On the Regularity and Learnability of Ordered DAG Languages

Henrik Björklund(✉), Johanna Björklund, and Petter Ericson

Department of Computing Science, Umeå University, Umeå, Sweden
henrikb@cs.umu.se

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, for example, [16] and [9, 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) [4]. This is a restricted type of hyper-edge replacement grammars [9] 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 $O(n^2 + nm)$, 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 [6] of Hyperedge Replacement Grammars (HRGs) to abstract meaning representations (AMRs) [2]. An AMR is a directed acyclic graph that describes the semantics

J. Björklund—Supported by the Swedish Research Council, Grant No. 621-2012-4555.

A. Carayol and C. Nicaud (Eds.): CIAA 2017, LNCS 10329, pp. 27–39, 2017.
DOI: 10.1007/978-3-319-60134-2_3

of a natural language sentence, and a corpus with approx. 8 000 AMRs has been compiled by the Information Sciences Institute (ISI) at USC.[1] The formalisation of AMRs is still under discussion, but although restricted, OPDGs retain sufficient expressive power to capture the AMRs in the ISI corpus.

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 (often called 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 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 [1,5,10,12,14,17,18]. 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 [15]. 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 consider MAT learning for context-deterministic regular formal graph systems [11]. 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.

Due to space limitations, most proofs have been omitted, but are available in a technical report [3].

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

[1] The ISI corpus is available at http://amr.isi.edu.

equivalence relation on S, $(S/\equiv)$ denotes the partitioning of S into equivalence classes induced by $\equiv$. The *index* of $\equiv$ is $|(S/\equiv)|$.

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 $i \leq j$.

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 $g\downarrow src(e)$ 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,

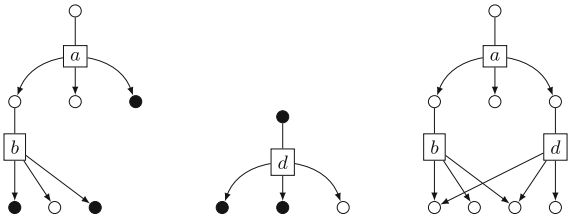

Fig. 1. A 2-context c, a 2-graph g, and the substitution $c[\![g]\!]$. Filled nodes convey the marking of c and g, respectively. Both targets of edges and external nodes of marked graphs are drawn in order from left to right.

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. For an example, see Fig. 1.

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 [4]. 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.[2] 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 [4]. 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 [4]). *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 $f \in \mathbb{G}_{\Sigma\cup N}^{rank(A)}$ 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*

[2] In [4], the grammars are called Restricted DAG Grammars, but we prefer to use a name that is more descriptive.

– *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 $\preceq_f$.*

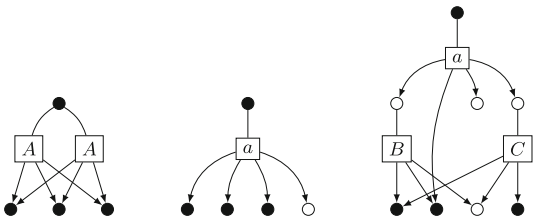

Fig. 2. Examples right-hand sides f of normal form rules of types (a), (b), and (c) for a nonterminal of rank 3.

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 [4]). *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 [4], every OPDG H can be rewritten to a language-equivalent OPDG H' in normal form in polynomial time. For an example of normal form rules, see Fig. 2.

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 class of rank k marked graphs that can be generated from a rank k nonterminal.

3.2 DAG Concatenation

In Sects. 4 and 5, we need algebraic operations to assemble and decompose graphs. For this purpose, we define graph concatenation operations that mimic

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, preserving their order, corresponding 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, 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. To specify the result of such a concatenation, and to ensure that it preserves order, we equip it with the following parameters.

(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 3 (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 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 is defined for a sequence of graphs if they have the appropriate ranks. 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 4. *$\mathcal{A}_\Sigma = \mathcal{H}_\Sigma$, and $\mathcal{A}^k_\Sigma = \mathcal{H}^k_\Sigma$ for all k.*

4 A Myhill-Nerode Theorem

This section defines the Nerode congruence $\equiv_L$ for an ordered DAG language L. A pair of graphs are congruent with respect to L, if they can be replaced by one another in any context in $\mathbb{C}_\Sigma$, without disturbing the encompassing graph's membership in L. The learning algorithm in Sect. 5 produces increasingly more refined approximations of $\equiv_L$ until it reaches $\equiv_L$ itself. This treats $\equiv_L$ as a corner case in a family of relations, each induced by a subset of $\mathbb{C}_\Sigma$.

Definition 5. *Let $C \subseteq \mathbb{C}_\Sigma$. The equivalence relation $\equiv_{L,C}$ on $\mathcal{A}_\Sigma$ 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 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 $g \in \mathcal{A}_\Sigma$ 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 6 (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 Sect. 5, Theorem 9 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 similar to that from the Nerode congruence, so introducing it here avoids repetition. 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 [7] introduce the notion of an *abstract observation table* (AOT); an abstract data type guaranteed to uphold certain helpful invariants.

Definition 7 (Abstract observation table, see [7]). *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 [7] accommodates production weights taken from general semirings. The version recalled here has a number of 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 [7] 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 [7] 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. This change is only superficial, as there is a direct correspondence between states and nonterminals, transitions and productions, and accepting states and initial nonterminals. From here on, a bold font is used to refer to graphs as nonterminals.

Definition 8. *Let $\mathrm{T} = (N, P, C, \rho)$ be an AOT with respect to L. Then G^T is the ODPG $(\Sigma, N^\mathrm{T}, I^\mathrm{T}, P^\mathrm{T})$ where $N^\mathrm{T} = N$, $I^\mathrm{T} = N \cap L$, and*

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

Given an ODPG $G = (\Sigma, N, I, P)$ and a nonterminal $\boldsymbol{f} \in N$, we let $G_{\boldsymbol{f}} = (\Sigma, N, \{\boldsymbol{f}\}, P)$. 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 $\boldsymbol{g} = \boldsymbol{h}$.

Theorem 9 (Myhill-Nerode theorem). *The language $L \subseteq \mathcal{A}_\Sigma$ 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 $\equiv_L$, and this is unique up to nonterminal names.*

In the following proof sketch, $D = \{g \in \mathcal{A}_\Sigma \mid g \text{ is dead}\}$. In the "if" direction, we consider an AOT (N, P, C, ρ) where N contains representative elements of $(\mathcal{A}_\Sigma / \equiv_L) \setminus \{D\}$, $P = \Sigma(N) \setminus D$, $C = \mathbb{C}_\Sigma$, and, for every $g \in P$, $\rho(g)$ is the representative of g's equivalence class in N. From the fact that $g \in \mathcal{L}(G^\mathrm{T}_{\boldsymbol{\rho(g)}})$,

for every $g \in \mathcal{A}_\Sigma$, the first result follows. In the "only if" direction, we note that if G is an OPDG with nonterminals N, and $\mathcal{L}(G_A)(g) = \mathcal{L}(G_A)(h)$ for $g, h \in \mathcal{A}_\Sigma$ and all $A \in N$, then $g \equiv_{\mathcal{L}(G)} h$. As N is finite, so is the index of $\equiv_{\mathcal{L}(G)}$.

Notice that when L only contains ordered ranked trees (i.e., when the root has out-degree one and no node has in-degree greater than one), Theorem 9 turns into the Myhill-Nerode theorem for regular tree languages [13], and the constructed device is essentially the minimal bottom-up tree automaton for L.

5 MAT Learnability

In Sect. 4, the data fields of the AOT were populated with a so-called characteristic set for L, and this yielded the minimal unambiguous OPDG G_L for L. In this section, we describe how the necessary information can be incrementally built up by querying a MAT oracle. Due to space restrictions, background results are covered in brief and we refer to [3] for a detailed exposition.

The learning algorithm (henceforth; the learner) interacts with the oracle (the teacher) through the following procedures:

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

The information gathered from the teacher 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.

- 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.

The learner and the procedure Extend are as they stand in [7]. The learner maintains an AOT T, from which it induces an OPDG G^{T}. This OPDG is given to the teacher in the form of an equivalence query. If the teacher responds with the token $\perp$, then the language has been successfully acquired. Otherwise, the learner 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 procedure Extend uses contradiction backtracking to gain new knowledge from the counterexample g [8]. This consists in simulating the parsing of 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 learner replaces it by $\rho(h)$ and checks whether the resulting graph g' is in L. If $L(g) \neq L(g')$,

Algorithm 1. 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 2. 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**
    $\rho(h) \leftarrow g$;

then evidence has been found that h and $\rho(h)$ do not represent the same congruence class and the learner 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 [7], 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 the learner terminates and returns a minimal OPDG generating L. It thus remains to discuss the procedures AddProduction and AddNonterminal (Algorithms 1 and 2, respectively), and show that these behave as desired. The procedure 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 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 Definition 7.

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 7 are always met. By [7, Corollary 8], we arrive at Theorem 10.

Theorem 10. *The learner terminates and returns* G_L.

We close this section with a discussion of the learner's complexity. To infer the canonical ODGP $G_L = (\Sigma, N, I, P)$ for L, the learner 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 learner is thus polynomial in $|G_L| = |N| + |P|$ and the combined size of the counterexamples provided by the teacher.

Acknowledgments. We thank the anonymous reviewers for their careful reading of our manuscript and their helpful comments.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**, 87–106 (1987)
2. Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., Schneider, N.: Abstract meaning representation for sembanking. In: 7th Linguistic Annotation Workshop & Interoperability with Discourse, Sofia, Bulgaria (2013)
3. Björklund, H., Björklund, J., Ericson, P.: On the regularity and learnability of ordered DAG languages. Technical report UMINF 17.12, Umeå University (2017)
4. Björklund, H., Drewes, F., Ericson, P.: Between a rock and a hard place - uniform parsing for hyperedge replacement DAG grammars. In: 10th International Conference on Language and Automata Theory and Applications, Prague, Czech Republic, pp. 521–532 (2016)
5. Björklund, J., Fernau, H., Kasprzik, A.: Polynomial inference of universal automata from membership and equivalence queries. Inf. Comput. **246**, 3–19 (2016)
6. Chiang, D., Andreas, J., Bauer, D., Hermann, K.M., Jones, B., Knight, K.: Parsing graphs with hyperedge replacement grammars. In: 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013), Sofia, Bulgaria, pp. 924–932 (2013)
7. Drewes, F., Björklund, J., Maletti, A.: MAT learners for tree series: an abstract data type and two realizations. Acta Informatica **48**(3), 165 (2011)
8. Drewes, F., Högberg, J.: Query learning of regular tree languages: how to avoid dead states. Theory Comput. Syst. **40**(2), 163–185 (2007)
9. Drewes, F., Kreowski, H.-J., Habel, A.: Hyperedge replacement graph grammars. In: Rozenberg, G. (ed.), Handbook of Graph Grammars, vol. 1, pp. 95–162. World Scientific (1997)
10. Drewes, F., Vogler, H.: Learning deterministically recognizable tree series. J. Automata, Lang. Comb **12**(3), 332–354 (2007)
11. Hara, S., Shoudai, T.: Polynomial time MAT learning of c-deterministic regular formal graph systems. In: International Conference on Advanced Applied Informatics (IIAI AAI 2014), Kitakyushu, Japan, pp. 204–211 (2014)

12. Högberg, J.: A randomised inference algorithm for regular tree languages. Nat. Lang. Eng. **17**(02), 203–219 (2011)
13. Kozen, D.: On the Myhill-Nerode theorem for trees. Bull. EATCS **47**, 170–173 (1992)
14. Maletti, A.: Learning deterministically recognizable tree series — revisited. In: Bozapalidis, S., Rahonis, G. (eds.) CAI 2007. LNCS, vol. 4728, pp. 218–235. Springer, Heidelberg (2007). doi:10.1007/978-3-540-75414-5_14
15. Okada, R., Matsumoto, S., Uchida, T., Suzuki, Y., Shoudai, T.: Exact learning of finite unions of graph patterns from queries. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 298–312. Springer, Heidelberg (2007). doi:10.1007/978-3-540-75225-7_25
16. Rozenberg, G., Welzl, E.: Boundary NLC graph grammars-basic definitions, normal forms, and complexity. Inf. Control **69**(1–3), 136–167 (1986)
17. Sakakibara, Y.: Learning context-free grammars from structural data in polynomial time. Theor. Comput. Sci. **76**(2–3), 223–242 (1990)
18. Shirakawa, H., Yokomori, T.: Polynomial-time MAT learning of c-deterministic context-free grammars. Trans. Inf. Process. Soc. Jpn. **34**, 380–390 (1993)