpha[t_i] \cdot H[t'] = \prod_{j=1}^{p} \alpha[t_j] \cdot H[?([t_1],[t_2],...,[t_p])$$

as required. □

The following lemma states that if $t = ?(t_1, t_2, ..., t_p)$ is co-linear to $s = ?(s_1, s_2, ..., s_p)$ and t_i is co-linear to s_i, for every $1 \leq i \leq p$ and $H[t] \neq 0$ then the ratio between the tree coefficient and the product of its children coefficients is the same.

Lemma 5.5. *Let $t = ?(t_1, t_2, ..., t_p)$ and $s = ?(s_1, s_2, ..., s_p)$ s.t. $t_i \equiv_H s_i$ for $1 \leq i \leq p$. Then*

$$\frac{\alpha[t]}{\prod_{i=1}^{p} \alpha[t_i]} = \frac{\alpha[s]}{\prod_{i=1}^{p} \alpha[s_i]}$$

Proof. Let $t' = ?([t_1],[t_2]...,[t_p])$. Note that we also have $t' = ?([s_1],[s_2],...,[s_p])$. Then from Lemma 5.4 we have that $H[t] = \prod_{i=1}^{p} \alpha[t_i] \cdot H[t']$. Similarly we have that $H[s] = \prod_{i=1}^{p} \alpha[s_i] \cdot H[t']$. It follows from Lemma 5.4 that $t \equiv_H s$, since for each i $t_i \equiv_H s_i$. Let $b = [t] = [s]$. We have $H[t] = \alpha[t] \cdot H[b]$, and $H[s] = \alpha[s] \cdot H[b]$. Thus we have

$$\alpha[t] \cdot H[b] = \prod_{i=1}^{p} \alpha[t_i] \cdot H[t']$$

and

$$\alpha[s] \cdot H[b] = \prod_{i=1}^{p} \alpha[s_i] \cdot H[t']$$

Hence we have

$$\frac{\alpha[t]}{\prod_{i=1}^{p} \alpha[t_i]} \cdot H[b] = H[t'] = \frac{\alpha[s]}{\prod_{i=1}^{p} \alpha[s_i]} \cdot H[b]$$

Since $H[t] \neq 0$, and $t \equiv_H b \equiv_H t'$ we obtain that $H[b] \neq 0$ and $H[t'] \neq 0$. Therefore

$$\frac{\alpha[t]}{\prod_{i=1}^{p} \alpha[t_i]} = \frac{\alpha[s]}{\prod_{i=1}^{p} \alpha[s_i]}$$

□

The next lemma relates the value $\mu(t)$ to t's coefficeint, $\alpha[t]$, and the vector for respective row in the basis, $\iota[t]$.

Lemma 5.6. *Let $t \in T$. If $H[t] \neq 0$ then $\mu(t) = \alpha[t] \cdot [t]$. If $H[t] = 0$ then $\mu(t) = 0$.*

Proof. The proof is by induction on the height of t. For the base case, $t = \sigma$ is a leaf, for some $\sigma \in \Sigma$. If $H[t] \neq 0$, by Alg. 5, we set $\sigma^{\iota[t]}$ to be $\alpha[t]$, and for every $j \neq \iota[t]$ we set σ^j to be 0, so $\mu(t) = \mu_\sigma = \alpha[t] \cdot [t]$ as required. Otherwise, if $H[t] = 0$ then we set σ^i to be 0 for every i, so $\mu(t) = 0$ as required.

For the induction step, t is not a leaf. Then $t = ?(t_1, t_2, ..., t_p)$. If $H[t] \neq 0$, then since H is zero-consistent, we have for every $1 \leq i \leq p$ that $H[t_i] \neq 0$. So for every $1 \leq j \leq p$ by induction hypothesis we have $\mu(t_j) = \alpha[t_j] \cdot [t_j]$. Hence

$$\begin{aligned} \mu(t) = & \ \mu_?(\mu(t_1), \ \ldots \ , \mu(t_p)) = \\ = & \ \mu_?(\alpha[t_1] \cdot [t_1], \ \ldots \ , \alpha[t_p] \cdot [t_p]) \end{aligned}$$

Therefore we have

$$\mu(t)[j] = \sum_{j_1,\ldots,j_p \in [n]^p} \sigma^j_{j_1,\ldots,j_p} \cdot \alpha[t_1][t_1][j_1] \cdots \alpha[t_p][t_p][j_p]$$

Note that for every $j_1, j_2, ..., j_p \neq \iota[t_1], \iota[t_2], ..., \iota[t_p]$ we have $\alpha[t_1][t_1][j_1] \ \cdots \ \alpha[t_p][t_p][j_p] = 0$, thus

$$\begin{aligned} \mu(t)[j] &= \sigma^j_{\iota[t_1],\ldots,\iota[t_p]} \cdot \ \alpha[t_1][t_1][\iota[t_1]] \ \cdots \ \alpha[t_p][t_p][\iota[t_p]] \\ &= \sigma^j_{\iota[t_1],\ldots,\iota[t_p]} \cdot \alpha[t_1] \cdot \alpha[t_2] \ \cdots \ \alpha[t_p] \end{aligned}$$

From Alg. 5, and Lemma 5.5 it follows that

$$\sigma^j_{\iota[t_1],\iota[t_2],\ldots,\iota[t_p]} = \begin{cases} 0 & \text{if } j \neq \iota[t] \\ \frac{\alpha[t]}{\prod_{j=1}^p B\alpha[t_p]} & \text{if } j = \iota[t] \end{cases}$$

Hence we obtain

$$\begin{aligned} \mu(t)[\iota[t]] &= \sigma^j_{\iota[t_1],\iota[t_2],\ldots,\iota[t_p]} \cdot \alpha[t_1] \cdot \alpha[t_2] \cdots \alpha[t_p] \\ &= \frac{\alpha[t]}{\prod_{j=1}^p \alpha[t_p]} \cdot \prod_{j=1}^p \alpha[t_p] = \alpha[t] \end{aligned}$$

Thus $\mu(t) = \alpha[t] \cdot [t]$ as required.

If $H[t] = 0$ then $\sigma^i_{\iota[t_1],\iota[t_2],\ldots,\iota[t_p]} = 0$ for every i, and we obtain $\mu(t) = 0$ as required. □

Next we show that rows in the basis get a standard basis vector.

Lemma 5.7. *For every $b_i \in B$, $\mu(b_i) = e_i$ where e_i is the i'th standard basis vector.*

Proof. By induction on the height of b_i. For the base case, b_i is a leaf, so $b = \sigma$ for $\sigma \in \Sigma_0$. By Alg. 5 we set σ_i to be 1 and σ_j to be 0 for every $j \neq i$, so $\mu(b_i) = e_i$.

For the induction step, b_i is not a leaf. Note that by definition of the method *Close* (Alg. 2), all the children of b_i are in B. So $b_i = \sigma(b_{i_1}, b_{i_2}, ..., b_{i_p})$ for some base rows b_{i_j}'s. Let's calculate $\mu(b_i)[j]$

$$\mu(b_i)[j] = \sum_{j_1,j_2,\ldots,j_p \in [n]^p} \sigma^j_{j_1,j_2,\ldots,j_p} \cdot \mu(b_{i_1})[j_1] \cdot \ldots \cdot \mu(b_{i_p})[j_p]$$

By the induction hypothesis, for every $1 \leq j \leq p$ we have that $\mu(b_{i_j})[i_j] = 1$, and $\mu(b_{i_j})[k] = 0$ for $k \neq i_j$. So for every vector $j_1, j_2, ..., j_p \neq i_1, i_2, ..., i_p$ we obtain

$$\mu(b_{i_1})[j_1] \cdot \ldots \cdot \mu(b_{i_p})[j_p] = 0$$

For $j_1, j_2, ..., j_p = i_1, i_2, ..., i_p$ we obtain

$$\mu(b_{i_1})[j_1] \cdot \ldots \cdot \mu(b_{i_p})[j_p] = 1$$

Hence we have

$$\mu(b_i)[j] = \sigma^i_{i_1,i_2,\ldots,i_p}$$

By Alg. 5 we have that $\sigma^i_{i_1,i_2,...,i_p} = 1$ and $\sigma^j_{i_1,i_2,...,i_p} = 0$ for $j \neq i$, so $\mu(b_i)[i] = 1$ and $\mu(b_i)[j] = 0$ for $j \neq i$. Hence $\mu(b_i) = e_i$ as required. □

The next lemma states for a tree $t = \sigma(b_{i_1}, b_{i_2}, ..., b_{i_p})$ with children in the basis, if $t \ltimes^{\alpha}_{H} b_i$ then $\mu(t) = \alpha \cdot e_i$ where e_i is the i'th standard basis vector.

Lemma 5.8. *Let $t = \sigma(b_{i_1}, b_{i_2}, ..., b_{i_p})$, s.t. $b_{i_j} \in B$ for $1 \leq j \leq p$. Assume $H[t] = \alpha \cdot H[b_i]$ for some i. Then $\mu(t) = \alpha \cdot e_i$.*

Proof. If $t = \sigma$ is a leaf, then by definition we have $\sigma_i = \alpha$ and $\sigma_j = 0$ for $j \neq i$, so $\mu(t) = \alpha \cdot e_i$. Otherwise, t isn't a leaf. Assume $t = \sigma(b_{i_1}, b_{i_2}, ..., b_{i_p})$. We thus have

$$\mu(t)[j] = \sum_{j_1,j_2,...,j_p \in [n]^p} \sigma^j_{j_1,j_2,...,j_p} \cdot \mu(b_{i_1})[j_1] \cdot \ldots \cdot \mu(b_{i_p})[j_p]$$

By Lemma 5.7 we have that $\mu(b_{i_j}) = e_{i_j}$ for $1 \leq j \leq p$, hence using a similar technique to the one used in the proof of Lemma 5.7 we obtain that for every $1 \leq j \leq p$:

$$\mu(t)[j] = \sigma^j_{i_1,i_2,...,i_p}$$

By Alg. 5 we have that $\sigma^j_{i_1,i_2,...,i_p} = \alpha$ for $i = j$ and $\sigma^j_{i_1,i_2,...,i_p} = 0$ for $i \neq j$, thus $\mu(t) = \alpha \cdot e_i$ as required. □

The following lemma generalizes the previous lemma to any tree $t \in T$.

Lemma 5.9. *Let H be a closed consistent sub-matrix of the Hankel Matrix. Then for every $t \in T$ s.t. $H[t] = \alpha \cdot H[b_i]$ we have $\mu(t) = \alpha \cdot e_i$*

Proof. By induction on the height of t. For the base case t is a leaf, and the claim holds by Lemma 5.8. Assume the claim holds for all trees of height at most h. Let t be a tree of height h. Then $t = \sigma(t_1, t_2, ..., t_p)$. Since T is prefix-closed, for every $1 \leq j \leq p$ we have that $t_j \in T$. And from the induction hypothesis for every $1 \leq j \leq p$ we have that $\mu(t_j) = \alpha_j \cdot e_{i_j}$. Hence

$$\begin{aligned}\mu(t) &= \mu(\sigma(t_1, t_2, ..., t_p)) = \mu_\sigma(\mu(t_1), \mu(t_2), ..., \mu(t_p)) \\ &= \mu_\sigma(\alpha_1 \cdot e_{i_1}, \alpha_2 \cdot e_{i_2}, ..., \alpha_p \cdot e_{i_p}) \\ &= \prod_{j=1}^{p} \alpha_j \cdot \mu_\sigma(e_{i_1}, e_{i_2}, ..., e_{i_p})\end{aligned}$$

Let $t' = \sigma(b_{i_1}, b_{i_2}, ..., b_{i_p})$. From Lemma 5.7 we have

$$\begin{aligned}\mu(t) &= \prod_{j=1}^{p} \alpha_j \cdot \mu_\sigma(e_{i_1}, e_{i_2}, ..., e_{i_p}) \\ &= \prod_{j=1}^{p} \alpha_j \cdot \mu_\sigma(\mu(b_{i_1}), \mu(b_{i_2}), ..., \mu(b_{i_p})) \\ &= \prod_{j=1}^{p} \alpha_j \cdot \mu(t')\end{aligned}$$

Since the table is consistent, we know that for each $1 \leq j \leq p$ and $c \in C$:

$$H[\sigma(t_1, t_2, ..., t_{j-1}, t_j, t_{j+1}, ..., t_p)][c] = \alpha_j \cdot H[\sigma(t_1, t_2, ..., t_{j-1}, b_{i_j}, t_{j+1}, ..., t_p)][c]$$

We can continue using consistency to obtain that

$$\begin{aligned} H[t][c] &= H[\sigma(t_1, t_2, ..., t_p)][c] \\ &= \prod_{j=1}^{p} \alpha_j \cdot H[\sigma(b_1, b_2, ..., , b_p)][c] \\ &= \prod_{j=1}^{p} \alpha_j \cdot H[t'][c] \end{aligned}$$

Thus $H[t] = \prod_{j=1}^{p} \alpha_j \cdot H[t']$. Let $\beta = \prod_{j=1}^{p} \alpha_j$, then $t \ltimes_H^{\beta} t'$. Let b be the element in the base s.t. $t' \ltimes_H^{\alpha} b_i$. From Lemma 5.8 we have that $\mu(t') = \alpha \cdot e_i$. Therefore $\mu(t) = \beta \cdot \alpha \cdot e_i$. We have $\mu(t) = \beta \cdot \alpha \cdot e_i$ and $t \ltimes_H^{\alpha \cdot \beta} b_i$. Therefore the claim holds. □

We are now ready to show that for every tree $t \in T$ and context $c \in C$ the obtained CMTA agrees with the observation table.

Lemma 5.10. *For every $t \in T$ and for every $c \in C$ we have that $\mathcal{A}(c[\![t]\!]) = H[t][c]$.*

Proof. Let $t \in T$. Since the table is closed, there exists $b_i \in B$ such that $t \ltimes_H^{\alpha} b_i$ for some $\alpha \in \mathbb{R}_+$. The proof is by induction on the depth of $\diamond$ in c. For the base case, the depth of c is 1, so $c = \diamond$, and by Lemma 5.9 we have that $\mu(c[\![t]\!]) = \mu(t) = \alpha \cdot e_i$. Therefore $\mathcal{A}(t) = \alpha \cdot e_i \cdot \lambda$. By Alg. 5 we have that $\lambda[i] = H[b_i][\diamond]$. Thus $\mathcal{A}(t) = \alpha \cdot H[b_i][\diamond] = H[t][\diamond]$ as required.

For the induction step, let c be a context such that the depth of $\diamond$ is $h+1$. Hence $c = c'[\![\sigma(t_1, t_2, ..., t_{i-1}, \diamond, t_i, ..., t_p)]\!]$ for some trees $t_j \in T$, and some context c' of depth h. For each $1 \leq j \leq p$, let b_{i_j} be the element in the base, s.t. $t_j \equiv_H b_{i_j}$, with co-efficient α_j. Let b be the element in the base s.t. $t \equiv_H b$ with coefficient α. Let $\widetilde{t}$ be the tree:

$$\widetilde{t} = \sigma(b_{i_1}, b_{i_2}, ..., b_{i_{k-1}}, b, b_{i_k}, ..., b_{i_p})$$

Note that $\widetilde{t} \in \Sigma(B)$ and hence $\widetilde{t} \in T$. From the induction hypothesis, we obtain:

$$\mathcal{A}(c'[\![\widetilde{t}]\!]) = H[\widetilde{t}][c']$$

Since the table is consistent, we have:

$$H[t][c] = H[\sigma(t_{i_1}, t_{i_2}, ..., t_{i_{k-1}}, t, t_{i_k}, ..., t_{i_p})][c'] = \alpha \cdot \prod_{i=1}^{p} \alpha_i \cdot H[\widetilde{t}][c']$$

Let $\beta = \alpha \cdot \prod_{i=1}^{p} \alpha_i$. By definition of $\mathcal{A}$ we have:

$$\mathcal{A}(c'[\![\sigma(b_{i_1}, b_{i_2}, ..., b_{i_{k-1}}, b, b_{i_k}, ..., b_{i_p})]\!]) = \mu(c'[\![\sigma(b_{i_1}, b_{i_2}, ..., b_{i_{k-1}}, b, b_{i_k}, ..., b_{i_p})]\!]) \cdot \lambda$$

Since each t_{i_j} is in T, from Proposition 5.9 we have that $\mu(t_{i_j}) = \alpha_j \cdot b_{i_j}$, and that $\mu(t) = \alpha \cdot \mu(b)$. Let $\hat{t} = \sigma(t_{i_1}, t_{i_2}, ..., t_{i_{k-1}}, t, t_{i_k}, ..., t_{i_p}))$. So

$$\begin{aligned} \mu(\hat{t}) &= \mu(\sigma(t_{i_1}, t_{i_2}, ..., t_{i_{k-1}}, t, t_{i_k}, ..., t_{i_p}))) \\ &= \mu_\sigma(\alpha_1 \cdot b_{i_1}, ..., \alpha_{k-1} \cdot b_{i_{k-1}}, \alpha \cdot b, \alpha_k \cdot b_{i_k}, ..., \alpha_p \cdot b_{i_p}) \\ &= \alpha \cdot \prod_{j=1}^{p} \alpha_i \cdot \mu_\sigma(b_{i_1}, ..., b_{i_{k-1}}, b, b_{i_k}, ..., b_{i_p}) = \beta \cdot \mu(\widetilde{t}) \end{aligned}$$

By Lemma 5.3 we have that

$$\mu(c'[\![\sigma(t_{i_1}, t_{i_2}, ..., t_{i_{k-1}}, t, t_{i_k}, ..., t_{i_p})]\!]) = \beta \cdot \mu(c'[\![\widetilde{t}]\!])$$

Hence

$$\begin{aligned}\mathcal{A}(c[\![t]\!]) &= \mathcal{A}(c'[\![\sigma(t_{i_1}, t_{i_2}, ..., t_{i_{k-1}}, t, t_{i_k}, ..., t_{i_p})]\!]) \\ &= \mu(c'[\![\sigma(t_{i_1}, t_{i_2}, ..., t_{i_{k-1}}, t, t_{i_k}, ..., t_{i_p})]\!]) \cdot \lambda = \beta \cdot \mu(c'[\![\widetilde{t}]\!]) \cdot \lambda\end{aligned}$$

Note that all the children of $\widetilde{t}$ are in B, and so $\widetilde{t} \in T$. Hence, from the induction hypothesis we have

$$H[\widetilde{t}][c'] = \mathcal{A}(c'[\![\widetilde{t}]\!]) = \mu(c'[\![\widetilde{t}]\!]) \cdot \lambda$$

Thus

$$\mathcal{A}(c[\![t]\!]) = \beta \cdot H[\widetilde{t}][c'] = H[t][c]$$

as required. □

6. Discussion

The quest to learn probabilistic automata and grammars is still ongoing. Because of the known hardness results some restrictions need to be applied. Recent works include an $\mathbf{L}^*$ learning algorithm for MDPs [TAB+21] (here the assumption is that states of the MDPs generate an observable output that allows identifying the current state based on the generated input-output sequence), a passive learning algorithm for a subclass of PCFGs obtained by imposing several structural restrictions [CF20, Cla21], and using PDFA learning to obtain an interpretable model of practically black-box models such as recurrent neural networks [WGY19].

We have presented an algorithm for learning structurally unambiguous PCFGs from a given black-box language model using structured membership and equivalence queries. To our knowledge this is the first algorithm provided for this question. Following the motivation of [WGY19], the present work offers obtaining intrepretable models also in cases where the studied object exhibits non-regular (yet context-free) behavior. For future work, we think that improving our method to be more noise-tolerant would make the algorithm able to learn complex regular and context-free grammars from recurrent neural networks.

References

[AK95] D. Angluin and M. Kharitonov. When won't membership queries help? *J. Comput. Syst. Sci.*, 50(2):336–355, 1995.

[AMP99] Steven Abney, David McAllester, and Fernando Pereira. Relating probabilistic grammars and automata. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 542–549, 1999.

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[Ang90] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[AW92] N. Abe and M. K. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260, 1992.

[Bak79] J. K. Baker. Trainable grammars for speech recognition. In D. H. Klatt and J. J. Wolf, editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550, 1979.

[BBB+00] A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000.

[BV96] F. Bergadano and S. Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM J. Comput.*, 25(6):1268–1280, 1996.

[CF20] Alexander Clark and Nathanaël Fijalkow. Consistent unsupervised estimators for anchored pcfgs. *Trans. Assoc. Comput. Linguistics*, 8:409–422, 2020.

[Cho56] Noam Chomsky. Three models for the description of language. *IRE Trans. Inf. Theory*, 2(3):113–124, 1956.

[Chu88] Kenneth Ward Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Second Conference on Applied Natural Language Processing*, pages 136–143, Austin, Texas, USA, February 1988. Association for Computational Linguistics.

[Cla21] Alexander Clark. Beyond chomsky normal form: Extending strong learning algorithms for pcfgs. In Jane Chandlee, Rémi Eyraud, Jeff Heinz, Adam Jardine, and Menno van Zaanen, editors, *Proceedings of the Fifteenth International Conference on Grammatical Inference*, volume 153 of *Proceedings of Machine Learning Research*, pages 4–17. PMLR, 23–27 Aug 2021.

[CR93] Joel E Cohen and Uriel G Rothblum. Nonnegative ranks, decompositions, and factorizations of nonnegative matrices. *Linear Algebra and its Applications*, 190:149–168, 1993.

[DH07] Frank Drewes and Johanna Högberg. Query learning of regular tree languages: How to avoid dead states. *Theory of Computing Systems*, 40(2):163–185, 2007.

[dlH10] C. de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, USA, 2010.

[Gol78] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[Gra95] Leslie Grate. Automatic RNA secondary structure determination with stochastic context-free grammars. In Christopher J. Rawlings, Dominic A. Clark, Russ B. Altman, Lawrence Hunter, Thomas Lengauer, and Shoshana J. Wodak, editors, *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology, Cambridge, United Kingdom, July 16-19, 1995*, pages 136–144. AAAI, 1995.

[HO06] Amaury Habrard and Jose Oncina. Learning multiplicity tree automata. In *International Colloquium on Grammatical Inference*, pages 268–280. Springer, 2006.

[HU79] John E. Hopcroft and Jeff D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

[LJ78a] Leon S. Levy and Aravind K. Joshi. Skeletal structural descriptions. *Information and Control*, 39(2):192 – 211, 1978.

[LJ78b] Leon S. Levy and Aravind K. Joshi. Skeletal structural descriptions. *Information and Control*, 39(2):192 – 211, 1978.

[LY90] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[NFZ21] Dolav Nitay, Dana Fisman, and Michal Ziv-Ukelson. Learning of structurally unambiguous probabilistic grammars. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 9170–9178, 2021.

[Rab89] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *PROCEEDINGS OF THE IEEE*, pages 257–286, 1989.

[Reg16] Rommel G. Regis. On the properties of positive spanning sets and positive bases. *Optimization and Engineering*, 17(1):229–262, Mar 2016.

[Sak88] Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. In *Proceedings of the First Annual Workshop on Computational Learning Theory, COLT '88, Cambridge, MA, USA, August 3-5, 1988*, pages 330–344, 1988.

[Sak92] Yasubumi Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Inform. Comput.*, 97:23–60, 1992.

[SJ07] Noah A Smith and Mark Johnson. Weighted and probabilistic context-free grammars are equally expressive. *Computational Linguistics*, 33(4):477–491, 2007.

[TAB$^+$21] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. L*-based learning of markov decision processes (extended version). *Formal Aspects Comput.*, 33(4):575–615, 2021.

[vHKRS20] Gerco van Heerdt, Clemens Kupke, Jurriaan Rot, and Alexandra Silva. Learning weighted automata over principal ideal domains. In *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, pages 602–621, 2020.

[WGY19] Gail Weiss, Yoav Goldberg, and Eran Yahav. Learning deterministic weighted automata with queries and counterexamples. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 8558–8569, 2019.