Learning of Structurally Unambiguous Probabilistic Grammars

Dolav Nitay Dana Fisman Michal Ziv-Ukelson

November 17, 2021

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. We demonstrate the usefulness of our algorithm in learning PCFGs over genomic data.

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 [11], natural language processing [12] and biological modeling, for instance, in probabilistic modeling of RNA structures [20]. Methods for learning PCFGs from experimental data have been thought 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 [19] that context-free grammars (CFGs) cannot be identified in the limit from positive examples, and from [5] that CFGs cannot be identified in polynomial time using equivalence queries only. Both results are not surprising for ones familiar with learning regular languages, as they hold for the class of regular languages as well. However, while regular languages can be learned using both membership queries and equivalence queries [4], 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 [6]. See more on the difficulties of learning context-free grammars in [14, Chapter 15]. Hardness results for the probabilistic setting have also been established. [1] 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 $\mathbf{RP} = \mathbf{NP}$.

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 [33] and the inside-outside algorithm for PCFGs [7, 30].

In this work we address the first problem. Due to the hardness results regarding learning probabilistic grammars using membership and equivalence queries (MQ and EQ) we use structured membership queries and structured equivalence queries (SMQ and SEQ), as was done by [35] for learning context-free grammars. *Structured strings*, proposed by [31], are strings over the given alphabet that includes parenthesis 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*. [35] 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. 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 (up to a predefined error margin) along with the probability p with which the unknown grammar $\mathcal{G}$ generates s.

[35] works with tree automata to model the derivation trees of the unknown grammars. 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 to as $\mathbf{M}^*$, was developed in [21, 15].[1]

A probabilistic grammar is a special case of a weighted grammar and [2, 38] have shown that convergent weighted CFGs (WCFG) 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 3.1, 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 3.2.[3] To overcome these issues we restrict attention to structurally unambiguous grammars (SUCFG, see section 3.1), 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 structurally unambiguous PCFGs using SMQs and SEQs as desired.

We demonstrate the capabilities of our learning algorithm on datasets of genomic data. An important problem in functional genomics is to learn grammar representations of gene clusters. A Gene cluster is a group of genes that are co-locally conserved, not necessarily in the same order, across many genomes [44]. The gene grammar corresponding to a given gene cluster describes its hierarchical inner structure and the relations between instances of the cluster succinctly; assists in predicting the functional association between the genes in the cluster; provides insights into the evolutionary history of the cluster; aids in filtering meaningful from apparently meaningless clusters; and provides a natural and meaningful way of visualizing complex clusters.

PQ trees have been advocated as a representation for gene-grammars [10, 9]. A PQ-tree represents the possible permutations of a given sequence, and can be constructed in polynomial-time [29]. A PQ-tree is a rooted tree with three types of nodes: *P-nodes*, *Q-nodes* and leaves. In the gene grammar inferred by a given PQ-tree, the children of a P-node can appear in any order, while the children of a Q-node must appear in either left-to-right or right-to-left order.

However, the PQ tree model suffers from limited specificity, which often does not scale up to encompass gene clusters. It also does not model tandem gene-duplications, which are a common event in the evolution of gene-clusters. We exemplify how our algorithm can learn a grammar that addresses both of these problems. Using the more general model of context free grammar, we can model evolutionary events that PQ-trees cannot, such as tandem gene-duplications. While the probabilities in our PCFG grant our approach the capability to model rare-occurring permutations (and weighing them as such), thus creating a specificity which PQ-trees lack.

Due to lack of space all proofs are deferred to appendix (App. B). The appendix also contains (i) a complete running example (App. A) and (ii) supplementary material for the demonstration section (App. C).

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.

1.1 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. The formal definition starts with a weighted grammar. We assume the reader is familiar with the standard definition of a context-free grammar (CFG) as a four tuple $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S \rangle$ 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 $\mathcal{W}(t) = \prod_{(V \to \alpha) \in R} \theta(V \to \alpha)^{\sharp_t(V \to \alpha)}$ where $\sharp_t(V \to \alpha)$ is the number of occurrences of the production $V \to \alpha$ in the derivation tree t. If the sum of all derivation trees in $[\![\mathcal{G}]\!]$ is finite, we say that $\mathcal{W}$ is *convergent*. 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 $\mathcal{W}(w) = 0$.

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

[1]Following a learning algorithm developed for multiplicity word automata [8].

[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.

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 $\mathcal{P}$ is convergent and $\mathcal{P}(w) \in [0,1]$ for every $w \in \Sigma^*$.

1.2 Word/Tree Series and Multiplicity Automata

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 union of alphabets Σ_k where Σ_0 is non-empty. Let *Trees*(Σ) 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 *Trees*(Σ)) to $\{0,1\}$. We are interested in assigning each word or tree a non-Boolean value, usually a weight $p \in [0,1]$. Let $\mathbb{K}$ be a field. 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 *Trees*(Σ) to $\mathbb{K}$, is referred to as a *tree series*.

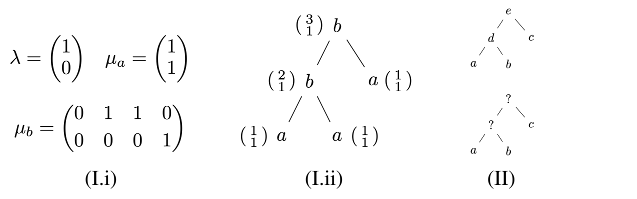
Word automata are machines that recognize word language, i.e. they define a function from Σ^* to $\{0,1\}$. *Tree automata* are machines that recognize tree language, i.e. they define a function from *Trees*(Σ) to $\{0,1\}$. *Multiplicity word automata* (MA) are machines to implement word series, i.e. they define a function from Σ^* to $\mathbb{K}$. *Multiplicity tree automata* (MTA) are machines to implement tree series, i.e. they define a function from *Trees*(Σ) to $\mathbb{K}$. Multiplicity automata can be thought of as an algebraic extension of automata, in which transition passage 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.

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 $\mathbb{K}$. 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 $c^i_{j_1 j_2 j_3} \in \mathbb{K}$ 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 $c^i_{j_1 j_2 \ldots j_k}$ then

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

$$M_\eta = \begin{pmatrix} c^1_{111} & c^1_{112} & c^1_{121} & c^1_{122} & c^1_{211} & c^1_{212} & c^1_{221} & c^1_{222} \\ c^2_{111} & c^2_{112} & c^2_{121} & c^2_{122} & c^2_{211} & c^2_{212} & c^2_{221} & c^2_{222} \end{pmatrix} \quad P_{xyz} = \begin{pmatrix} x_1 y_1 z_1 \\ x_1 y_1 z_2 \\ x_1 y_2 z_1 \\ \ldots \\ x_2 y_2 z_2 \end{pmatrix}$$

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



$\lambda = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ $\mu_a = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
$\mu_b = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
$\binom{3}{1}$ b
$\binom{2}{1}$ b
a $\binom{1}{1}$
$\binom{1}{1}$ a
a $\binom{1}{1}$
e
d
c
a
b
?
?
c
a
b
(I.i)
(I.ii)
(II)

Figure 2: (I.i) 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. (I.ii) a tree where a node t is annotated by $\mu(t)$. Since $\mu(t_\epsilon) = \binom{3}{1}$, where t_ϵ is the root, the value of the entire tree is $\lambda \cdot \binom{3}{1} = 3$. (II) a derivation tree and its corresponding skeletal tree.

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 the respective field, 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 $0 \leq k \leq p$ 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 $\mathcal{M}(t) = \lambda \cdot \mu(t)$. Fig. 2(I.i) provides an example of an MTA and the value for a computed tree Fig. 2(I.ii).

Contexts In the course of the algorithm we need a way 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 *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. We refer to an element of *Trees*$_\diamond(\Sigma)$ as a *context*. Note that at most one child of any node

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

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* which are strings with parenthesis exposing the structure of a derivation tree for the corresponding trees. These are defined formally as follows (and depicted in Fig. 2 (II)). 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 $?$. Let $t \in \mathit{Trees}(\Sigma)$, 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 $\{t \in T \mid \mathsf{S}(t) = s\}$.

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, 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.

2 From Positive MTAs to PCFGs

Our learning algorithm for probabilistic grammars builds on the relation between WCFGs with positive weights and PCFGs [2, 38]. In particular, we first establish that a *positive multiplicity tree automaton* (PMTA), which is a multiplicity tree automaton (MTA) where all weights 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 [2, 38] to yield a PCFG $\mathcal{P}$ such that $\mathcal{W}(s) = \mathcal{P}(s)$, obtaining the desired PCFG for the unknown tree series.

Transforming a PMTA into WCFG 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 WCFG $\mathcal{W}_\mathcal{A} = (\mathcal{G}_\mathcal{A}, \theta)$ for $\mathcal{G}_\mathcal{A} = (\mathcal{V}, \Sigma_0, R, S)$ as provided in Fig. 3 where $c^i_{i_1, i_2, \ldots, i_p}$ is the respective coefficient in the matrix corresponding to $\mu_?$ for $? \in \Sigma_p$. The following proposition states that the transformation preserves the weights.

Proposition 2.1. $\mathcal{W}(t) = \mathcal{A}(t)$ *for every* $t \in \mathit{Trees}(\Sigma)$.

$$\begin{array}{ll}
\mathcal{V} = \{S\} \cup \{V_i \mid 1 \leq i \leq d\} & \\
R = \{S \to V_i \mid 1 \leq i \leq d\} \cup & \theta(S \to V_i) = \lambda[i] \\
\quad \{V_i \to \sigma \mid 1 \leq i \leq d,\ \sigma \in \Sigma_0\} \cup & \theta(V_i \to \sigma) = \mu_\sigma[i] \\
\quad \left\{ V_i \to V_{i_1} V_{i_2} \ldots V_{i_k} \;\middle|\; \begin{array}{l} 1 \leq i, i_j \leq d, \\ \text{for } 1 \leq j \leq k \end{array} \right\} & \theta(V_i \to V_{i_1} V_{i_2} \ldots V_{i_k}) = c^i_{i_1, i_2, \ldots, i_p}
\end{array}$$

Figure 3: Transforming a PMTA into a PCFG

In Section 4, Thm. 4.1, 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 2.2. *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 2.2, on the complexity of converting a WCFG into a PCFG [2], for which an exact bound is not provided, but the method is reported to converge quickly [38, §2.1].

3 Learning Struc. Unamb. PCFGs

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 [35]), 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 $\text{SMQ}(s)$ is answered with the value $\mathcal{T}(s)$. Given an automaton $\mathcal{A}$ the query $\text{SEQ}(\mathcal{A})$ 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 $\mathbf{M}^*$ [21] 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 the algorithm itself.

Hankel Matrix All the generalizations of $\mathbf{L}^*$ (the algorithm for learning regular languages using MQs and EQs, that introduced this learning paradigm [4]) share a general idea that can be explained as follows. A word or tree language as well as 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 $c_i \in \mathit{Trees}_\diamond(\Sigma)$). The entry $H(i, j)$ holds in the case of words the value for the word $w_i \cdot w_j$ and in the case of trees 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 equivalent to rows in the basis. In the case of $\mathbf{M}^*$(that learns tree-series) rows not in the basis should be expressible as a linear combination of rows in the basis. In our case, in order to apply the PMTA to PCFG conversion we need the algorithm to find a *positive linear combination* of rows to act as the basis. 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.

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 [13, 34].[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:

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

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 $\textit{c-rank}_+(A)$ (resp. $\textit{r-rank}_+(A)$), 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 is known that $\textit{c-rank}_+(A) = \textit{r-rank}_+(A)$ for any matrix A [13]. Thus one can freely use $\textit{rank}_+(A)$ for *positive rank*, to refer to either one of these.

Issues with positive spans The first question that comes to mind, is whether we can use the $\mathbf{M}^*$ algorithm as is to learn a positive tree series. We show that this is not the case. In particular, there are positive tree series for which applying the $\mathbf{M}^*$ algorithm results in an MTA with negative weights. Moreover, this holds also if we consider word (rather than tree) series, and if we restrict the weights to be probabilistic (rather than simply positive).

Proposition 3.1. *There exists a probabilistic word series for which the $\mathbf{M}^*$ alg. may return an MTA with negative weights.*

The proof shows this is the case for the word series over alphabet $\Sigma = \{a, b, c\}$ which assigns the following six strings: aa, ab, ac, ba, cb, cc probability of $\frac{1}{6}$ each, and probability 0 to all other strings.

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

Hence, 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 $\mathbb{R}^d_+$ [34]. 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 3.2. *There exists a PCFG $\mathcal{G}$ s.t. the Hankel Matrix $H_{\mathcal{G}}$ corresponding to its tree-series $\mathcal{T}_G$ has the property that no finite number of rows positively spans the entire matrix.*

The proof shows this is the case for the following PCFG:

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

3.1 Focusing on Strucutrally Unambiguous CFGs

To overcome these obstacles we restrict attention to strucutrally 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 structuraly 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 [23, 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 strucutrally 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 $t_1 \ltimes^{\alpha}_{H} t_2$ 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 $t_1 \ltimes^{\alpha}_{H} t_2 \ltimes^{\alpha}_{H} t_1$ for every $\alpha > 0$. We say that $t_1 \equiv_H t_2$ if $t_1 \ltimes^{\alpha}_{H} t_2$ 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 3.3. *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 $t_1 \ltimes^{\alpha}_{H} t_2$ for some $\alpha \neq 0$.*

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

Next we would like to reveal the restrictions that can be emposed on a PMTA that corresponds to an SUPCFG. We term an MTA *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.

Proposition 3.5. *A CMTA can represent an SUPCFG.*

The proof relies on showing that a WCFG is strucuturally unambiguous iff it is invertible and converting an invertible WCFG into a PMTA yields a CMTA.[6]

4 The Learning Algorithm

et $\mathcal{T}$: *Trees*$(\Sigma) \to \mathbb{R}$ be an unknown tree series, and let H 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 $T \subseteq$ *Trees*(Σ) is a set of row titles, $C \subseteq$ *Trees*$_\diamond(\Sigma)$ is a set of column titles, $H : T \times C \to \mathbb{R}$ is a sub-matrix of H, 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.

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     $t \leftarrow$ SEQ$(\mathcal{A})$
6     if $t$ is null then
7         return $\mathcal{A}$
8     $Complete(T, C, H, B, Pref(t))$
```

Algorithm 2 *Consistent*(T, C, H, B)

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

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*$(T, C, H, B)$ checks if $H[t][C]$ is 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 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 $H[t] = \bar{0}$ it holds that $H[c[\![t']\!]] = \bar{0}$ for every $t' \in \Sigma(T, t)$ and $c \in C$. It is said to be *co-linear consistent* if for every $t_1, t_2 \in T$ s.t. $t_1 \ltimes^{\alpha}_{H} t_2$ and every context $c \in \Sigma(T, \diamond)$ we have that $c[\![t_1]\!] \ltimes^{\alpha}_{H} c[\![t_2]\!]$. The procedure *Consistent* looks for trees which violate the zero-consistency or co-linear consistency requirement, and for every violation, the respective context is added to C.

The procedure *Complete*(T, C, H, B, S) 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. 3

Overall we can show that the algorithm always terminates, returning a correct CMTA whose dimension is minimal, namely it equals the rank n of 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 the a symbol in Σ. Hence for a grammar in Chomsky Normal Form, where $p = 2$ it is polynomial in all parameters.

Theorem 4.1. *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 algo-*

[6]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 $A = B$ [36].

rithm makes at most $n \cdot (n + m \cdot n + |\Sigma| \cdot (n + m \cdot n)^p)$ SMQs and at most n SEQs.

Algorithm 3 *ExtractCMTA*(M, T, C, B).

```
1  Let $d = |B|$
2  for $0 \leq k \leq p$ do
3      for $\sigma \in \Sigma_k$ do
4          for $(i_1, i_2, ..., i_k) \in \{1, 2, \ldots, d\}^k$ do
5              Let $t_{i_1, i_2, \ldots, i_k} = \sigma(b_{i_1}, b_{i_2}, ..., b_{i_k})$
6              if $H[t] = \bar{0}$ then
7                  for $1 \leq j \leq d$ do
8                      $\sigma^j_{i_1, i_2, \ldots, i_k} \leftarrow 0$
9              else
10                 Let $b_i \in B$, $\alpha \in \mathbb{R}$ be s.t. $t_{i_1 \ldots i_k} \ltimes^{\alpha}_{H} b_i$
11                 for $1 \leq j \leq d$ do
12                     if $j = i$ then
13                         $\sigma^j_{i_1, i_2, \ldots, i_k} \leftarrow \alpha$
14                     else
15                         $\sigma^j_{i_1, i_2, \ldots, i_k} \leftarrow 0$
16         Let $\mu_\sigma$ be the $d \times d^k$ matrix obtained from the
               respective coefficients
17 for $1 \leq j \leq d$ do
18     $\lambda[j] \leftarrow H(\diamond, b_j)$
19 return $\mathcal{A} = (\Sigma, \mathbb{R}, d, \mu, \lambda)$        ▷ where $\mu = \{\mu_\sigma\}_{\sigma \in \Sigma}$
```

5 Demonstration

In our preliminary results, we apply our algorithm to the learning of gene cluster grammars.[7] Gene-clusters in prokaryotic genomes often correspond to (one or several) operons; those are neighboring genes that constitute a single unit of transcription and translation [24].

We represent gene-cluster grammars as CFG's, since those are robust enough to model the typical events in operon evolution, including in-tandem gene duplications [32, 27] and progressive merging of sub-operons [16, 17]. The former event cannot be modelled by a PQ-tree.[8]

Given a set of trees $T \subseteq \mathit{Trees}(\Sigma)$, a probability distribution p on T, an edit distance function $d : \mathit{Trees}(\Sigma) \times \mathit{Trees}(\Sigma) \to \mathbb{N} \cup \{\infty\}$, and a *decay-factor* $q \in [0, 1]$, we define the tree-series $\mathcal{T}_T : \mathit{Trees}(\Sigma) \to [0, 1]$ that describes a gene-cluster grammar on T:

$$\mathcal{T}_T(t) = \max_{t' \in T} p(t') \cdot q^{d(t,t')} \;;\; q^{\infty} = 0 \text{ and } d(t, t) = 0$$

We give two examples of gene-cluster grammars. The first is a PCFG describing a gene cluster corresponding to a multi-drug efflux pump (MDR). MDR's are used by some bacteria as a mechanism for antibiotic resistance, and hence are the focus of research aimed towards the development of new therapeutic strategies. In this example, a swap-event counting measure was used as edit-distance – modeling distinctly ordered merge events of sub-operons. The resulting learned gene-cluster grammar is illustrated in Fig. 4. A biological interpretation of the learned grammar, associating the highly probable

[7]See definition of gene cluster grammars in the introduction

[8]See definition of PQ-trees in the introduction

$S \to$ R N7 [0.456] | R N3 [0.185] | C N1 [0.137] | N7 R [0.078] |
R N6 [0.053] | N1 C [0.034] | N5 R [0.028] | N6 R [0.013] |
N3 R [0.012] | R N5 [0.004]
$N1 \to$ R N2 [0.640] | N2 R [0.160] | R N4 [0.160] | N4 R [0.040]
$N2 \to$ A B [1.000] $N3 \to$ C N2 [1.000] $N4 \to$ B A [1.000]
$N5 \to$ N4 C [1.000] $N6 \to$ C N4 [0.882] | N8 A [0.095] | A N8 [0.024]
$N7 \to$ N2 C [1.000] $N8 \to$ B C [0.800] | C B [0.200]
$R \to$ AcrR [1.000] $A \to$ AcrA [1.000] $B \to$ AcrB [1.000] $C \to$ TolC [1.000]

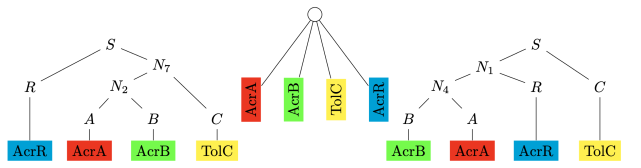


Figure 4: The PCFG learned from the MDR dataset. Also shown are the most probable tree according to the grammar (left) with $p = 0.456$, a non-probable tree (right) with $p = 0.001$, and the PQ-tree (middle) showing a complete lack of specificity.

rules with possible evolutionary events that explain them, is given in App. C.

Note that, in contrast to the highly specific PCFG learned by our algorithm, the PQ-tree constructed for this gene cluster (see Fig. 4) places all leaves under a single P-node, resulting in a complete loss of specificity regarding conserved gene orders and hierarchical swaps.

The second example of a learned gene cluster grammar, exemplifying tandem duplication events, is given in App. C. Since this grammar was created with a duplication-counting edit-distance measure, the yielded grammar demonstrates a learning of an infinite language, with exponentially decaying probabilities. Formal definitions of both edit-distance measures are given in App. C, along with the methods and datasets used in this section.

6 Discussion

We have presented algorithms 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. A recent paper [42] advocates one can obtain an interpretable model of practically black-box models such as recurrent neural networks, using PDFA learning. The present work extends on this and offers obtaining interpretable models also in cases where the studied object exhibits non-regular (yet context-free) behavior, as is the case, e.g. in Gene Cluster grammars.

References

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

[2] Steven Abney, David McAllester, and Fernando Pereira. Relating probabilistic grammars and automata. In *Pro-*

ceedings of the 37th Annual Meeting of the Association for Computational Linguistics, pages 542–549, 1999.

[3] Carolina Alvarez-Ortega, Jorge Olivares, and José L Martínez. Rnd multidrug efflux pumps: what are they good for? *Frontiers in microbiology*, 4:7, 2013.

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

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

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

[7] 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.

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

[9] Anne Bergeron. Formal models of gene clusters. 2008.

[10] Kellogg S Booth and George S Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

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

[12] 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.

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

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

[15] 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.

[16] Renato Fani, Matteo Brilli, and Pietro Lio. The origin and evolution of operons: the piecewise building of the proteobacterial histidine operon. *Journal of molecular evolution*, 60(3):378–390, 2005.

[17] Marco Fondi, Giovanni Emiliani, and Renato Fani. Origin and evolution of operons and metabolic pathways. *Research in microbiology*, 160(7):502–512, 2009.

[18] Qiang Ge, Yoichi Yamada, and Helen Zgurskaya. The c-terminal domain of acra is essential for the assembly and function of the multidrug efflux pump acrab-tolc. *Journal of bacteriology*, 191(13):4365–4371, 2009.

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

[20] 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.

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

[22] Katsuhiko Hayashi, Ryosuke Nakashima, Keisuke Sakurai, Kimie Kitagawa, Seiji Yamasaki, Kunihiko Nishino, and Akihito Yamaguchi. Acrb-acra fusion proteins that act as multidrug efflux transporters. *Journal of bacteriology*, 198(2):332–342, 2016.

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

[24] François Jacob and Jacques Monod. Genetic regulatory mechanisms in the synthesis of proteins. *Journal of molecular biology*, 3(3):318–356, 1961.

[25] Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*, 1966.

[26] Jessica Kobylka, Miriam S Kuth, Reinke T Müller, Eric R Geertsma, and Klaas M Pos. Acrb: a mean, keen, drug efflux machine. *Annals of the New York Academy of Sciences*, 1459(1):38–68, 2020.

[27] Bernard Labedan and Monica Riley. Widespread protein sequence similarities: origins of escherichia coli genes. *Journal of bacteriology*, 177(6):1585–1588, 1995.

[28] Jean-Benoît Lalanne, James C Taggart, Monica S Guo, Lydia Herzel, Ariel Schieler, and Gene-Wei Li. Evolutionary convergence of pathway-specific enzyme expression stoichiometry. *Cell*, 173(3):749–761, 2018.

[29] Gad M Landau, Laxmi Parida, and Oren Weimann. Gene proximity analysis across whole genomes via pq trees1. *Journal of Computational Biology*, 12(10):1289–1306, 2005.

[30] 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.

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

[32] Edward B Lewis. Pseudoallelism and gene evolution. In *Cold Spring Harbor Symposia on Quantitative Biology*, volume 16, pages 159–174. Cold Spring Harbor Laboratory Press, 1951.

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

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

[35] 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.

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

[37] Xiaodong Shi, Muyuan Chen, Zhili Yu, James M Bell, Hans Wang, Isaac Forrester, Heather Villarreal, Joanita Jakana, Dijun Du, Ben F Luisi, et al. In situ structure and assembly of the multidrug efflux pump acrab-tolc. *Nature communications*, 10(1):1–6, 2019.

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

[39] Dina Svetlitsky, Tal Dagan, Vered Chalifa-Caspi, and Michal Ziv-Ukelson. Csbfinder: discovery of colinear syntenic blocks across thousands of prokaryotic genomes. *Bioinformatics*, 35(10):1634–1643, 2019.

[40] Dina Svetlitsky, Tal Dagan, and Michal Ziv-Ukelson. Discovery of multi-operon colinear syntenic blocks in microbial genomes. In press, 2020.

[41] Roman L Tatusov, Michael Y Galperin, Darren A Natale, and Eugene V Koonin. The cog database: a tool for genome-scale analysis of protein functions and evolution. *Nucleic acids research*, 28(1):33–36, 2000.

[42] 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.

[43] Natasha Weston, Prateek Sharma, Vito Ricci, and Laura JV Piddock. Regulation of the acrab-tolc efflux pump in enterobacteriaceae. *Research in microbiology*, 169(7-8):425–431, 2018.

[44] Sascha Winter, Katharina Jahn, Stefanie Wehner, Leon Kuchenbecker, Manja Marz, Jens Stoye, and Sebastian Böcker. Finding approximate gene clusters with gecko 3. *Nucleic Acids Research*, 44(20):9600–9610, 2016.

[45] Daniel H Younger. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208, 1967.

A 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:

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

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 blue), resulting in the following observation table.

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

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 following table. From now on we omit 0 rows of $\Sigma(T)$ for brevity.

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

The table is now closed but it is not zero-consistent, since we have $H[a] = \bar{0}$, 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 following table:

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

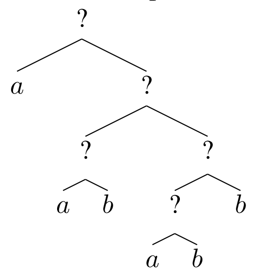
Note that t_4 was added to T since it wasn't spanned by T, but it is not a member of B, since $H[t_4] = 0$. We can extract the following CMTA $\mathcal{A}_1 = (\Sigma, \mathbb{R}, d, \mu, \lambda)$ of dimension $d = 3$ since $|B| = |\{t_1, t_2, t_3\}| = 3$. Let $\mathbb{V} = \mathbb{R}^3$. For the letters $\sigma \in \Sigma_0 = \{a, b\}$ 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. 3 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. 3 . 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. 3 , and we get $\lambda = (0, 0, 0.5)$.

The algorithm now asks an equivalence query and receives the following tree as a counter-example:

$p =$

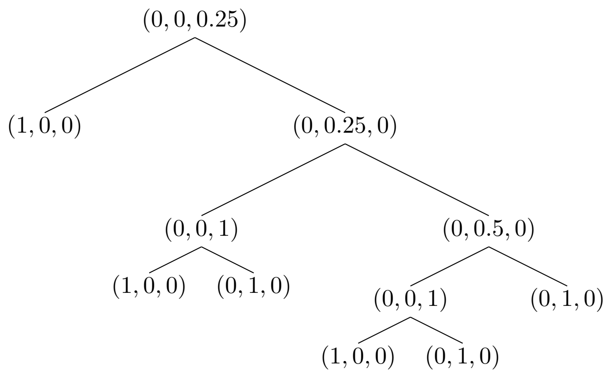


Indeed, while $\text{SMQ}(p) = 0$ we have that $\mathcal{A}(p) = 0.125$. To see why $\mathcal{A}(p) = 0.125$, let's look at the values $\mu(t)$ for every sub-tree t of p. For the leaves, we have $\mu(a) = (1,0,0)$ and $\mu(b) = (0,1,0)$.

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. 1.2, see also Fig. 2. The vector $P_{\mu(a),\mu(b)}$ is: $P_{\mu(a),\mu(b)} = (0,1,0,0,0,0,0,0,0)$. When multiplying this vector by the matrix $\eta_?$ we obtain $(0,0,1)$. So $\mu(?(a,b)) = (0,0,1)$. Similarly, to obtain $\mu(?(?(a,b),a))$ 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,

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

The following tree 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 counter-example to T and we obtain the following table:

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

This table is not consistent since while $t_6 \ltimes_H^{0,5} t_2$ this colinearity 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 table:

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

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 interesting entries are $c^3_{1,2} = 1$, $c^4_{3,2} = 1$ and $c^3_{1,4} = \frac{1}{2}$ since $t_{1,2} \ltimes_H^1 t_3$, $t_{3,2} \ltimes_H^1 t_5$, $t_{1,5} \ltimes_H^1 t_6 \ltimes_H^{0,5} t_3$. And for every other combination of unit-basis vectors we have $t_{i,j} \ltimes_H^1 t_4$. 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. 3 we obtain the following WCFG:

$$\begin{aligned} S &\longrightarrow N_3\ [0.5] \\ N_1 &\longrightarrow a\ [1.0] \\ N_2 &\longrightarrow b\ [1.0] \\ N_3 &\longrightarrow N_1N_2\ [1.0]\ \mid\ N_1N_4\ [0.5] \\ N_4 &\longrightarrow N_3N_2\ [1.0] \end{aligned}$$

Now, following [2, 38] we can calculate the partitions functions for each non-terminal. Let f_N be the sum of the weights of all trees whose root is N, we obtain:

$$\begin{aligned} f_S &= 1 \\ f_{N_1} &= 1 \\ f_{N_2} &= 1 \\ f_{N_3} &= 2 \\ f_{N_4} &= 2 \end{aligned}$$

Hence we obtain the PCFG

$$\begin{aligned} S &\longrightarrow N_3\ [1.0] \\ N_1 &\longrightarrow a\ [1.0] \\ N_2 &\longrightarrow b\ [1.0] \\ N_3 &\longrightarrow N_1N_2\ [0.5] \mid N_1N_4\ [0.5] \\ N_4 &\longrightarrow N_3N_2\ [1.0] \end{aligned}$$

which is a correct grammar for the unknown probabilistic series.

B Omitted Proofs

B.1 Proofs of Section 3

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 notation are used in the proof of Proposition 2.1. For a skeletal tree s and a non-terminal N we use $\mathcal{W}_N(s)$ for the weight of all derivation trees t in which the root is labeled by non-terminal N and s is their skeletal form.

Assume $\mathcal{G} = \langle \mathcal{V}, \Sigma, R, S \rangle$. Lemma B.1 below follows in a straight forward manner from the definition of $\mathcal{W}(\cdot)$ (given in SubSec. 1.1).

Lemma B.1. *Let $s = ?(s_1, s_2, \ldots, s_k)$. Then the following holds for each non-terminal $N \in \mathcal{V}$:*

$$\mathcal{W}_N(s) = \sum_{(X_1, X_2, \ldots, X_k) \in \mathcal{V}^k} \begin{array}{l} \theta(N \to X_1 X_2 \cdots X_k) \cdot \\ \mathcal{W}_{X_1}(s_1) \mathcal{W}_{X_2}(s_2) \cdots \mathcal{W}_{X_k}(s_k) \end{array}$$

Recall that the transformation from a PMTA to a WCFG (provided in Fig. 3) 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 B.2. *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 $1 \le i \le d$.*

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 MTAs computation. On the other hand, by the definition of the transformation in Fig. 3, 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. 3 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:

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

which according to Lemma B.1 is equal to $\mathcal{W}_{V_i}(s)$ as required. $\square$

We are finally ready to prove **Proposition 2.1**. which states that

$$\mathcal{W}(t) = \mathcal{A}(t) \quad \textit{for every } t \in \mathsf{S}(\mathit{Trees}(\Sigma)).$$

Proof. 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. 3 we have that $\lambda[i] = \theta(S \to V_i)$ for each i. So we have:

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

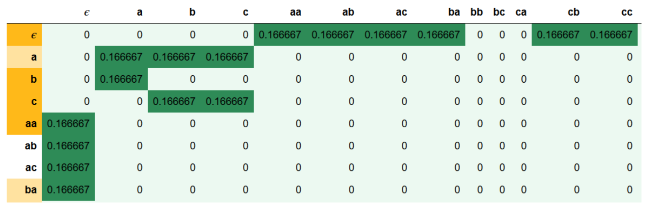
By our claim, for each i, v_i is equal to the probability of deriving t starting from the non-terminal V_i, so we have that the value calculated by $\mathcal{A}$ is the probability of deriving the tree starting from the start symbol S. That is, $\mathcal{W}(t) = \mathcal{W}_S(t) = \mathcal{A}(t)$. $\square$

B.2 Proofs of Section 4

We start with the proof of **Proposition 3.1**. which states that

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

Proof. The first rows of Hankel Matrix for this word series are given in the following figure (all entries not in the figure are 0). One can see that the rows ϵ, b, c, ba 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. Since the row of c is obtained by substracting the row of b from the row of a, this MTA will contain negative weights.



	ϵ	a	b	c	aa	ab	ac	ba	bb	bc	ca	cb	cc
ϵ	0	0	0	0	0.166667	0.166667	0.166667	0.166667	0	0	0	0.166667	0.166667
a	0	0.166667	0.166667	0.166667	0	0	0	0	0	0	0	0	0
b	0	0.166667	0	0	0	0	0	0	0	0	0	0	0
c	0	0	0.166667	0.166667	0	0	0	0	0	0	0	0	0
aa	0.166667	0	0	0	0	0	0	0	0	0	0	0	0
ab	0.166667	0	0	0	0	0	0	0	0	0	0	0	0
ac	0.166667	0	0	0	0	0	0	0	0	0	0	0	0
ba	0.166667	0	0	0	0	0	0	0	0	0	0	0	0

$\square$

Next, we would like to show that there exists a PCFG $\mathcal{G}$ such that the Hankel Matrix H_G corresponding to its tree-series $\mathcal{T}_G$ has the property that no finite number of rows spans the entire matrix.

We first prove the following lemma about positive indepedent sets.

Lemma B.3. *Let $B = \{b_1, b_2, ..., b_p\}$ be set of positively independent vectors. Let $\hat{B}$ be a matrix whose rows 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 each $b > 0$, and each $\alpha_j \geq 0$, with $\alpha_i > 1$.

Hence $\alpha_i = 1$. Therefore we have:

$$\sum_{j \neq i} \alpha_j b_j = 0$$

Since $\alpha_j \geq 0$, and $b_j > 0$ the only solution is that $\alpha_j = 0$ for every $j \neq i$. □

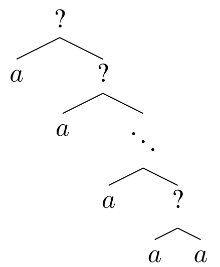
We are now ready to prove **Proposition 3.2**. which states:

> *There exists a PCFG $\mathcal{G}$ such that the Hankel Matrix $H_{\mathcal{G}}$ corresponding to its tree-series $\mathcal{T}_G$ has the property that no finite number of rows 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 $[\![G]\!]$ have a right-chain structure (and the terminals are always the letter a), and can be depicted as follows:



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's 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,

$$\begin{aligned}
\frac{3}{4}\cdot v_r-\frac{1}{24}\cdot v_{r-1}&=\alpha\hat{B}\\
v_r&=\frac{4}{3}\cdot\alpha\hat{B}+\frac{1}{18}\cdot v_{r-1}
\end{aligned}$$

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 B.3 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 provide here the proof of **Proposition 3.3**. which states that

Let H be the Hankel Matrix of an SUWCFG. Let t_1,t_2 be derivation trees rooted by the same non-terminal N_i. Assume $\mathbb{P}(t_1),\mathbb{P}(t_2)>0$. Then $t_1\ltimes^{\alpha}_{H}t_2$ 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. u and v might be ε. We denote by $\mathbb{P}_i(c)$ the probability of deriving this context, 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. So we obtain:

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

So we obtain, for every context c, assuming that $P_i(c)\neq 0$:

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

For a context c s.t. $P_i(c)=0$ we obtain that $\mathbb{P}(c[\![t_1]\!])=\mathbb{P}(c[\![t_2]\!])=0$. So for every context:

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

So $H[t_1]=\alpha\cdot H[t_2]$ for $\alpha=\frac{\mathbb{P}(t_1)}{\mathbb{P}(t_2)}$, so $H[t_1]$ and $H[t_2]$ are co-linear, and $t_1\ltimes^{\alpha}_{H}t_2$. □

We turn to prove **Corollary 3.4**. which states that

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 s.t. $\mathsf{S}(t)=s$. So for every s there is a single possible tagging, and a single possible non-terminal in the root. By Proposition 3.3 every two 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 $t\in \mathit{Trees}$ s.t. $\mathbb{P}(t)=0$. Since there is a finite number of non-terminals, there is a finite number of equivalence classes under $\equiv_H$. □

To prove Proposition 3.5 we first show how to convert a WCFG into a PMTA. Then we claim, that in case the WCFG is structurally unambiguous the resulting PMTA is a CMTA.

Converting a WCFG to a PMTA Let $\langle \mathcal{G}, \theta \rangle$ be a WCFG 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

$$\lambda = [1, 0, ..., 0]$$

(that is, $\lambda[0] = 1$, and for $1 < i \leq n$ $\lambda[i] = 0$).

For each $\sigma \in \Sigma$ we define

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

For $(i, i_1, i_2, \ldots, i_j) \in \{1, 2 \ldots, |\mathcal{V}|\}^{|j|+1}$, 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:

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

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

Proposition B.4. *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 2.1. 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 $i \leq |\mathcal{V}|$, $v[i] = \mathcal{W}_{N_i}(t)$; 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 $i > |\mathcal{V}|$ $\mu_?{}^i_{i_1,\ldots,i_j} = 0$ 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 $i \leq |\mathcal{V}|$. By definition of a multi-linear map we have that:

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

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

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

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

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

For each i, by our 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 $j_i \leq |\mathcal{V}|$, we have that $v_i[j_i] = \mathcal{W}_{N_{j_i}}(t_i)$. Therefore we have:

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

So by lemma B.1 we have that $v[i] = \mathcal{W}_{N_i}(t)$ as required.

Finally, since $S = N_0$ and since by our claim, for each $i \leq |\mathcal{V}|$, $v_i = v[i] = \mathcal{W}_{N_i}(t)$, 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. We recall that 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 B.5. *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 $N_1 \to \alpha$ and $N_2 \to \alpha$. Then the tree $?(\alpha)$ is structurally ambiguous since its root can be tagged by both N_1 and N_2.

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. $N \to \alpha$. Hence, there is a single tagging for the root of t as well. Thus $\mathcal{G}$ is structurally unambiguous. ☐

We are finally ready to prove **Proposition 3.5.**:

A CMTA can represent an SUWCFG.

Proof. By Proposition B.4 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 B.5 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. ☐

B.3 Proofs of Section 5

We first provide a detailed description of the algorithms that were not given in the body of the paper due to lack of space, and then we delve into the proof of correctness of the learning algorithm.

Algorithm 1 *Complete*(T, C, H, B, S).

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

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     $T \leftarrow T \cup \{t\}$
```

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 B.6. *Let B be a set of 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 B.7. *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:

$$\mu(c_1[\![t_1']\!]) = \alpha \cdot \mu(c_1[\![t_2']\!])$$

So:

$$\mu(c[\![t_1]\!]) = \mu(c_1[\![t_1']\!]) = \alpha \cdot \mu(c_1[\![t_2']\!]) = \alpha \cdot \mu(c[\![t_2]\!])$$

As required. ☐

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 $b \in B$, s.t. $t \ltimes_H^\alpha b$. 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 s.t. $t \ltimes_H^\alpha b_i$, 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 B.8. *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 $t_1, t_2, ..., t_p$ s.t. $t_i \neq [t_i]$. 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 s.t. $t_i \neq [t_i]$. 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 s.t. $t_i \neq [t_i]$, so from the induction hypothesis we have:

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

So 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 $H[y] \neq 0$ then the ratio between the tree coeficient and the product of its children coefficeints is the same.

Lemma B.9. *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 B.8 we have that $H[t] = \prod_{i=1}^{p} \alpha[t_i] \cdot H[t']$. Similarly we have that $H[s] = \prod_{i=1}^{p} \alpha[s_i] \cdot H[t']$. 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 B.10. *Let $t \in T$. If $H[t] \neq 0$ then $\mu(t) = \alpha[t] \cdot e_{\iota[t]}$. 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. 3 , we set $\sigma^{\iota[t]}$ to be $\alpha[t]$, and for every $j \neq \iota[t]$ we set σ^j to be 0, so $\mu(t) = \mu_\sigma = \alpha[t] \cdot e_{\iota[t]}$ 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 $\mu(t_j) = \alpha[t_j] \cdot e_{\iota[t_j]}$. So:

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

Therefore we have:

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

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

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

By Alg. 3 , and Lemma B.9 we have that

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

hence we obtain:

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

Thus $\mu(t) = \alpha[t] \cdot e_{\iota[t]}$ as required.

If $H[t] = 0$ then $\sigma^i_{\iota[t_1],\iota[t_2],\dots,\iota[t_p]} = 0$ 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 B.11. *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.

Base case: b_i is a leaf, so $b = \sigma$ for $\sigma \in \Sigma_0$. By Alg. 3 we set σ_i to be 1 and σ_j to be 0 for every $j \neq i$, so $\mu(b_i) = e_i$.

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 $\mu(b_i)[j]$

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

By 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:

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

And for $j_1, j_2, ..., j_p = i_1, i_2, ..., i_p$ we obtain:

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

So we have:

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

By Alg. 3 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 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 B.12. *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 B.11 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 B.11 we obtain that for every $1 \leq j \leq p$:

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

By Alg. 3 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 $i \neq j$, so $\mu(t) = \alpha \cdot e_i$ as required. ☐

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

Lemma B.13. *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 B.12.

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 B.11 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 $t \ltimes^\beta_H t'$. Let b be the element in the base s.t. $t' \ltimes^\alpha_H b_i$. From Lemma B.12 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 $t \ltimes^{\alpha \cdot \beta}_H b_i$. 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 B.14. *For every $t \in T$ and for every $c \in C$ we have that $\mathcal{A}(c[\![t]\!]) = H[t][c]$*

Proof. Let $t \in T$, s.t. $t \ltimes^\alpha_H b_i$ for some $b_i \in B$. The proof is by induction on the depth of $\diamond$ in c.

Base case: The depth of c is 1, so $c = \diamond$, and by lemma B.13 we have that $\mu(c[\![t]\!]) = \mu(t) = \alpha \cdot e_i$. Therefore $\mathcal{A}(t) = \alpha \cdot e_i \cdot \lambda$. By Alg. 3 we have that $\lambda[i] = H[b_i][\diamond]$. So $\mathcal{A}(t) = \alpha \cdot H[b_i][\diamond] = H[t][\diamond]$ as required.

Induction step: Let c be a context s.t. the depth of $\diamond$ is $h+1$. So $c = c'[\![\sigma(t_1, t_2, ..., t_{i-1}, \diamond, t_i, ..., t_p)]\!]$ 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:

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

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

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

Since each t_{i_j} is in T, from Proposition B.13 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 B.7 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:

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

So:

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

As required. ☐

Proposition B.15. *In every iteration of Alg. 1 , the set B maintained by the algorithm is contained in some correct solution.*

Proof. We show by induction that in each iteration, $B \subseteq B^*$ for some co-linear base B^*.

The base case is trivial since $B = \emptyset$. Hence, clearly $B \subseteq B^*$ for some co-linear base B^*.

For the induction step, let B be the set created by the algorithm in the previous iteration. Let t be the tree picked by the algorithm to be added to the basis in the current iteration. If $t \in B^*$ then $B \cup \{t\} \subseteq B^*$. Otherwise, $t \notin B^*$. Let N be the tagging for the root of t. Since t was picked, we have that $H[t][C]$ is co-linearly independent from $B[C]$. By Lemma B.6 $H[t]$ is co-linearly independent from B, and so by Prop. 3.3 there is no tree in B whose root is tagged by N. Since B^* is a co-linear base, there must be a tree $t' \in B^* \setminus B$ whose root is tagged by N, by Prop. 3.3 t and t' are co-linear. Let $B^{*'} = B^* \setminus \{t'\} \cup \{t\}$. $B^{*'}$ is still a co-linear base, because t and t' are co-linear. So, $B \cup \{t\} \subseteq B^{*'}$ and $B^{*'}$ is a co-linear base. ☐

Proposition B.16. *Let (T, C, H, B) be a closed and consistent observation table. Let t be a counterexample, and let $(T', C', H', B') = \mathit{Complete}(T, C, H, B, \mathit{Pref}(t))$. Then $|B'| > |B|$.*

Proof. Since t was given as a counterexample, the previously extracted CMTA $\mathcal{A} = \mathit{CMTAB}(T, C, H, B)$ gave a wrong answer for it. By the method complete, $t \in T'$, hence by Lemma B.13 we have that $\mathcal{A}' = \mathit{CMTAB}(T', C', H', B')$ agrees with t. Since we only add elements to the base, we have that $B \subseteq B'$. Now, clearly $B \neq B'$ since otherwise (following Alg. 3) we would obtain $\mathcal{A} = \mathcal{A}'$. Hence $B \subsetneq B'$ implying $|B'| > |B|$. ☐

We are now ready to prove the main theorem, **Theorem 4.1**. which states that

> *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 $n \cdot (n + m \cdot n + |\Sigma| \cdot (n + m \cdot n)^p)$ MQ and at most n EQ.*

Proof. By Prop B.16 we have that the rank of the finite Hankel Matrix $H[C]$ increases by at least one after each equivalence query. Since the rank of the infinite Hankel Matrix H is n, it follows that the learner makes at most n equivalence queries.

For the set of contexts, the algorithm starts with $C = \{\diamond\}$. The algorithm adds new contexts following failed consistency checks. Each added context thus separates two trees in T and increases the rank of $H[C]$ by 1. Therefore, $|C| \leq n$.

We add an element to set of trees T in two cases: if it is in $\Sigma(T)$, and it is co-linearly independent from T, or it is a prefix of a counterexample given to us by the teacher.

The first case can occur at most n times, since each time we add an element from $\Sigma(T)$ to T we increase the rank of the table by 1.

Since the number of EQ made by the learner is at most n, the learner receives at most n counterexamples from the teacher. Let m be the size of the largest counterexample given by the teacher. We add all its prefixes to T, so each counterexample adds at most m elements to T. So we have that by the end of the algorithm, $|T| \leq n + m \cdot n$. Let p be the maximal rank of symbol in Σ, then $|\Sigma(T)| \leq |\Sigma| \cdot |T|^p \leq |\Sigma| \cdot (n + m \cdot n)^p$. Hence the number of rows in the table is at most:

$$n + m \cdot n + |\Sigma| \cdot (n + m \cdot n)^p$$

Since the number of columns in the table is at most n, the algorithm makes at most:

$$n \cdot (n + m \cdot n + |\Sigma| \cdot (n + m \cdot n)^p)$$

membership queries. For a Chomsky Normal Form grammar, we have that $p = 2$ and therefore

$$n \cdot (n + m \cdot n + |\Sigma| \cdot (n + m \cdot n)^2)$$

□

C Demonstration - Supplementary Material

We implemented our algorithm within a tool called PCFGLearner. The code and the data-sets used in this section, as well as a manual explaining how to employ our tool, are available with the submitted supplementary material. In this section we apply PCFGLearner to the learning of two PCFGs over genomic data.

The section starts with a description of how the string dataset was generated (Section C.1). Then, the conversion of the strings to parse trees is described in Section C.2. Next, the implementation details of the oracle are given: Section C.3 describes how the SMQ was implemented, and Section C.4 describes how the SEQ was implemented. A biological interpretation of the first grammar is given in Section C.5, and the description of the second grammar is given in Section C.6.

C.1 Methods and datasets

Genes in our experiment are represented by their membership in Clusters of Orthologous Genes (COGs) [41]. $1,487$ fully sequenced prokaryotic strains with COG ID annotations were downloaded from GenBank (NCBI; ver 10/2012). The gene clusters were generated using the tool CSBFinder-S [40].

CSBFinder-S was applied to all the chromosomal genomes in the dataset after removing their plasmids, using parameters $q = 1$ (a colinear gene cluster is required to appear in at least one genome) and $k = 0$ (no insertions are allowed in a colinear gene cluster), resulting in 595,708 colinear gene clusters. Next, ignoring strand and gene order information, colinear gene clusters that contain the exact same COGs were united to form the generalized set of gene clusters.

To generate the trees for the first example, a subset of strings over the COGs AcrA(COG0845), AcrB(COG0841), TolC(COG1538) and AcrR(COG1309), where each of these COGs appeared at least once, was considered. This yielded 415 instances over 11 distinct strings. After the removal of strand information, 7 distinct strings remained, from which the trees were constructed.

To generate the trees for the second example, a subset of strings over the COGs FimA(COG3539), FimC(COG3121), FimD(COG3188), and CitB(COG2197) was considered. This yielded 1899 instances over 28 distinct strings. After the removal of strand information, 25 distinct strings remained, from which the trees were constructed.

In both examples, the constructed trees were annotated with a probability, according to the frequency of the corresponding strings in the dataset. For the sake of simplicity and efficiency, we used binary trees in the given examples.

C.2 Tree construction

Since the gene-cluster-data is available as strings, and our learning algorithm accepts trees, we propose the following approach to construct the expected parse trees from the strings by modelling two of the main events in operon evolution: in-tandem gene duplications [32, 27] and progressive merging of sub-operons [16, 17].

To model the former event, if two homologous genes (represented by the same COG) are found next to each other across many genomes, we assume that this is a result of a duplication event, and we place these two genes in the same sub-tree. To model the latter event, we assume that if a sub-string is significantly over-represented in our data-set (indicating that it is conserved across many bacteria), then it is likely that this sub-string encodes a conserved functional unit, and that it should be confined to a distinguished sub-tree.

The following scoring measure is used by the proposed parser to represent how likely it is that a considered parse tree structurally interprets a string. For a given string s, let $s[i]$ be the i'th character in the string, and let $s[i:j]$ be the sub-string from the i'th character to the j'th character.

Given a weight function $w : \Sigma^* \to \mathbb{R}^+$, for a given string $s = \sigma_1\sigma_2...\sigma_n$, let $w_{\textit{tree}}(s)$ denote the following function:

$$w_{\textit{tree}}(s) = \begin{cases} w(s) & \text{if } |s| \leq 2 \\ w(s) + \max\limits_{1\leq i\leq n}(w_{\textit{tree}}(s[1:i]) + \\ \qquad\qquad w_{\textit{tree}}(s[i+1:n])) & \text{otherwise} \end{cases}$$

The optimal score $w_{\textit{tree}}(s)$, as well as a corresponding optimal tree, can now be computed in $O(n^3)$ using a simple dynamic-programming algorithm, which is a variation of the CKY algorithm [25, 45].

To force consecutive runs of a gene to appear together, the string s is pre-processed before applying the algorithm. For each $\sigma \in \Sigma$, consecutive runs of σ^k are merged to a new symbol σ_k. The new symbol σ_k doesn't appear in s, and maintains that $w(u\sigma v) = w(u\sigma_k v)$ for every $u, v \in \Sigma^*$.

After computing the optimal tree for the pre-processed string, every leaf whose label is σ_k is replaced by a right-chain (defined in the next section) containing exactly k leaves tagged with σ.

The tree construction algorithm is used both during the conversion of sequences to trees, as well as during the execution of an equivalence query of the MDR experiment (see section C.4).

The scoring $w : \Sigma^* \to \mathbb{R}$ is implemented according to [39], and we refer the reader to that paper for a detailed elaboration on how it is computed. In a nutshell, for a given string s, and a given dataset G, let q_s denote the number of genomes from G in which s occurs. Assuming a uniform random order of genes in a genome, the ranking score computation first evaluates how likely it is for s to occur in at least q_s genomes from G by mere chance, and then reports the negative logarithm (base e) of the computed score, so that the higher the ranking score of s, the less likely to be formed merely by chance. The parameters considered in the ranking score computation for s include the number of input genomes in G, the average length of a genome, the length of s, the number of genomes from G in which s occurs, and the frequencies of each gene from s in the data.

C.3 Membership query implementation

In this section we describe the edit-distance functions that were used in the implementation of the oracle, for SMQ computation. We assume that all trees here are binary, and that their labels are in $\mathbb{N}$. We use the notation $l(t)$ for the label of the tree t.

For a tree t, if t is not a leaf, let t_1 and t_2 be its left and right children, respectively.

Two trees t, s are *incompatible* if t and s are both leaves, and $l(t) \neq l(s)$, or if only one of t, s is a leaf, while the other is an internal node.

Swap-Event-Counting Edit Distance

The swap-event-counting edit distance measure $w(t, s)$ is computed using the following recursive formula:

$$w(t,s) = \begin{cases} 0 & \text{if } t \text{ and } s \text{ are leaves and } l(t) = l(s) \\ \infty & \text{if } t \text{ and } s \text{ are incompatible} \\ \min(w(t_1,s_1) + w(t_2,s_2), \\ \qquad w(t_1,s_2) + w(t_2,s_1) + 1) & \text{otherwise} \end{cases}$$

Duplication-Event-Counting Edit Distance

A tree t is a right-chain if t is a leaf, or if the left child of t is a leaf, the right child of t is a right-chain and the labels on all the leaves of t are equal. (Symmetrically, for a left-chain.)

For a right or left chain t, let $\textit{size}(t)$ be the number of leaves of t, and let $\textit{label}(t)$ be the tagging of the leaves of t. (Note that in a chain, all the leaves have the same tagging, so *label* is well-defined.)

Two trees t and s, are *right-chain-homologous*, if both are right-chains with the same tagging. For two chain-homologous trees, we say that $|\textit{size}(t) - \textit{size}(s)|$ is the *copy-number difference* between them. For two trees t and s, we say that t and s are *right-homologous* if either t and s are right-chain-homologous, or if $t = (t_1, t_2)$, $s = (s_1, s_2)$ and both t_1 and s_1, and t_2 and s_2 are right-homologous. In that case, the copy-number difference between t and s is the sum of the copy-number differences between t_1 and s_1, and t_2 and s_2. (Symmetrically, for left-homologous trees.)

For two given trees t, s, let $w(t, s)$ denote the duplication-event-counting measure. The value of $w(t, s)$ is the copy-number difference between t and s if they are right-homologous, or ∞ if they aren't. The choice of right-homologous instead of left-homologous is arbitrary.

$$w(t,s) = \begin{cases} |\mathit{size}(t) - \mathit{size}(s)| & \text{if } s \text{ and } t \text{ are right-chain-homologous} \\ \infty & \text{if } s \text{ and } t \text{ are incompatible and aren't right-chain-homologous} \\ w(t_1, s_1) + w(t_2, s_2) & \text{otherwise} \end{cases}$$

C.4 Equivalence query implementation

PCFGLearner supports three types of equivalence queries: Exhaustive Search, Random Sampling and Duplications Generator. In all these implementations, the oracle generates a set of trees T. Then, the oracle looks for a tree $t \in T$ s.t. $\mathcal{A}(t) \neq \textsc{smq}(t)$. If such a tree is found, it is returned as a counterexample, otherwise, a positive answer to the equivalence query is returned. The three implementations differ in the way they generate the trees.

In the Exhaustive Search equivalence query, given an alphabet Σ, and a maximum length l, all strings up to length l are generated. For each string s, the optimal tree for it t is constructed, as described in Section C.2.

In the Random Sampling equivalence query, instead of generating all strings up-to length l, two parameters are given, r and s. For each $\textsc{seq}$, r strings are independently sampled from a uniform distribution of all strings of length at most s. To each string the oracle then constructs the optimal tree, as in the Exhaustive Search equivalence query.

In the Duplications Generator, given a set of trees R, and a parameter d. The oracle generates for each $t \in R$ all the trees t' that can be obtained from t by duplicating each of its leaves at most d times.

C.5 Biological interpretation of the Multi Drug Resistance Efflux Pump grammar

In this section, we propose some biological interpretation of the grammar learned in the first example given in Section 5 of the main paper, associating the highly probable rules of this grammar with explanatory evolutionary events, or with the functional reasoning underlying the strong conservation of the corresponding gene orders.

AcrAB–TolC is a multidrug efflux pump that is widely distributed among gram-negative bacteria and extrudes diverse substrates from the cell, conferring resistance to a broad spectrum of antibiotics [26]. This pump belongs to the resistance-nodulation-cell division (RND) family. In Gram-negative bacteria, RND pumps exist in a tripartite form, composed of an outer-membrane protein (TolC in our example), an inner membrane protein (AcrB in our example), and a periplasmic membrane fusion protein (AcrA in our example) that connects the other two proteins. The genes of the RND pump are often flanked with genes that code for local regulatory proteins, such as, in our example, the response regulator AcrR.

The pair of genes encoding the AcrB and AcrA proteins usually appear as an adjacent pair in our data, with AcrA preceding AcrB in the direction of transcription. The conserved order AcrA-AcrB could be explained by the order of assembly of the products of these genes into the AcrAB complex [37], and by stochiometry [28]. The grammar learned for the AcrABR-TolC gene cluster (Figure 4 in the main paper) indeed reflects this structural phenomenon, as AB, which is derived from the non-terminal N_2, is 4 times more likely to be derived than BA, which is derived from the non-terminal N_4.

The outer-membrane protein TolC consistently appears, in our training dataset, adjacently to the AcrA-AcrB gene pair. However it forms a separate sub-tree from AcrAB in the highly probable trees generated by the learned grammar. This is due to the fact that, while the ordered pair AcrA-AcrB is very highly conserved in our data ($w(\text{AcrA AcrB}) = 31816.63$), TolC typically joins this triplet either upstream ($w(\text{TolC AcrA}) = 9494.15$) to it or downstream ($w(\text{AcrB TolC}) = 15783.51$) to it.

Indeed, several studies indicate that the assembly and docking of the AcrAB-TolC efflux pump occur as a multi-step process, starting with the assembly of the AcrA-AcrB complex (see Figure 5), and only then activating the pump by the docking of TolC to the AcrAB complex [18, 37].

This assembly process is explained by additional studies, speculating that the TolC channel and inner membrane efflux AcrA-AcrB components may form a transient complex with the outer membrane channel TolC during efflux, due to the fact that TolC should be ready to use for not only AcrAB but also other efflux, secretion and transport systems in which it participates [22]. Indeed, in our general dataset, TolC participates in additional gene clusters encoding other export systems. This, along with the fact that in our string dataset this gene appears both upstream and downstream to the AcrAB pair, support the hypothesis that during the evolution of this pump across a wide-range of gram-negative bacteria, TolC was merged more than once with AcrAB (as well as with other export systems), in distinct evolutionary events.

The gene cluster includes another gene, AcrR, that codes for a protein regulating the expression of the tripartite pump. This gene appears in a separate subtree from the AcrAB-TolC subtree, furthermore it is more likely to appear upstream to the AcrAB-TolC sub-tree in the highly probable trees. This could be explained by AcrR's functional annotation as a response regulator, whose role is to respond to the presence of a substrate, and consequently to enhance the expression of the RND tripartite efflux pump genes [3].

The fact that AcrR forms a separate subtree from the tripartite pump further exemplifies the role of merge events in gene cluster evolution. Indeed, other gene clusters in our dataset that include the tripartite pump are flanked by alternative response regulators, indicating that AcrAB-TolC homologs have merged with various regulators throughout the evoulution of gram-negative bacteria, yielding response to a variety of distinct drugs [43].

Thus, the grammar learned by PCFGLearner exemplifies

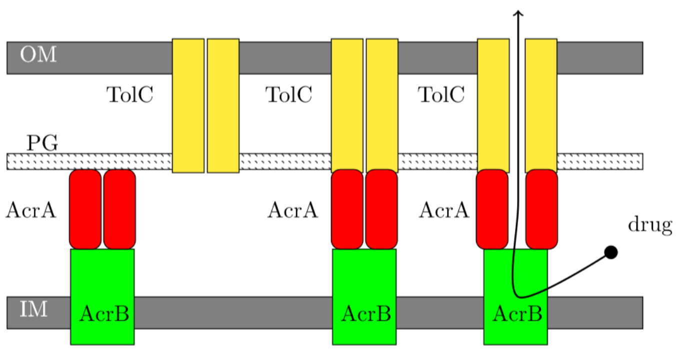


Figure 5: In vivo assembly and functioning mechanism for multidrug efflux pump AcrAB-TolC according to [18, 37]. First AcrB associates with AcrA, to form the bipartite complex AcrAB. Next, AcrA changes its conformation to recruit TolC. Once TolC binds with the AcrAB bipartite complex, the fully assembled tripartite pump remains in the resting state. When AcrB encounters a drug molecule, the pump adopts conformation accompanied with a contraction along the long axis and the substrate is expelled through the channel and out of the cell. This figure was prepared according to Figure 4 in [37]. OM, outer membrane; PG, peptidoglycan; IM, inner membrane.

how our proposed approach can be harnessed to study biological systems that are conserved as gene clusters, and to explore their function and their evolution.

C.6 A grammar exemplifying duplication events

In this example, we applied PCFGLearner to the FimACD gene cluster data-set, using the duplication-event-counting edit-distance metric with decay factor $q = 0.2$, and the Duplications Generator equivalence query with parameter $d = 2$.

In the rest of this section we give a few selected rules from the resulting grammar, mainly to demonstrate that this grammar can generate an infinite number of strings, with exponentially decaying probabilities. The entire grammar is available in section C.7. The exemplified grammar allows each symbol in Σ to duplicate with a probability of 0.2 for each duplication. For example, for FimA we have:

$$\begin{aligned} N_7 &\to N_1 N_8 [0.8] \\ N_7 &\to N_1 N_7 [0.2] \\ N_8 &\to N_1 N_1 [1.0] \\ N_1 &\to FimA [1.0] \end{aligned}$$

We refer the reader to Figures 6 and 7, for an example of a parse tree from the grammar, and an illustration of how the right-chain with n FimA-tagged leaves can be obtained with probability of $0.8 \cdot (0.2)^{n-3}$ for $n \geq 3$ from N_7. The term $n \geq 3$ is due to the fact that N_7 already represents a sub-tree learned from the data-set, with three consecutive copies of FimA.

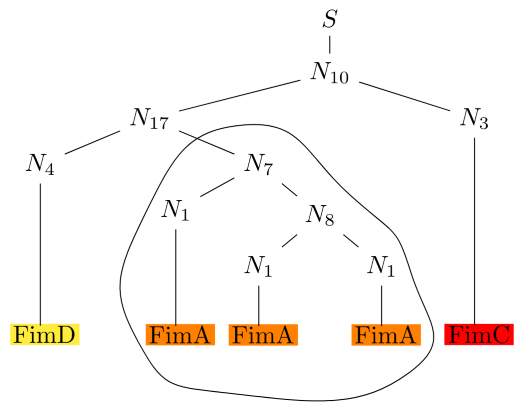


Figure 6: A parse tree from the grammar learned from the FimACD gene cluster dataset. The marked sub-tree can grow indefinitely (see Figure 7), using the production $N7 \to N1N7$ with a probability of 0.2.

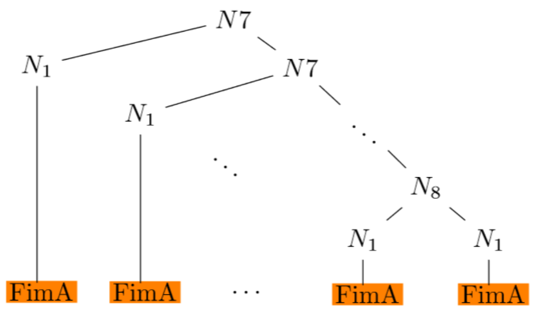


Figure 7: A right chain formed from $N7$, generating n FimA's with a probability of $0.8 \cdot (0.2)^{n-3}$ for $n \geq 3$

C.7 FimACD Grammar

Production	Probability
$S \to N2$	0.103
$S \to N9$	0.335
$S \to N10$	0.050
$S \to N11$	0.062
$S \to N12$	0.028
$S \to N18$	0.036
$S \to N19$	0.056
$S \to N20$	0.032
$S \to N21$	0.030
$S \to N22$	0.053
$S \to N23$	0.039
$S \to N24$	0.026
$S \to N25$	0.037
$S \to N26$	0.008
$S \to N27$	0.037
$S \to N28$	0.038
$S \to N29$	0.030
$N1 \to$ FimA	1.000
$N2 \to N3\,N4$	0.213
$N2 \to N3\,N17$	0.587
$N2 \to N13\,N4$	0.053
$N2 \to N13\,N17$	0.147
$N3 \to$ FimC	1.000
$N4 \to$ FimD	1.000
$N5 \to$ CitB	1.000
$N6 \to N1\,N3$	0.014
$N6 \to N1\,N13$	0.004
$N6 \to N7\,N3$	0.446
$N6 \to N7\,N13$	0.111
$N6 \to N8\,N3$	0.071
$N6 \to N8\,N13$	0.018
$N6 \to N14\,N3$	0.139
$N6 \to N14\,N13$	0.035
$N6 \to N15\,N3$	0.018
$N6 \to N15\,N13$	0.004
$N6 \to N16\,N3$	0.111
$N6 \to N16\,N13$	0.028
$N7 \to N1\,N7$	0.200
$N7 \to N1\,N8$	0.800
$N8 \to N1\,N1$	1.000
$N9 \to N1\,N10$	0.115
$N9 \to N1\,N26$	0.023
$N9 \to N1\,N27$	0.088
$N9 \to N1\,N28$	0.100
$N9 \to N1\,N29$	0.075
$N9 \to N2\,N6$	0.092
$N9 \to N7\,N10$	0.008
$N9 \to N7\,N26$	0.001
$N9 \to N7\,N27$	0.004
$N9 \to N7\,N28$	0.005
$N9 \to N7\,N29$	0.004
$N9 \to N8\,N10$	0.030
$N9 \to N8\,N26$	0.005
$N9 \to N8\,N27$	0.018
$N9 \to N8\,N28$	0.020

$N9 \to N8\,N29$	0.015
$N9 \to N11\,N6$	0.085
$N9 \to N12\,N6$	0.021
$N9 \to N15\,N2$	0.016
$N9 \to N15\,N10$	0.008
$N9 \to N15\,N19$	0.016
$N9 \to N15\,N20$	0.004
$N9 \to N15\,N25$	0.018
$N9 \to N16\,N2$	0.004
$N9 \to N16\,N10$	0.002
$N9 \to N16\,N19$	0.004
$N9 \to N16\,N20$	0.001
$N9 \to N16\,N25$	0.004
$N9 \to N18\,N5$	0.053
$N9 \to N18\,N30$	0.013
$N9 \to N19\,N5$	0.009
$N9 \to N19\,N30$	0.002
$N9 \to N20\,N5$	0.014
$N9 \to N20\,N30$	0.003
$N9 \to N21\,N5$	0.013
$N9 \to N21\,N30$	0.003
$N9 \to N22\,N5$	0.009
$N9 \to N22\,N30$	0.002
$N9 \to N23\,N5$	0.017
$N9 \to N23\,N30$	0.004
$N9 \to N24\,N5$	0.002
$N9 \to N24\,N30$	0.001
$N9 \to N25\,N5$	0.055
$N9 \to N25\,N30$	0.014
$N10 \to N4\,N3$	0.213
$N10 \to N4\,N13$	0.053
$N10 \to N17\,N3$	0.587
$N10 \to N17\,N13$	0.147
$N11 \to N1\,N2$	1.000
$N12 \to N7\,N2$	0.200
$N12 \to N8\,N2$	0.800
$N13 \to N3\,N3$	0.800
$N13 \to N3\,N13$	0.200
$N14 \to N1\,N15$	0.640
$N14 \to N1\,N16$	0.160
$N14 \to N7\,N15$	0.032
$N14 \to N7\,N16$	0.008
$N14 \to N8\,N15$	0.128
$N14 \to N8\,N16$	0.032
$N15 \to N7\,N1$	0.200
$N15 \to N8\,N1$	0.800
$N16 \to N7\,N7$	0.040
$N16 \to N7\,N8$	0.160
$N16 \to N8\,N7$	0.160
$N16 \to N8\,N8$	0.640
$N17 \to N4\,N1$	0.003
$N17 \to N4\,N4$	0.073
$N17 \to N4\,N7$	0.108
$N17 \to N4\,N8$	0.017
$N17 \to N4\,N14$	0.034
$N17 \to N4\,N15$	0.004
$N17 \to N4\,N16$	0.027
$N17 \to N4\,N17$	0.200
$N17 \to N17\,N1$	0.010
$N17 \to N17\,N7$	0.297
$N17 \to N17\,N8$	0.048
$N17 \to N17\,N14$	0.093
$N17 \to N17\,N15$	0.012
$N17 \to N17\,N16$	0.074
$N18 \to N1\,N25$	1.000
$N19 \to N2\,N1$	1.000
$N20 \to N2\,N7$	0.800
$N20 \to N2\,N16$	0.200
$N21 \to N7\,N25$	0.200
$N21 \to N8\,N25$	0.800
$N22 \to N1\,N19$	1.000
$N23 \to N1\,N20$	0.800
$N23 \to N7\,N20$	0.040
$N23 \to N8\,N20$	0.160
$N24 \to N7\,N19$	0.200
$N24 \to N8\,N19$	0.800
$N25 \to N2\,N8$	0.800
$N25 \to N2\,N15$	0.200
$N26 \to N2\,N14$	1.000
$N27 \to N10\,N1$	1.000
$N28 \to N10\,N7$	0.640
$N28 \to N10\,N14$	0.200
$N28 \to N10\,N16$	0.160
$N29 \to N10\,N8$	0.800
$N29 \to N10\,N15$	0.200
$N30 \to N5\,N5$	0.800
$N30 \to N5\,N30$	0.200