On the Regularity and Learnability of Ordered DAG Languages

Henrik Björklund, Johanna Björklund, and Petter Ericson

Dept. Computing Science, Umeå University

Abstract. Order-Preserving DAG Grammars (OPDGs) is a subclass of Hyper-Edge Replacement Grammars that can be parsed in polynomial time. Their associated class of languages is known as Ordered DAG Languages, and the graphs they generate are characterised by being acyclic, rooted, and having a natural order on their nodes. OPDGs are useful in natural-language processing to model abstract meaning representations. We state and prove a Myhill-Nerode theorem for ordered DAG languages, and translate it into a MAT-learning algorithm for the same class. The algorithm infers a minimal OPDG G for the target language in time polynomial in G and the samples provided by the MAT oracle.

1 Introduction

Graphs are one of the fundamental data structures of computer science, and appear in every conceivable application field. We see them as atomic structures in physics, as migration patterns in biology, and as interaction networks in sociology. For computers to process potentially infinite sets of graphs, i.e., graph languages, these must be represented in a finite form akin to grammars or automata. However, the very expressiveness of graph languages often causes problems, and many of the early formalisms have NP-hard membership problems; see, e.g., [15] and [8, Theorem 2.7.1].

Motivated by applications in natural language processing (NLP) that require more light-weight forms of representation, there is an on-going search for grammars that allow polynomial time parsing. A recent addition to this effort was the introduction of order-preserving DAG grammars (OPDGs) [3]. This is a restricted type of hyper-edge replacement grammars [8] that generate languages of directed acyclic graphs in which the nodes are inherently ordered. The authors provide a parsing algorithm that exploits this order, thereby limiting nondeterminism and placing the membership problem for OPDGs in $O\big(n^2 + nm\big)$, where m and n are the sizes of the grammar and the input graph, respectively. This is to be compared with the unrestricted case, in which parsing is NP-complete.

The introduction of OPDGs is a response to the recent application [5] of Hyperedge Replacent Grammars (HRGs) to abstract meaning representations (AMRs) [2]. An AMR is a directed acyclic graph that describes the semantics of a natural language sentence. Although restricted, OPDGs retain enough expressive power to capture AMRs.

In this paper, we continue to explore the OPDGs mathematical properties. We provide an algebraic representation of their domain, and a Myhill-Nerode theorem for the ordered DAG languages. We show that every ordered DAG language L is generated by a minimal unambiguous OPDG G_L, and that this grammar is unique up to renaming of nonterminals. In this context, 'unambiguous' means that every graph is generated by at most one nonterminal. This is similar the behaviour of deterministic automata, in particular that of bottom-up deterministic tree automata which take each input tree to at most one state.

One way of understanding the complexity of the class of ordered DAG languages, is to ask what kind of information is needed to infer its members. MAT learning [1], where MAT is short for minimal adequate teacher, is one of the most popular and well-studied learning paradigms. In this setting, we have access to an oracle (the teacher) that can answer *membership queries* and *equivalence queries*. In a membership query, we present the teacher with a graph g and are told whether g is in the target language L. In an *equivalence* query, we give the teacher an OPDG H and receive in return an element in in the symmetric difference of $L(H)$ and L. This element is called a *counterexample*. If L has been successfully inferred and no counterexample exists, then the teacher instead returns the special token $\perp$.

MAT learning algorithms have been presented for a range of language classes and representational devices [1,16,17,9,11,4,13]. There have also been some results on MAT learning for graph languages. Okada et al. present an algorithm for learning unions of linear graph patterns from queries [14]. These patterns are designed to model structured data (HTML/XML). The linearity of the patterns means that no variable can appear more than once. Hara and Shoudai do MAT learning for context-deterministic regular formal graph systems [10]. Intuitively, the context determinism means that a context uniquely determines a nonterminal, and only graphs derived from this nonterminal may be inserted into the context. Both restrictions are interesting, but neither is compatible with our intended applications.

2 Preliminaries

Sets, sequences and numbers. The set of non-negative integers is denoted by $\mathbb{N}$. For $n \in \mathbb{N}$, $[n]$ abbreviates $\{1, \ldots, n\}$, and $\langle n \rangle$ the sequence $1 \cdots n$. In particular, $[0] = \emptyset$ and $\langle 0 \rangle = \lambda$. We also allow the use of sets as predicates: Given a set S and an element s, $S(s)$ is *true* if $s \in S$, and *false* otherwise. When $\equiv$ is an equivalence relation on S, $(S/\equiv)$ denotes the partitioning of S into equivalence classes induced by $\equiv$. The *index* of $\equiv$ is $|(S/\equiv)|$. For $s \in S$, $[s]_\equiv$, or simply $[s]$, is the equivalence class of s with respect to $\equiv$.

Let S° be the set of non-repeating sequences of elements of S. We refer to the ith member of a sequence s as s_i. When there is no risk for confusion, we use sequences directly in set operations, as the set of their members. Given a partial order $\preceq$ on S, the sequence $s_1 \cdots s_k \in S^\circ$ respects $\preceq$ if $s_i \preceq s_j$ implies $i \preceq j$.

A *ranked alphabet* is a pair $(\Sigma, rank)$ consisting of a finite set Σ of symbols and a *ranking function* $rank : \Sigma \mapsto \mathbb{N}$ which assigns a rank $rank(a)$ to every symbol $a \in \Sigma$. The pair $(\Sigma, rank)$ is typically identified with Σ, and the second component is kept implicit.

Graphs. Let Σ be a ranked alphabet. A (directed edge-labelled) *hypergraph* over Σ is a tuple $g = (V, E, src, tar, lab)$ consisting of

- finite sets V and E of *nodes* and *edges*, respectively,
- *source* and *target mappings* $src\colon E \mapsto V$ and $tar\colon E \mapsto V^\circ$ assigning to each edge e its source $src(e)$ and its sequence $tar(e)$ of targets, and
- a *labelling* $lab\colon E \mapsto \Sigma$ such that $rank(lab(e)) = |tar(e)|$ for every $e \in E$.

Since we are only concerned with hypergraphs, we simply call them graphs.

A *path* in g is a finite and possibly empty sequence $p = e_1, e_2, \ldots, e_k$ of edges such that for each $i \in [k-1]$ the source of e_{i+1} is a target of e_i. The *length* of p is k, and p is a *cycle* if $src(e_1)$ appears in $tar(e_k)$. If g does not contain any cycle then it is a *directed acyclic graph* (DAG). The *height* of a DAG G is the maximum length of any path in g. A node v is a *descendant* of a node u if $u = v$ or there is a nonempty path $e_1, \ldots, e_k$ in g such that $u = src(e_1)$ and $v \in tar(e_k)$. An edge e' is a *descendant edge* of an edge e if there is a path $e_1, \ldots, e_k$ in g such that $e_1 = e$ and $e_k = e'$.

The *in-degree* and *out-degree* of a node $u \in V$ is $|\{e \in E \mid u \in tar(e)\}|$ and $|\{e \in E \mid u = src(e)\}|$, respectively. A node with in-degree 0 is a *root* and a node with out-degree 0 is a *leaf*. For a single-rooted graph g, we write $root(g)$ for the unique root node.

For a node u of a DAG $g = (V, E, src, tar, lab)$, the *sub-DAG rooted at* u is the DAG $g{\downarrow}u$ induced by the descendants of u. Thus $g{\downarrow}u = (U, E', src', tar', lab')$ where U is the set of all descendants of u, $E' = \{e \in E \mid src(e) \in U\}$, and src', tar', and lab' are the restrictions of src, tar and lab to E'. A leaf v of $g{\downarrow}u$ is *reentrant* if there exists an edge $e \in E \setminus E'$ such that v occurs in $tar(e)$. Similarly, for an edge e we write $g{\downarrow}e$ for the subgraph induced by $src(e)$, $tar(e)$, and all descendants of nodes in $tar(e)$. This is distinct from $subgg src(e)$ iff $src e$ has out-degree greater than 1.

Marked graphs. Although graphs, as defined above, are the objects we are ultimately interested in, we will mostly discuss marked graphs. When combining smaller graphs into larger ones, whether with a grammar or algebraic operations, the markings are used to know which nodes to merge with which.

A *marked DAG* is a tuple $g = (V, E, src, tar, lab, X)$ where (V, E, src, tar, lab) is a DAG and $X \in V^\circ$ is nonempty. The sequence X is called the *marking* of g, and the nodes in X are referred to as *external nodes*. For $X = v_0 v_1 \cdots v_k$, we write $head(g) = v_0$ and $ext(g) = v_1 \cdots v_k$. We say that two marked graphs are isomorphic modulo markings if their underlying unmarked graphs are isomorphic. The *rank* of a marked graph g is $|ext(g)|$.

Graph operations. Let g be a single-rooted marked DAG with external nodes X and $|ext(g)| = k$. Then g is called a *k-graph* if $head(g)$ is the unique root of g, and all nodes in $ext(g)$ are leaves.

If $head(g)$ has out-degree at most 1 (but is not necessarily the root of g), and either $head(g)$ has out-degree 0 or $ext(g)$ is exactly the reentrant nodes of $g \downarrow head(g)$, then g is a *k-context*. We denote the set of all k-graphs over Σ by $\mathbb{G}^k_\Sigma$, and the set of all k-contexts over Σ by $\mathbb{C}^k_\Sigma$. Furthermore, $\mathbb{G}_\Sigma = \cup_{k\in\mathbb{N}}\mathbb{G}^k_\Sigma$ and $\mathbb{C}_\Sigma = \cup_{k\in\mathbb{N}}\mathbb{C}^k_\Sigma$. Note that the intersection $\mathbb{G}_\Sigma \cap \mathbb{C}_\Sigma$ is typically not empty. Finally, the *empty context* consisting of a single node, which is external, is denoted by ϵ.

Given $g \in \mathbb{G}^k_\Sigma$ and $c \in \mathbb{C}^k_\Sigma$, the *substitution* $c[\![g]\!]$ of g into c is obtained by first taking the disjoint union of g and c, and then merging $head(g)$ and $head(c)$, as well as the sequences $ext(g)$ and $ext(c)$ element-wise. The results is a single-rooted, unmarked DAG.

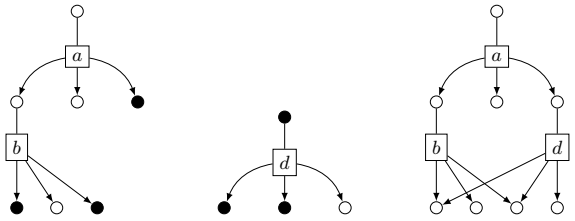

Fig. 1. A 2-context c, a 2-graph g, and the concatenation $c[\![g]\!]$. Filled nodes convey the marking of c and g, respectively. Both targets of edges and external nodes of marked graphs are drawn in order from left to right unless otherwise noted.

Let g be a graph in $\mathbb{G}^0_\Sigma$, e an edge and let h be the marked graph given by taking $g \downarrow e$ and marking the (single) root, and all reentrant nodes. Then the *quotient* of $g \in \mathbb{G}^0_\Sigma$ with respect to h, denoted g/h is the unique context $c \in \mathbb{C}^k_\Sigma$ such that $c[\![h]\!] = g$. The *quotient* of a graph language $L \subseteq \mathbb{G}_\Sigma$ with respect to $g \in \mathbb{G}_\Sigma$ is the set of contexts $L/g = \{c \mid c[\![g]\!] \in L\}$.

Let A be a symbol of rank k. Then $A^\bullet$ is the graph $(V, \{e\}, src, tar, lab, X)$, where $V = \{v_0, v_1, \ldots, v_k\}$, $src(e) = v_0$, $tar(e) = v_1 \ldots v_k$, $lab(e) = A$, and $X = v_0 \ldots v_k$. Similarly, $A^\odot$ is the very same graph, but with only the root marked, in other words, $X = v_0$.

3 Well-ordered DAGs

In this section, we present two formalisms for generating languages of DAGs, one grammatical and one algebraic. Both generate graphs that are *well-ordered* in the sense defined below. We show that the two formalisms define the same families of languages. This allows us to use the algebraic formulation as a basis for the upcoming Myhill-Nerode theorem and MAT learning algorithm.

An edge e with $tar(e) = w$ is a *common ancestor edge* of nodes u and u' if there are t and t' in w such that u is a descendant of t and u' is a descendant of t'. If, in addition, there is no edge with its source in w that is a common ancestor edge of u and u', we say that e is a *closest common ancestor edge* of u and u'. If e is a common ancestor edge of u and v we say that e *orders* u and v, with u before v, if $tar(e)$ can be written as wtw', where t is an ancestor of u and every ancestor of v in $tar(e)$ can be found in w'.

The relation $\preceq_g$ is defined as follows: $u \preceq_g v$ if every closest common ancestor edge e of u and v orders them with u before v. It is a partial order on the leaves of g[3]. Let g be a graph. We call g *well-ordered*, if we can define a total order $\trianglelefteq$ on the leaves of g such that $\preceq_g \subseteq \trianglelefteq$, and for every $v \in V$ and every pair u, u' of leaves of $g{\downarrow}v$, $u \preceq_{g\downarrow v} u'$ implies $u \trianglelefteq u'$.

3.1 Order-preserving DAG grammars

Order-preserving DAG grammars (OPDGs) are essentially hyper-edge replacement grammars with added structural constraints to allow efficient parsing.[1] The idea is to enforce an easily recognisable order on the nodes of the generated graphs, that provides evidence of how they were derived. The constraints are rather strict, but even small relaxations make parsing NP-hard; for details, see [3]. Intuitively, the following holds for any graph g generated by an OPDG:

- g is a connected, single-rooted DAG,
- only leaves of g have in-degree greater than 1, and
- g is well-ordered

Definition 1 (Order-preserving DAG grammar [3]). *An* order-preserving DAG grammar *is a system $H = (\Sigma, N, I, P)$ where Σ and N are disjoint ranked alphabets of* terminals *and* nonterminals*, respectively, I is the set of* starting nonterminals*, and P is a set of* productions*. Each production is of the form $A \to f$ where $A \in N$ and $f \in \mathbb{G}^{rank(A)}_{\Sigma \cup N}$ satisfies one of the following two cases:*

1. *f consists of exactly two nonterminal edges e_1 and e_2, both labelled by A, such that $src(e_1) = src(e_2) = head(f)$ and $tar(e_1) = tar(e_2) = ext(f)$. In this case, we call $A \to f$ a* clone rule*.*
2. *f meets the following restrictions:*
   - *no node has out-degree larger than* 1
   - *if a node has in-degree larger than one, then it is a leaf;*
   - *if a leaf has in-degree exactly one, then it is an external node or its unique incoming edge is terminal*
   - *for every nonterminal edge e in f, all nodes in $tar(e)$ are leaves, and $src(e) \neq head(f)$*
   - *the leaves of f are totally ordered by $\preceq_f$ and $ext(f)$ respects $\preceq_f$.*

[1] In [3], the grammars are called Restricted DAG Grammars, but we prefer to use a name that is more descriptive.

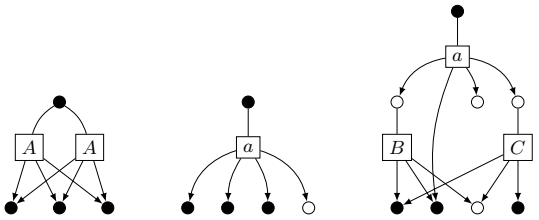

Fig. 2. Examples right-hand sides f of normal form rules of types (a), (b), and (c) for a nonterminal of rank 3.

A derivation step of H is defined as follows. Let $\rho = A \to f$ be a production, g a graph, and g_A a subgraph of g isomorphic modulo markings to $A^{\odot}$. The result of applying ρ to g at g_A is the graph $g' = (g/g_A)[\![f]\!]$, and we write $g \Rightarrow_\rho g'$. Similarly, we write $g \Rightarrow_H^* g'$ if g' can be derived from g in zero or more derivation steps. The language $\mathcal{L}(H)$ of H are all graphs g over the terminal alphabet Σ such that $S^\bullet \Rightarrow_H^* g$, for some $S \in I$. Notice that since a derivation step never removes nodes and never introduces new markings, if we start with a graph g with $|ext(g)| = k$, all derived graphs g' will have $|ext(g')| = k$. In particular, if we start from $S^\bullet$, all derived graphs will have $|ext(g')| = rank(S)$.

Definition 2 (Normal form [3]). *An OPDG H is on* normal form *if every production $A \to f$ is in one of the following forms:*

(a) The rule is a clone rule.
(b) f has a single edge e, which is terminal.
(c) f has height 2, the unique edge e with $src(e) = head(f)$ is terminal, and all other edges are nonterminal.

We say that a pair of grammars H and H' are *language-equivalent* if $\mathcal{L}(H) = \mathcal{L}(H')$. As shown in [3], every OPDG H can be rewritten to a language-equivalent OPDG H' in normal form in polynomial time.

For a given alphabet Σ, we denote the class of graphs $\cup_{H \text{ is an OPDG}} \mathcal{L}(H)$ that can be generated by some OPDG by $\mathcal{H}_\Sigma$, and by $\mathcal{H}_\Sigma^k$ the set of rank k marked graphs that can be generated from a rank k nonterminal.

3.2 DAG concatenation

In sections 4 and 5, we need algebraic operations to assemble and decompose graphs. For this purpose, we define graph concatenation operations that mirror the behaviour of our grammars and show that the class of graphs that can be constructed in this way is equal to $\mathcal{H}_\Sigma$.

In particular, we construct our graphs in two separate ways, mirroring the cloning and non-cloning rules of the grammars:

– 2-concatenation, which takes 2 rank-m graphs and merges their external nodes. This corresponds to the clone rules in Definition 2.

– a-concatenation, for $a \in \Sigma$, takes an a-labelled $rank(a)$ terminal edge and a number (less than or equal to $rank(a)$) of marked graphs, puts the graphs under targets of the terminal edge, and merges some of the leaves. This corresponds to rules of type (b) or (c) in Definition 2.

The second operation is more complex, as we must make sure that the output conforms to the ordering and structural constraints of OPDG. Given a terminal a of rank k and a sequence $g_1, \ldots, g_n$, with $n \leq k$ of marked graphs, we create new graphs in the following way. We start with $a^{\odot}$ and, for each $i \in [n]$ identify $head(g_i)$ with a unique leaf of $a^{\odot}$, intuitively "hanging" $g_1, \ldots, g_n$ under an edge labelled a. We then identify some of the leaves of the resulting graph, but only in such a way that the resulting graph is well-ordered. The intuition is that we mirror productions of type (b) and (c) from Definition 2, but instead of producing a graph containing nonterminal edges, we immediately replace the nonterminals by graphs that can be derived from them. More formally:

Definition 3 (2-concatenation). *Let $m \in \mathbb{N}$ and let $g_1, g_2 \in \mathbb{G}_{\Sigma}^{m}$ be disjoint graphs. A* 2-concatenation $2[g_1, g_2]$ *is obtained by merging the roots and external nodes of g_1 and g_2. The root and external nodes of $2[g_1, g_2]$ are the merged roots and external nodes, respectively.*

Observation 4 (k-concatenation) *An obvious extension of* 2*-concatenation is to use an arbitrary number k of graphs from $\mathbb{G}_{\Sigma}^{m}$ instead of just* 2*. Such operations can be implemented using iterated* 2*-concatenations, and we refer to them as k-concatenations.*

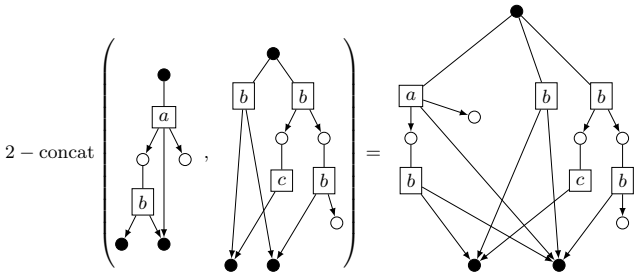

Fig. 3. A 2-concatenation of two graphs

The second operation is more complex, since we must make sure that order is preserved. Given a terminal a of rank k and a sequence $g_1, \ldots, g_n$, with $n \leq k$ of marked graphs, new graphs are created in the following way. We start with $a^{\odot}$ and, for each $i \in [n]$ identify $head(g_i)$ with a unique leaf of $a^{\odot}$, intuitively "hanging" $g_1, \ldots, g_n$ under an edge labelled a. We then identify some of the leaves of the resulting graph. In order to fully specify the result of such a concatenation, and to make sure that it preserves order, we need to parameterize it with the following.

(1) A number m. This is the number of nodes we will merge the external nodes of the graphs $g_1, \ldots, g_n$ and the remaining leaves of the a-labelled edge into.
(2) A subsequence $s = s_1 \ldots s_n$ of $\langle k \rangle$ of length n. This sequence defines under which leaves of $a^{\odot}$ we are going to hang which graph.
(3) A subsequence x of $\langle m \rangle$. This sequence defines which of the leaves of the resulting graph will be external.
(4) An order-preserving function φ that defines which leaves to merge. Its domain consists of the external leaves of the graphs $g_1, \ldots, g_n$ as well as the leaves of $a^{\odot}$ to which no graph from $g_1, \ldots, g_n$ is assigned. Its range is $[m]$.

Before we describe the details of the concatenation operation, we must go into the rather technical definition of what it means for φ to be order-preserving. It has to fulfil the following conditions:

(i) If both u and v are marked leaves of g_i, for some $i \in [n]$, and u comes before v in $ext(g_i)$, then $\varphi(u) < \varphi(v)$.
(ii) If $|\varphi^{-1}(i)| = 1$, then either $i \in x$ or the unique node v with $\varphi(v) = i$ belongs to $a^{\odot}$.
(iii) If there are i and j in $[m]$, with $i < j$ such that no graph g_ℓ for $\ell \in [n]$ contains both a member of $\varphi^{-1}(i)$ and a member of $\varphi^{-1}(j)$, then there exists a $p \in [k]$ such that either
  – p is the qth member of s, and g_q contains a member of $\varphi^{-1}(i)$, or
  – the pth member of $tar(a)$ is in $\varphi^{-1}(i)$
  and furthermore there is no $r < p$ such that either
  – r is the tth member of s and g_t contains a member of $\varphi^{-1}(j)$, or
  – the rth member of $tar(a)$ is itself in $\varphi^{-1}(j)$

Definition 5 (a-concatenation). *Given a terminal a, the a-concatenation of $g_1, \ldots, g_n$, parameterized by m, s, x, ϕ is the graph g obtained by doing the following. For each $i \in [n]$, identify $head(g_i)$ with the leaf of $a^{\odot}$ indicated by s_i. For each $j \in [m]$, identify all nodes in $\varphi^{-1}(j)$. Finally, $ext(g)$ is the subsequence of the m nodes from the previous step indicated by x.*

We note that our concatenation operations can be seen as algebraic operations independent of their input. $2-$concatenation is defined for any pair of graphs, as long as both of them have the same rank. For the $a-$concatenations, things are a little bit more complex, but once we fix the parameters m, s, x, φ, we can see (a, m, s, x, φ) as a well-defined operator that can take any sequence of $|s|$

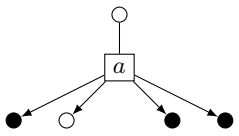

Fig. 4. The topmost terminal graph used in a-concatenation, for a rank-4 terminal symbol a, with the subsequence s of $\langle 4 \rangle$ where subgraphs will be attached indicated as filled leaves

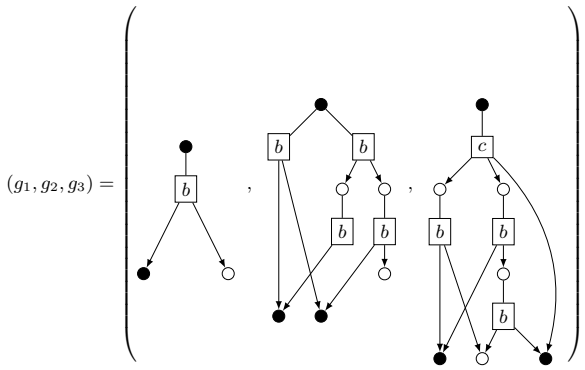

Fig. 5. The (previously constructed) marked graphs, here used as arguments to our example a-concatenation.

Fig. 6. The nodes that leaves are being merged into in an a-concatenation. The subsequence x of external nodes of the concatenated graph are indicated as filled nodes.

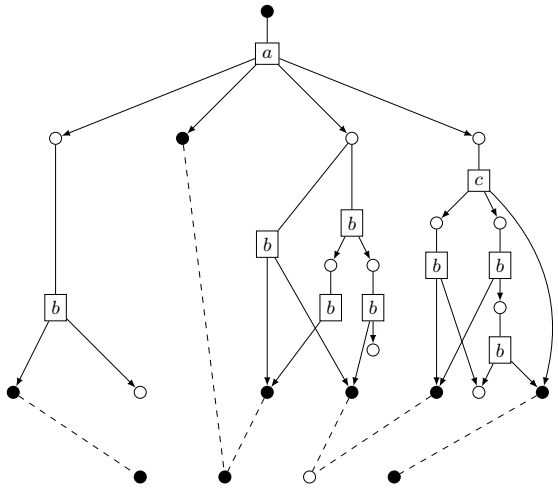

Fig. 7. A "halfway done" a-concatenation, where the input graphs has been hanged underneath the terminal edge, but leaves have not yet been merged. The filled leaves in the graph indicate the domain of φ, and the dashed lines show which node in $\langle m \rangle$ each leaf is merged into. As previously, x is given by the filled nodes of $\langle m \rangle$.

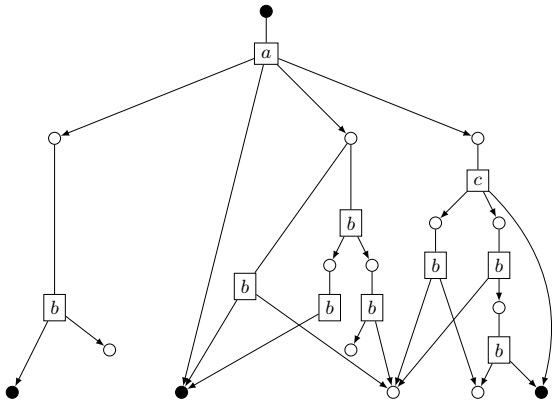

Fig. 8. The marked graphs that is result of the complete a-concatenation, including the merges indicated by φ

graphs as input, as long as their ranks match what φ expects. Indeed, instead of defining the range of φ as the external nodes of the input graphs together with the unused leaves of $a^{\odot}$, i.e., those not indicated by s, we can see it as a function from numbers and pairs of numbers in the following way. If φ is defined for a leaf ℓ of $a^{\odot}$ whose position in $tar(a)$ is i, then we redefine φ so that $\varphi(i) = \varphi(\ell)$. If, on the other hand, ℓ is the jth member of $ext(g_i)$, then we set $\varphi(i, j) = \varphi(\ell)$.

We denote by $\mathcal{A}_\Sigma$ the class of marked graphs that can be assembled from Σ through a- and 2-concatenation, and by $\mathcal{A}^k_\Sigma \subseteq \mathcal{A}_\Sigma$ the graphs of rank k.

Each concatenation operation can be defined as an algebraic operation that takes a number of graphs (of certain ranks) and combines them.

Observation 6 *Let ψ be a concatenation operator and $g_1, \ldots, g_n$ a sequence of graphs for which it is defined. Let $g = \psi(g_1, \ldots, g_n)$. For some $i \in n$, let g' be a graph of the same rank as g_i. Then $\psi(g_1, \ldots, g_{i-1}, g', g_{i+1}, \ldots, g_n) = (g/g_i)[\![g']\!]$.*

The following is the main result of this section.

Theorem 7. *$\mathcal{A}_\Sigma = \mathcal{H}_\Sigma$, and $\mathcal{A}^k_\Sigma = \mathcal{H}^k_\Sigma$ for all k.*

Proof. The proof for $\mathcal{A}_\Sigma \subseteq \mathcal{H}_\Sigma$ is by induction on the size of a graph g. For the base step, we observe that all connected graphs consisting of a single edge with the appropriate number nodes trivially belong to both $\mathcal{A}_\Sigma$ and $\mathcal{H}_\Sigma$. For the inductive step, there are two cases.

We first address 2-concatenation. Let $g = 2[g_1, g_2]$, and assume both g_1 and g_2 are in both $\mathcal{H}^\ell_\Sigma$ and $\mathcal{A}^\ell_\Sigma$, for $\ell = |ext(g)|$. Thus, there are OPDGs H_1 and H_2 generating g_1 and g_2 with some derivations $S^\bullet_i \Rightarrow f_i \Rightarrow^* g_i$ for $i \in \{1, 2\}$. We can construct an OPDG H that generates g by essentially merging H_1 and H_2, and adding a new nonterminal S' of rank ℓ, with a cloning rule and the productions $S' \to f_i$ for $i \in \{1, 2\}$. This will allow the grammar H to generate g, as well as any k-concatenation involving any combination of the same two graphs.

For a-concatenation, we reason as follows. Let g be obtained from $g_1, \ldots, g_n$ by applying a-concatenation, parameterized by m, s, x, φ. Since every g_i is smaller than g, it belongs to $\mathcal{H}_\Sigma$, and hence there is an OPDG H_i with initial nonterminal S_i, such that $S_i^\bullet \Rightarrow^* g_i$. We construct an OPDG H such that $g \in \mathcal{L}(H)$ as follows:

- We add all the productions, nonterminals etc. from H_i, $i \in [n]$, keeping the sets of nonterminals disjoint.
- We add a starting nonterminal S' of rank $|ext(g)|$ and the production $S' \Rightarrow f$ where $f = (a, m, s, x, \varphi)[S_1^\bullet, \ldots, S_n^\bullet]$.

By the context-freeness lemma for HRGs[8], there is a derivation $f \Rightarrow^* g$. It remains to be proved that f is a valid right-hand side, and thus H a valid OPDG. The single edge connected to the root of f is terminal. There are no nodes of out-degree greater than 1, and only a single layer of n nonterminal edges. Leaves of in-degree 1 are either connected to the terminal or are external (or both). This is ensured by item (ii) in the requirements for φ to be order-preserving. Any nonexternal leaves are either connected to the terminal or at least of in-degree 2. Let us now check that the leaves are totally ordered by $\preceq_f$, and moreover that $ext(f)$ respects it. As there are no nodes of out-degree greater than 1, the only way $\preceq_f$ can fail to be total is if two leaves u, v have two closest common ancestor edges e_i, e_j such that u comes before v in $tar(e_i)$, but not in $tar(e_j)$. However, the requirement that φ is order-preserving precludes this.

To prove the opposite direction, let $H = (\Sigma, N, S, P)$ be an OPDG on normal form. We show by induction on the length of the derivations that both the terminal graphs and the intermediate graphs that arise during the derivations are in $\mathcal{A}_{\Sigma \cup N}$.

Again, the base case is trivial — for any start symbol S, the graph $S^\bullet$ clearly belongs $\mathcal{A}_{\Sigma \cup N}$. Moving on to the inductive step, we assume that we have a derivation $S^\bullet \Rightarrow^* g \Rightarrow_{A \to f} g'$ with $g \in \mathcal{A}_{\Sigma \cup N}$. For the rule $A \to f$ to be applicable, g must have a subgraph h that is isomorphic modulo markings to $A^\bullet$. We know that $g' = (g/h)[\![f]\!]$. We first argue that if $f \in \mathcal{A}_{\Sigma \cup N}$, then $g' \in \mathcal{A}_{\Sigma \cup N}$. For since h is a part of g, any construction of g using concatenation operators must at some point use a concatenation operation $\psi(g_1, \ldots, g_n)$, with $g_i = A^\bullet$ for some $i \in [n]$, resulting in a graph h'. By Observation 6, if we use f instead of $A^\bullet$ in this operation, we get a graph $(h'/A^\bullet)[\![f]\!]$. If in all later concatenations, we use this graph instead of h', then, by induction, we will in the end obtain $(g/h)[\![f]\!] = g'$. It remains to show that $f \in \mathcal{A}_{\Sigma \cup N}$. There are three cases:

(1) $A \to f$ is a clone rule, in which case $f = 2[A^\bullet, A^\bullet]$.
(2) f is a single terminal edge, in which case it is also clearly a member of $\mathcal{A}_\Sigma$.
(3) f is of height 2, and the single edge of rank k connected to the root is terminal. This closely mirrors an a-concatenation, where the graphs $g_1, \ldots g_n$ are graphs with just a single nonterminal edge each. The parameters of the concatenation can be read directly from the form of f. The one thing to notice is that the conditions that the leaves of f be totally ordered by $\preceq_f$ and that $ext(f)$ respect $\preceq_f$ ensures that we can find an order-preserving φ and make x be a subsequence of $\langle m \rangle$. □

4 A Myhill-Nerode theorem

We begin by defining the Nerode congruence for ordered DAG languages. From here on, let L be such a language. Intuitively, a pair of graphs are equivalent with respect to L if they can be freely substituted for one another in any context, without disturbing the resulting graph's membership in L. For our purposes, it is useful to view the Nerode congruence as a corner case in a family of relations, each focusing on a subset of $\mathbb{C}_\Sigma$.

Definition 8. *Let $C \subseteq \mathbb{C}_\Sigma$. The equivalence relation $\equiv_{L,C}$ on $\mathcal{H}_\Sigma$ is given by: $g \equiv_{L,C} g'$ if and only if $(L/g \cap C) = (L/g' \cap C)$. The relation $\equiv_{L,\mathbb{C}_\Sigma}$ is known as the* Nerode congruence with respect to L *and is written $\equiv_L$.*

It is easy to see that for two graphs to be equivalent, they must have equally many external nodes. The graph g is *dead* (with respect to L) if $L/g = \emptyset$, and graphs that are not dead are *live*. Thus, if $\equiv_L$ has finite index, there must be a $k \in \mathbb{N}$ such that every $g \in \mathcal{H}_\Sigma$ with more than k external nodes is dead.

In the following, we use $\Psi(\Sigma)$ to denote the set of all concatenation operators applicable to graphs over Σ.

Definition 9 (Σ-expansion). *Given $N \subseteq \mathcal{A}_\Sigma$, we write $\Sigma(N)$ for the set:*

$$\{\psi(g_1, \ldots, g_m) \mid \psi \in \Psi(\Sigma), g_1, \ldots, g_m \in N \text{ and } \psi(g_1, \ldots, g_m) \text{ is defined}\}\ .$$

In the upcoming Section 5, Theorem 13 will form the basis for a MAT learning algorithm. As is common, this algorithm maintains an *observation table* T that collects the information needed to build a finite-state device for the target language L. The construction of an OPDG G^{T} from T is very similar to that from the Nerode congruence, so by introducing it here, we can make use of it twice. Intuitively, the observation table is made up of two sets of graphs N and P, representing nonterminals and production rules, respectively, and a set of contexts C used to explore the congruence classes of $N \cup P$ with respect to L.

To facilitate the design of new MAT learning algorithms, the authors of [6] introduce the notion of an *abstract observation table* (AOT); an abstract data type guaranteed to uphold certain helpful invariants.

Definition 10 (Abstract observation table, see [6]). *Let $N \subseteq P \subseteq \Sigma(N) \subseteq \mathcal{A}_\Sigma$, with N finite. Let $C \subseteq \mathbb{C}_\Sigma$, and let $\rho : P \mapsto N$. The tuple (N, P, C, ρ) is an* abstract observation table with respect to L *if for every $g \in P$,*

1. *$L/g \neq \emptyset$, and*
2. *$\forall g' \in N \setminus \{\rho(g)\} : g \not\equiv_{L,C} g'$.*

The AOT in [6] accommodates production weights taken from general semirings. The version that we have recalled here has three modifications: First, we dispense with the sign-of-life function that maps every graph $g \in N$ to an element in L/g. Its usages in [6] are to avoid dead graphs, and to compute the

weights of productions involving g. From the way new productions and nonterminals are discovered, we already know that they are live, and as we are working in the Boolean setting, there are no transition weights to worry about. Second, we explicitly represent the set of contexts C to prove that the nonterminals in N are distinct. Both realisations of the AOT discussed in [6] collect such contexts, though it is not enforced by the AOT. Third, we do not require that $L(g) = L(\rho(g))$, as this condition is not necessary for correctness, though it may reduce the number of counterexamples needed. The data fields and procedures have also been renamed to reflect the shift from automata to grammars. From here on, we use a bold font when referring to graphs as nonterminals.

Definition 11. *Let $\mathrm{T} = (N, P, C, \rho)$ be an AOT with respect to L. Then G^{T} is the OPDG $(\Sigma, N^{\mathrm{T}}, I^{\mathrm{T}}, P^{\mathrm{T}})$ where $N^{\mathrm{T}} = N$, $I^{\mathrm{T}} = N \cap L$, and*

$$P^{\mathrm{T}} = \{\boldsymbol{\rho(g)} \to \psi(\boldsymbol{\rho(g_1)}, \ldots, \boldsymbol{\rho(g_m)}) \mid g = \psi(g_1, \ldots, g_m) \in P\} \ .$$

In preparation for Theorem 13, we expand our technical vocabulary. Given an ODPG $G = (\Sigma, N, I, P)$ and a nonterminal $\boldsymbol{f} \in N$, $G_{\boldsymbol{f}} = (\Sigma, N, \{\boldsymbol{f}\}, P)$. The grammar G is *unambiguous* if for every $\boldsymbol{g}, \boldsymbol{h} \in N$, $\mathcal{L}(G_{\boldsymbol{g}}) \cap \mathcal{L}(G_{\boldsymbol{h}}) \neq \emptyset$ implies that $\boldsymbol{g} = \boldsymbol{h}$.

Lemma 12. *If $\equiv_L$ has finite index, then there is a $k \in \mathbb{N}_0$ such that for graph $g \in \mathcal{H}_\Sigma$ with more than k external nodes, L/g is empty.*

Proof. If $|ext(g)| \neq |ext(g')|$, then $L/g \cap L/g' = \emptyset$. By Definition 8, this means that either $L/g = L/g' = \emptyset$, or that $g \not\equiv_L g'$. Since $\equiv_L$ has finite index, $\{ext(g) \mid L/g \neq \emptyset\}$ must be bounded from above. □

Theorem 13 (Myhill-Nerode theorem). *The language L can be generated by an OPDG if and only if $\equiv_L$ has finite index. Furthermore, there is a minimal unambiguous OPDG G_L with $\mathcal{L}(G_L) = L$ that has one nonterminal for every live equivalence class of $\equiv_L$. The OPDG G_L is unique up to nonterminal names.*

Proof. We begin by proving the "if" direction. Let $D = \{g \in \mathcal{A}_\Sigma \mid g \text{ is dead}\}$, and N be a selection of representative elements of $(\mathcal{A}_\Sigma / \equiv_L) \setminus \{D\}$. Let $P = \Sigma(N) \setminus D$. Since L has finite index, N and P are finite sets. Finally, let $C = \mathbb{C}_\Sigma$ and, for every $g \in P$, let $\rho(g)$ be the representative of $[g]_{\equiv_L}$ in N. It is easy to verify that $\mathrm{T} = (N, P, C, \rho)$ is an abstract observation table.

Let us now argue by contradiction that (1) $g \in \mathcal{L}(G^{\mathrm{T}}_{\boldsymbol{\rho(g)}})$ and $g \notin \mathcal{L}(G^{\mathrm{T}}_{\boldsymbol{g'}})$, for every $g \in \mathcal{A}_\Sigma \setminus D$ and $g' \in N \setminus \{\rho g\}$. Suppose that $g \notin \mathcal{L}(G^{\mathrm{T}}_{\boldsymbol{\rho(g)}})$. We decompose g into $c[\![g']\!]$ such that Statement 1 is not true for $g' = \psi(g_1, \ldots, g_m)$, but it is true for every proper subgraph of g'.

By construction of T, there is a graph $h' = \psi(\rho(g_1), \ldots, \rho(g_m)) \in P$, and hence a production

$$\boldsymbol{\rho(h')} \to \psi(\boldsymbol{\rho(g_1)}, \ldots, \boldsymbol{\rho(g_m)}) \in P^{\mathrm{T}} \ .$$

Since $g_1 \equiv_L \rho(g_1)$, we have $\psi(\rho(g_1), \rho(g_2), \ldots, \rho(g_m) \equiv_L \psi(g_1, \rho(g_2), \ldots, \rho(g_m))$. As $g_i \equiv_L \rho(g_i)$ for all $i \in [m]$, we can repeat the argument $m-1$ times, and learn that

$$h' = \psi(\rho(g_1), \ldots, \rho(g_m)) \equiv_L \psi(g_1, \ldots, g_m) = g' \ .$$

This means that $g' \in \mathcal{L}(G^{\mathrm{T}}_{\boldsymbol{\rho(h')}}) = \mathcal{L}(G^{\mathrm{T}}_{\boldsymbol{\rho(g')}})$, contrary to our initial assumption.

The "if" direction is completed by noticing that since $g \in \mathcal{L}(G^{\mathrm{T}}_{\boldsymbol{\rho(g)}})$,

$$g \in \mathcal{L}(G^{\mathrm{T}}) \iff \boldsymbol{\rho(g)} \in I \iff \rho(g) \in L \iff g \in L \ .$$

Now for the proof of the "only if" direction. Assume that L is generated by the OPDG $H = (\Sigma, N, I, P)$. For every $g \in \mathcal{H}_\Sigma$, let $NT(g) = \{A \in N \mid g \in \mathcal{L}(H_A)\}$. We show that if, for $g, g' \in \mathcal{H}_\Sigma$, $NT(g) = NT(g')$, then $L/g = L/g'$. Suppose that $NT(g) = NT(g')$ and that $c \in L/g$. This means that there is a derivation $I \Rightarrow^* c[\![A]\!] \Rightarrow^* c[\![g]\!]$. Since A is also in $NT(g')$, there is an alternative derivation $I \Rightarrow^* c[\![A]\!] \Rightarrow^* c[\![g']\!]$. This is due to the context-freeness of the grammars; see, e.g., [8], and implies that $c \in L/g'$ which proves the claim. As the powerset of N is finite, so is the index of $\equiv_L$. This completes the "only if" direction.

To see that G^{T} is an unambiguous OPDG, we note that if $g, h \in \mathcal{L}(G^{\mathrm{T}})_{\boldsymbol{f}}$ for some $f \in N$, then $g \equiv h$. There cannot be an unambiguous OPDG with fewer nonterminals, since then two graphs belonging to different congruence classes would be generated from the same nonterminal $\boldsymbol{f}$, and since they can *only* be generated from $\boldsymbol{f}$, they would appear in exactly the same set of contexts. G^{T} has thus the minimal number of nonterminals. Neither can any production be removed, as every production is used in the generation of some live graph $g \in P$, and removing it would cancel all graphs on the form $c[\![g]\!]$ from the language. We conclude that G^{T} is a minimal unambiguous OPDG for L, and that it is unique up to renaming of nonterminals. □

Notice that when L only contains ordered ranked trees (i.e., when the root has exactly one child and no node has more than one ancestor), then Theorem 13 turns into the Myhill-Nerode theorem for regular tree languages [12], and the constructed device is essentially the minimal bottom-up tree automaton for L.

5 MAT learnability

In Section 4, the data fields N, P, and C of the AOT were populated with what is usually called a *characteristic set* for L, to derive the minimal unambiguous OPDG G_L that generates L. In this section, we describe how the necessary information can be incrementally built up by querying a MAT oracle. The learning algorithm interacts with the oracle through the following procedures:

- Equals?(H) returns a graph in $\mathcal{L}(H) \ominus L = \{g \mid \mathcal{L}(H)(g) \neq L(g)\}$, or $\perp$ if no such exists.
- Member?(g) returns the Boolean value $L(g)$.

The information gathered from the oracle is written and read from the AOT through the procedures listed below. In the declaration of these, (N, P, C, ρ) and (N', P, C', ρ') are the data values before and after application, respectively. The procedures are then as follows:

- INITIALISE sets $N' = P' = C' = \emptyset$.
- ADDPRODUCTION(g) with $g \in \Sigma(N) \setminus P$. Requires that $L/g \neq \emptyset$, and guarantees that $N \subseteq N'$ and $P \cup \{g\} \subseteq P'$.
- ADDNONTERMINAL(c, g) with $g \in P \setminus N$ and $c \in \mathbb{C}_\Sigma$. Requires that $\forall g' \in N : g \not\equiv_{L, C \cup \{c\}} g'$, and guarantees that $N \cup \{g\} \subseteq N'$, $P \subseteq P'$, and $C \subseteq C' \subseteq C \cup \{c\}$.
- GRAMMAR returns G^{T} without modifying the data fields.

Algorithms 1 and 2 are recalled almost exactly as they stand in [6], with the only adjustments being those needed to go from weighted automata to unweighted grammars. Algorithm 1 maintains an AOT T, from which it induces an OPDG G^{T}. This OPDG is given to the language oracle LANG in the form of an equivalence query. If the oracle responds with the token $\perp$, then the language has been successfully acquired. Otherwise, the algorithm receives a counterexample $g \in \mathcal{L}(G^{\mathrm{T}}) \ominus L$, from which it extracts new facts about L through the procedure EXTEND and includes these in T.

The technique used in Algorithm 2, EXTEND, is known as *contradiction backtracking*. We cover it superficially here; a closer discussion is available in [7]. The contradiction backtracking essentially consists of simulating the parsing of the counterexample g with respect to the OPDG G^{T}. The simulation is done incrementally, and in each step a subgraph $h \in \Sigma(N) \setminus N$ of g is nondeterministically selected. If h is not in P, this indicates that a production is missing from G^{T} and the problem is solved by a call to ADDPRODUCTION. If h is in P, then the algorithm replaces it by $\rho(h)$ and checks whether the resulting graph g' is in L. If its membership has changed (i.e., if $L(g) \neq L(g')$), then

evidence has been found that h and $\rho(h)$ do not represent the same congruence class and the algorithm calls ADDNONTERMINAL. If the membership has not changed, then the procedure calls itself recursively with the graph g' as argument, which has strictly fewer subgraphs not in P. Since g is a counterexample, so is g'.

If this parsing process succeeds in replacing all of g with a graph $g' \in N$, then $L(g) = L(g')$ and $g \in \mathcal{L}(G^{\mathrm{T}}_{g'})$. Since $g' \in N$, $\mathcal{L}(G^{\mathrm{T}})(g') = L(g')$. It follows that $\mathcal{L}(G^{\mathrm{T}})(g) = L(g)$ which contradicts g being a counterexample.

From [6], we know that if EXTEND adheres to the pre- and postconditions of the AOT procedures, and the target language L can be computed by an OPDG, then Algorithm 1 terminates and returns a minimal OPDG generating L. It thus remains to add realisations of ADDPRODUCTION and ADDNONTERMINAL, and to show that all procedures behave as desired.

Algorithm 1: Template learning algorithm [6]

```
T.Initialise();
while true do
    G^T ← T.Grammar();
    g ← Lang.Equal?(G^T);
    if g = ⊥ then
        return G^T
    else
        T.Extend(g)
```

Algorithm 2: The procedure Extend [6]

```
Data: g ∈ L(G^T) ⊖ L
Decompose g into g = c⟦h⟧ where c ∈ ℂ_Σ, h ∈ Σ(N) \ N;
if h ∉ P then
    T.AddProduction(h);
else
    if Lang.Member?(c⟦h⟧) ≠ Lang.Member?(c⟦ρ(h)⟧) then
        T.AddNonterminal(c, h);
    else
        Extend(c⟦ρ(h)⟧);
```

Consider the implementations of AddProduction and AddNonterminal, shown in Algorithm 3 and Algorithm 4, respectively. AddProduction simply adds its argument g to the set P of graphs representing productions. It then looks for a representative g' for g in N, such that $g' \equiv_{L,C} g$. If no such graph exists, it simply chooses any $g' \in N$, or if N is empty, adds g itself to N with a call to AddNonterminal. Similarly, AddNonterminal adds g to the set N of graphs representing nonterminals. If g cannot be distinguished from $\rho(g)$, which is the only element in N that could possibly be indistinguishable from g, then c is added to C to tell g and $\rho(g)$ apart. Finally, the representative function ρ is updated to satisfy Condition 2 of Definition 10.

It is easy to verify that (i) the proposed procedures deliver on their guarantees if their requirements are fulfilled, (ii) that where they are invoked, the requirements are indeed fulfilled, and (iii) the conditions on the observation table given in Definition 10 are always met. By [6, Corollary 8], we arrive at Theorem 15.

Algorithm 3: The procedure AddProduction

Data: $p \in \Sigma(N) \setminus P$
$P \leftarrow P \cup \{g\}$;
if $\exists g' \in N : g \equiv_{L,C} g'$ **then**
  $\rho(g) \leftarrow g'$;
else
  if $\exists g' \in N$ **then**
    $\rho(g) \leftarrow g'$;
  else
    AddNonterminal(ϵ, g);

Algorithm 4: The procedure AddNonterminal

Data: $g \in P \setminus N$, $c \in \mathbb{C}_\Sigma$, and $\forall g' \in N : g \not\equiv_{L,C\cup\{c\}} g'$
$N \leftarrow N \cup \{g\}$;
if $g \equiv_{L,C} \rho(g)$ **then**
  $C \leftarrow C \cup \{c\}$;
$g' \leftarrow \rho(g)$;
for $h \in \rho^{-1}(g')$ **do**
  if $h \equiv_{L,C} g$ **then**
    $\rho(h) \leftarrow g$;

Lemma 14. *For every $g \in P$, $g \in \mathcal{L}(G^{\mathrm{T}}_{\boldsymbol{\rho(g)}})$.*

Proof. We first prove that for every $g \in N$, $g \in \mathcal{L}(G^{\mathrm{T}}_{\boldsymbol{g}})$. The argument is by induction on the number of edges in g. If g consists of a single edge, then $g = \psi$ for some concatenation operator of rank 0, so the result is trivially true. Assume now that $g = \psi(g_1, \ldots, g_m)$. Since $N \subseteq P \subseteq \Sigma(N)$, there is a production $\boldsymbol{g} \to \psi(\boldsymbol{g_1}, \ldots, \boldsymbol{g_m}) \in P^{\mathrm{T}}$ and $g_i \in N$, $i \in [m]$. By the induction hypothesis, $g_i \in \mathcal{L}(G^{\mathrm{T}}_{\boldsymbol{g_i}})$, $i \in [m]$. It follows that $g \in \mathcal{L}(G^{\mathrm{T}}_{\boldsymbol{g}})$.

Assume now that $g = \psi(g_1, \ldots, g_m) \in P$. Since $P \subseteq \Sigma(N)$, $g_i \in N$, $i \in [m]$. By the above argument, $g_i \in \mathcal{L}(G^{\mathrm{T}}_{\boldsymbol{g_i}})$ for every $i \in [m]$, and since $g \in P$,

$$\boldsymbol{\rho(g)} \to \psi(\boldsymbol{\rho(g_1)}, \ldots, \boldsymbol{\rho(g_m)}) \;=\; \boldsymbol{\rho(g)} \to \psi(\boldsymbol{g_1}, \ldots, \boldsymbol{g_m}) \in P^{\mathrm{T}}$$

so $g \in \mathcal{L}(G^{\mathrm{T}})_{\boldsymbol{\rho(g)}}$. □

Theorem 15. *Algorithm 1 terminates and returns G_L.*

Proof. It should be clear that Initialise trivially fulfils the conditions of Definition 10, and that Grammar has no effect on the data fields at all. Since AddProduction depends on AddNonterminal, we begin verifying the latter.

We us assume that $g \in P \setminus N$, $c \in \mathbb{C}_\Sigma$, and that $\forall g' \in N : g' \not\equiv_{L,C\cup\{c\}} g$. Since N is updated to $N \cup \{g\}$, and P is unchanged, the guarantees of AddNonterminal are fulfilled. Condition 1 of Definition 10 is not affected, and the requirement that $\forall g' \in N : g' \not\equiv_{L,C\cup\{c\}}$ ensures that Condition 2 continues to hold.

Let us now look at AddProduction. Here, we assume that $p \in \Sigma(N) \setminus P$, $c \in \mathbb{C}_\Sigma$, and $L/g \neq \emptyset$, which immediately fulfils Condition 1 of Definition 10, and since N is not updated, Condition 2 is trivially met. Finally we note that in the call to AddNonterminal, $N = \emptyset$, so ϵ trivially a separates g from every other graph in N.

We conclude by ensuring that the AddProduction and AddNonterminal are called from Extend with their requirements met. In case of AddProduction, we know that $c[\![g]\!] \in L$ since $g \notin P$ so $c[\![g]\!] \notin \mathcal{L}(G^{\mathrm{T}})$ and $c[\![g]\!]$ is supposed to be a counterexample. This means in particular that $\{c\} \subseteq L/g$, so L/g is not empty. Also the requirement of AddProduction is met due to the if-clause on Line 2, since by assumption $\forall g' \in N \setminus \{\rho(g)\} : g' \not\equiv_{L,C} g$ and we know that $c \in L/g \ominus L/\rho(g)$.

Since the Conditions of Definition 10 are respected, and the associated procedures have their requirements met and fulfil their guarantees, [6, Corollary 8] ensures that the learning algorithm terminates and outputs G_L. □

We close this section with a discussion of the complexity of Algorithm 1. To infer the minimal unambiguous ODGP $G_L = (\Sigma, N, I, P)$ recognising L, the algorithm must gather as many graphs as there are nonterminals and transitions in G_L. In each iteration of the main loop, it parses a counterexample g in polynomial time in the size of g and T (the latter is limited by the size of G_L), and is rewarded with at least one production or nonterminal. The algorithm is thus polynomial in $|G_L| = |N| + |P|$ and the combined size of the counterexamples provided by the MAT oracle.

References

1. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.
2. L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider. Abstract meaning representation for sembanking. In *7th Linguistic Annotation Workshop (ACL 2013 Workshop)*, 2013.
3. H. Björklund, F. Drewes, and P. Ericson. Between a rock and a hard place – uniform parsing for hyperedge replacement DAG grammars. In A.-H. Dediu, J. Janousek, C. Martín-Vide, and B. Truthe, editors, *10th International Conference on Language and Automata Theory and Applications, Prague, Czech Republic, 2016*, volume 9618 of *Lecture Notes in Computer Science*, pages 521–532. Springer, 2016.
4. J. Björklund, H. Fernau, and A. Kasprzik. Polynomial inference of universal automata from membership and equivalence queries. *Information and Computation*, 246:3–19, 2016.
5. D. Chiang, J. Andreas, D. Bauer, K. M. Hermann, B. Jones, and K. Knight. Parsing graphs with hyperedge replacement grammars. In *51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, volume Volume 1: Long Papers, pages 924–932. The Association for Computer Linguistics, 2013.
6. F. Drewes, J. Björklund, and A. Maletti. MAT learners for tree series: an abstract data type and two realizations. *Acta Informatica*, 48(3):165, 2011.
7. F. Drewes and J. Högberg. Query learning of regular tree languages: How to avoid dead states. *Theory of Computing Systems*, 40(2):163–185, 2007.
8. F. Drewes, H.-J. Kreowski, and A. Habel. Hyperedge replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars*, volume 1, pages 95–162. World Scientific, 1997.
9. F. Drewes and H. Vogler. Learning deterministically recognizable tree series. *Journal of Automata, Languages and Combinatorics*, 12(3):332–354, 2007.
10. S. Hara and T. Shoudai. Polynomial time MAT learning of c-deterministic regular formal graph systems. In *International Conference on Advanced Applied Informatics (IIAI AAI 2014)*, pages 204–211, 2014.
11. J. Högberg. A randomised inference algorithm for regular tree languages. *Natural Language Engineering*, 17(02):203–219, 2011.
12. Dexter Kozen. On the Myhill-Nerode theorem for trees. *Bulletin of the EATCS*, 47:170–173, 1992.
13. A. Maletti. Learning deterministically recognizable tree series—revisited. In *Algebraic Informatics*, pages 218–235. Springer, 2007.
14. R. Okada, S. Matsumoto, T. Uchida, Y. Suzuki, and T. Shoudai. Exact learning of finite unions of graph patterns from queries. In *The 18th International Conference on Algorithmic Learning Theory (ALT 2007)*, pages 298–312, 2007.
15. G. Rozenberg and E. Welzl. Boundary NLC graph grammars—basic definitions, normal forms, and complexity. *Information and Control*, 69(1-3):136–167, 1986.
16. Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76(2–3):223–242, 1990.
17. H. Shirakawa and T. Yokomori. Polynomial-time MAT learning of c-deterministic context-free grammars. *Transaction of Information Processing Society of Japan*, 34:380–390, 1993.