Minimisation and Characterisation of Order-Preserving DAG Grammars

Henrik Björklund[a], Johanna Björklund[a], Petter Ericson[a]

[a] *Department of Computing Science, Umeå University, Sweden*

Order-preserving DAG grammars (OPDGs) is a formalism for processing semantic information in natural languages [5, 4]. OPDGs are sufficiently expressive to model abstract meaning representations, a graph-based form of semantic representation in which nodes encode objects and edges relations. At the same time, they allow for efficient parsing in the uniform setting, where both the grammar and subject graph are taken as part of the input.

In this article, we introduce an initial algebra semantic for OPDGs, which allows us to view them as regular tree grammars. This makes it possible to transfer a number of results from that domain to OPDGs, both in the unweighted and the weighted case. In particular, we show that deterministic OPDGs can be minimised efficiently, and that they are learnable in the so-called MAT setting. To conclude, we show that the languages generated by OPDGs are MSO-definable.

1. Introduction

Order-Preserving DAG Grammars (OPDGs) [5] is a subclass of Hyper-Edge Replacement Grammars (HRGs) [12], motivated by the need to model semantic information in natural-language processing. In OPDGs, the basic units of computation are *directed hyperedges*, the generalisation of regular directed edges that comes from permitting any finite number of target vertices. The left-hand side of a production rule is a single k-targeted hyperedge labelled by a nonterminal symbol, and the right-hand side is a graph with $k+1$ marked vertices. The generation process starts out from an initial graph in which the edges are labelled with nonterminals or terminals. It then iteratively replaces nonterminal edges by larger graph fragments, until only terminal edges remain. The replacement step involves a simple form of graph concatenation, illustrated in Figure 1.

To ensure efficient parsing, the graphs that appear as right-hand sides in OPDG productions must be on one of three allowed forms, illustrated in Figure 2. As a result, the generated graphs are acyclic, rooted, and have a natural order on their nodes. This is restrictive compared to HRGs in general, but sufficiently expressive to model semantic representations such as abstract meaning representations [2]. Moreover, this normal form places parsing in $O\big(n^2 + nm\big)$, where m and n are the sizes of the grammar and the input graph, respectively. For full HRGs, parsing is NP-complete even in the non-uniform case, when the grammar is fixed and only the graph is considered as input; see, for example, [12]. In [5], it is shown that even small relaxations of the restrictions on the right-hand sides lead to NP-complete parsing as well.

In [4], we provided an algebraic representation of the languages generated by OPDGs. This allowed us to state and prove a Myhill-Nerode theorem for order-preserving DAG grammars, and in doing so also provide a canonical form and an Angluin-style MAT learning algorithm. In the present work, we generalise these results to the weighted case. This is done by providing an initial algebra semantic for OPDGs, which allows us to transfer a number

of results from the tree case. We also introduce the notion of bottom-up determinism for OPDGs and provide an efficient minimisation algorithm for weighted OPDGs.

A further area of study regarding graph grammars is their relation to logic. In particular, the relation between Monadic Second-Order (MSO) logic on graphs and HRG is an active research topic, with recent results [14] exploring Regular Graph Grammars, a formalism that is both a subclass of HRL and MSO definable. We show that OPDGs occupy a similar position by proving that for every OPDG, we can construct an MSO formula that defines the same graph language.

Both the regular graph grammars of Gilroy et al. [14] and the grammars proposed by Chiang et al. [10] are variants of HRGs that are potential candidates for modelling natural language semantic data. Unlike OPDGs, however, none of these models allow for polynomial time parsing. Further related work include efforts on the generalisation of OPDGs to cover restricted types of cyclic graphs [6]. Additionally, the Regular DAG Automata proposed by Chiang et al. [11] is a recent graph formalism, also studied in, e.g., [8, 3], intended for the same applications as the present work. It shares some desirable properties with OPDGs, though not polynomial time parsing.

2. Preliminaries

Sets, sequences and numbers. The set of non-negative integers is denoted by $\mathbb{N}$. For $n \in \mathbb{N}$, $[n]$ abbreviates $\{1, \ldots, n\}$. In particular, $[0] = \emptyset$. 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 $\equiv$. For $s \in S$, $[s]_\equiv$ is the equivalence class of s with respect to $\equiv$.

Let S and T be sets. The set of all bijective functions from S to T is denoted $\mathrm{biject}(S, T)$. Note that $\mathrm{biject}(S, T) = \emptyset$ unless $|S| = |T|$.

Let $S^{\circledast}$ be the set of non-repeating sequences of elements of S. We refer to the ith member of a sequence s as s_i. Given a sequence s, we write $[s]$ for the set of elements of s. Given a partial order $\preceq$ on S, the sequence $s_1 \cdots s_k \in S^{\circledast}$ respects $\preceq$ if $s_i \preceq s_j$ implies $i \leq j$. We write $S^{\oplus}$ for $S^{\circledast} \setminus \{\lambda\}$ where λ denotes the empty sequence.

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

The set $\mathbb{T}_\Sigma$ of *trees* over the ranked alphabet Σ is defined inductively as follows:

- Every symbol $f \in \Sigma$ of rank 0 is a tree.
- Every *top-concatenation* $f[t_1, \ldots, t_k]$ of a symbol $f \in \Sigma$ of rank k with trees $t_1 \ldots t_k \in \mathbb{T}_\Sigma$ is a tree.

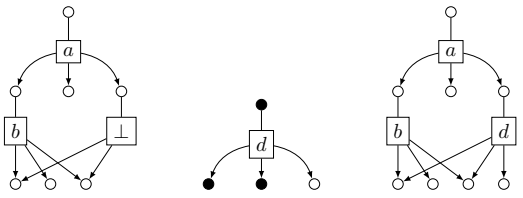

Figure 1: A graph context c, a graph g, and the substitution of g into c. Filled nodes indicate the marking of g.

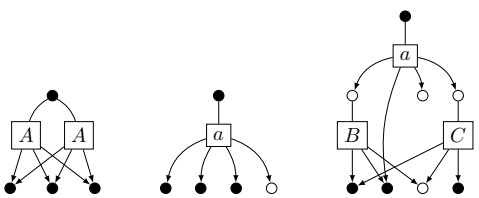

Figure 2: Example right-hand sides.

From here on, let X be a ranked alphabet containing only 0-ranked symbols, called variables, disjoint from every other alphabet discussed here. The set $\mathbb{T}_\Sigma(X)$ is the set of trees over $\Sigma \cup X$. A *tree language* is a subset of $\mathbb{T}_\Sigma$.

A *context over* Σ is a tree in $\mathbb{T}_\Sigma(X)$ containing exactly one occurrence of a symbol in X. The set of contexts over Σ is written $\mathbb{C}_\Sigma$. The substitution of $t \in \mathbb{T}_\Sigma$ into $c \in \mathbb{C}_\Sigma(X)$ is $c[\![t]\!] = c[x \leftarrow t]$ for x the single symbol from X. The tree t is a *subtree* of $s \in \mathbb{T}_\Sigma$ if there is a $c \in \mathbb{C}_\Sigma$, such that $s = c[\![t]\!]$. If t is a tree and v a position in t, we write t/v for the subtree of t rooted at v.

Typed alphabets and graphs. A *typed ranked alphabet* is a tuple (Σ, rk, tp), where (Σ, rk) is a ranked alphabet, and $tp : \Sigma \to \mathbb{N} \times \mathbb{N}^*$ assigns a type $tp(a) \in \mathbb{N} \times \mathbb{N}^{rk(a)}$ to every symbol $a \in \Sigma$. For $tp(a) = (o, i)$, where $o \in \mathbb{N}$ and $i \in \mathbb{N}^*$, we call o the *output type* and i the sequence of *argument types*, respectively, and write $otp(a) = o$, $atp(a) = i$.

Definition 2.1 (hypergraph)**.** *A directed, edge-labeled, marked* hypergraph *over a ranked alphabet* Σ *is a tuple* $g = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext})$ *with the following components:*

- V *and* E *are disjoint finite sets of* nodes *and* edges*, respectively.*
- *The* attachment $\mathrm{att} : E \to V^\oplus$ *assigns a sequence of nodes to each hyperedge. For* $\mathrm{att}(e) = vw$ *with* $v \in V$ *and* $w \in V^\circledast$*, we call* v *the* source *and* w *the sequence of* targets*, respectively, and write* $\mathrm{src}(e) = v$ *and* $\mathrm{tar}(e) = w$*.*
- *The* labeling $\mathrm{lab}\colon E \to \Sigma$ *assigns a label to each edge, subject to the condition that* $\mathrm{rank}(\mathrm{lab}(e)) = |\mathrm{tar}(e)|$ *for every* $e \in E$*.*
- *The sequence* $\mathrm{ext} \in V^\oplus$ *is the sequence of* external nodes*. If* $\mathrm{ext}_G = vw$*, then the node* v *is denoted by* $\dot{g}$ *and the sequence* w *of nodes by* $g_{\bullet\bullet}$*, respectively, and we impose the additional requirement that* $\mathrm{src}(e) \notin [g_{\bullet\bullet}]$ *for all* $e \in E$*. The type* $tp(g)$ *of* g *is* $(|g_{\bullet\bullet}|, \varepsilon)$*.*

In the following, we will only deal with the directed, edge-labeled, marked hypergraphs from Definition 2.1, and will therefore simply call them *graphs*.

A *path* in g is a finite and possibly empty sequence $\rho = e_1 e_2 \cdots e_k$ 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 ρ is k, and ρ is a *cycle* if $\mathrm{src}(e_1)$ appears in $\mathrm{tar}(e_k)$. 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 $e_1 \cdots e_k$ in g such that $u = \mathrm{src}(e_1)$ and $v \in [\mathrm{tar}(e_k)]$. An edge e' is a *descendant edge* of an edge e if there is a path $e_1 \cdots e_k$ in g such that $e_1 = e$ and $e_k = e'$. An edge or node is an *ancestor* of its descendants. The *in-degree* and *out-degree* of a node $u \in V$ is $|\{e \in E \mid u \in [\mathrm{tar}(e)]\}|$ and $|\{e \in E \mid u = \mathrm{src}(e)\}|$, 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 $\mathrm{root}(g)$ for the root node.

If A is a nonterminal of rank k, we write $A^\bullet$ for the graph consisting of a single edge, labeled A, with its $k+1$ attached nodes, which are all external.

For nodes u and v of a DAG $g = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext})$, a node or edge x is a *common ancestor* of u and v if it is an ancestor of both. It is a *closest common ancestor* if there is no descendant of x that is a common ancestor of u and v. A closest common ancestor edge e orders u before v if e's ith target is an ancestor of u, and for all j such that e's jth target is an ancestor of v, $i < j$. The partial order $\preceq_g$ on the leaves of a graph g is, if defined, the reflexive and transitive closure of the relation $\mathrm{before}(u, v)$, which holds if u, v have at least one closest common ancestor edge and if all such edges order u before v.

For a node u of a marked DAG $g = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext})$, the *sub-DAG rooted at* u is the DAG $g{\downarrow_u}$ induced by the descendants of u. Thus $g{\downarrow_u} = (U, E', \mathrm{att}', lab', \mathrm{ext}')$ where U is

the set of all descendant nodes of u, $E' = \{e \in E \mid \mathrm{src}(e) \in U\}$, and att', and lab' are the restrictions of att and lab to E'. A leaf v of $g{\downarrow_u}$ is *reentrant* in regards to u if there exists an edge $e \in E \setminus E'$ such that v occurs in $\mathrm{tar}(e)$ or in ext. We define ext' to be the sequence starting with u, and continuing with the reentrant nodes of u, ordered by $\preceq_g$, if defined. If $\preceq_g$ is not defined, we let ext' consist only of u. We note that $\preceq_{g{\downarrow_u}}$ is defined and is a subset of $\preceq_g$, if $\preceq_g$ is defined. We also note that if $y \in g{\downarrow_x} \setminus \mathrm{ext}_{g{\downarrow_x}}$ then $g{\downarrow_x}{\downarrow_y} = g{\downarrow_y}$. For proofs of these properties, see [5, 4]. For both nodes and edges x, the set of reentrant leaves of x in the graph g is denoted $\mathrm{reent}_g(x)$.

For an edge e we write $g{\downarrow_e}$ for the subgraph induced by $\mathrm{src}(e)$, $\mathrm{tar}(e)$, and all descendants of nodes in $\mathrm{tar}(e)$, with the same reasoning as above on the definition of ext'. This is distinct from $g{\downarrow_{\mathrm{src}(e)}}$ if and only if $\mathrm{src}(e)$ has out-degree greater than 1.

Let $g = (V_g, E_g, \mathrm{att}_g, \mathrm{lab}_g, \mathrm{ext}_g)$ and $h = (V_h, E_h, \mathrm{att}_h, \mathrm{lab}_h, \mathrm{ext}_h)$ be DAGs. We say that g and h are *isomorphic*, and write $g \approx h$, if there are two bijective functions $f_V : V_g \to V_h$ and $f_E : E_g \to E_h$ such that $\mathrm{att}_h \circ f_E = f_V \circ \mathrm{att}_g$, $\mathrm{lab}_h \circ f_E = \mathrm{lab}_g$, and $\mathrm{ext}_H = f_V(\mathrm{ext}_G)$.

For graphs g, h, f and an edge $e \in E_h$ with $|\mathrm{tar}_h(e)| = |f_{\bullet\bullet}|$, we call $g = h[\![e : f]\!]$ the *graph substitution* of e by f in h, if

- $E_g = E_h \setminus \{e\} \cup E_f$
- $V_g = V_h \cup V_f$
- $\mathrm{ext}_g = \mathrm{ext}_h$

and $\mathrm{att}_g(e') = \mathrm{att}_f(e'), \mathrm{lab}_g(e') = \mathrm{lab}_f(e')$ for $e' \in E_f$, and $\mathrm{att}_g(e') = \mathrm{att}_h(e'), \mathrm{lab}_g(e') = \mathrm{lab}_h(e')$ for $e' \in E_h \setminus \{e\}$. We require that $\mathrm{att}_h(e) = \mathrm{ext}_f = V_f \cap V_h$. Note that we can always choose isomorphic copies of f and h such that this is the case.

For $e, e' \in E_h$, $g = h[\![e : f]\!]$ and $g' = g[\![e' : f']\!]$, we write $g' = h[\![e : f, e' : f']\!]$, and extend this notation to any number of edges in h.

3. Well-ordered DAGs

In this section, we define a universe of *well-ordered DAGs* and discuss formalisms for expressing subsets of this universe, i.e., well-ordered DAG languages (WODLs).

Well-ordered DAGs were initially introduced as the class of graphs recognised by *order-preserving DAG grammars*[1] (OPDG) [5]. Some further properties of OPDGs are studied in [4]. Intuitively, every DAG generated by an ODPG has a partial order on its node set. This order is easily decidable from the structure of the DAG, and simplifies several processing tasks, most notably parsing.

3.1. Order-preserving DAG algebras

Well-ordered DAGs can be inductively assembled using *concatenation operations*, analogously to the step-wise construction of strings or trees through the concatenation of symbols from an alphabet. In the string case, each symbol is a string, and concatenating a string with a symbol yields a new string. In the tree case, each rank-0 symbol is a tree, and top concatenating k trees with a rank-k symbol yields a new tree.

In our domain of well-ordered DAGs, every concatenation operation is assigned a *type* that reflects the structure of the graphs it takes as input and the graph it produces as output. The operations are based on *concatenation schemata*, which also have types. Concatenation schemata are special kinds of DAGs, where some edges are *place-holders* and carry no label.

[1] In [5], the grammars were called "restricted DAG grammars", but in [4], the more descriptive name "order-preserving DAG grammars" was substituted.

Definition 3.1. *Let Σ be a ranked alphabet. A DAG f is a* concatenation schema *over Σ if either of the two following conditions hold.*

1. *f contains exactly two edges, both of rank k, both place-holders, and both have the same source and the same targets, in the same order. All nodes of f are external and connected to the two edges. We call such a graph a* clone. *Its type is (k, kk).*
2. *f has height at most two and satisfies the following.*
   - *No node has an out-degree larger than one.*
   - *There is a single root with a single edge attached to it. This edge is labeled by a terminal from Σ.*
   - *All other edges are place-holders.*
   - *Only leaves have in-degrees larger than one.*
   - *All targets of place-holder edges have in-degree larger than one or are external.*
   - *The ordering $\preceq_f$ is total on the leaves and is respected by $f_{\cdot\cdot}$.*

For a concatenation schema that is not a clone, there is a natural ordering on the place-holder edges. This is because there is a unique edge connected to the root, all place-holders have targets of this edge as sources, and no two place-holders share a source. Thus, if f has ℓ place-holders, we can refer to them as $f_1, \ldots, f_\ell$. In the case of clone rules, the two edges are isomorphic, and we can simply pick any ordering. The number ℓ of place-holder edges in f is the *arity* of f, denoted $arity(f)$. The type of such concatenation schema is $(|f_{\cdot\cdot}|, |\mathrm{tar}_f(f_1)||\mathrm{tar}_f(f_2)| \cdots |\mathrm{tar}_f(f_l)|)$.

Each concatenation schema f gives rise to a concatenation operator concat_f of arity $arity(f)$ as described in the following definition.

Definition 3.2. *Let f be a concatenation schema of type $(o, a_1 \ldots a_\ell)$, and $f_1 \ldots f_\ell$ its place-holder edges. The concatenation operation $\mathrm{concat}_f(g_1, \ldots, g_\ell)$ is defined for well-ordered DAGs $g_1 \ldots g_\ell$ where $otp(g_i) = a_i$ for all $i \in [\ell]$. It yields the graph $g = f[\![f_1 : g_1, \ldots, f_\ell : g_\ell]\!]$.*

If f is a concatenation schema over Σ, we call concat_f a *concatenation operator* over Σ. The set of all such operators is denoted concat_Σ.

A special case of concatenation schemata is the one where the graph f has height one, but is not a clone. In this case, f consists of a single terminal edge. The external nodes include the source and any subsequence of the targets.

Definition 3.3. *Let Σ be an alphabet. The* well-ordered *DAGs over Σ, denoted $\mathcal{A}_\Sigma$, is the set of graphs that can be constructed using operations from* concat_Σ.

3.2. Order-preserving DAG grammars

Order-preserving DAG grammars (OPDGs) produce well-ordered DAGs [5, 4]. In other words, every language produced by an OPDG over Σ is a subset of $\mathcal{A}_\Sigma$. When we next recall the definition, we restrict ourselves to grammars on a particular normal form. As shown in [5], every OPDG can be rewritten into one on this normal form in polynomial time.

If Σ is a ranked alphabet of terminals and N a ranked alphabet of non-terminals, we call a graph f an *N-instantiated concatenation schema over Σ* if f can be obtained from a concatenation schema over Σ by assigning each place-holder a nonterminal from N of appropriate rank.

An order-preserving DAG grammar (OPDG) is a structure $G = (\Sigma, N, P, S)$ where

- Σ is the ranked alphabet of terminal symbols,
- N is the ranked alphabet of nonterminal symbols,
- P is the set of production rules, described below, and
- $S \in N$ is the starting nonterminal

A production rule has the form $A \to f$, where A is a nonterminal and f an N-instantiated concatenation schema over Σ. We require that $rk(A) = otp(f)$ and that if f is a clone, then both its edges are labelled A.

A derivation step $g \to_p h$ for a production $A \to f = p \in P$ consists of replacing an edge marked with A in g with f, producing h. We write $\to_G$ for a derivation step using any of the rules of P, and $\to_G^*$ for the reflexive and transitive closure. We write $\mathcal{L}(G)$, indicating the *language* of the grammar G for the set of terminal graphs g such that $S^\bullet \to_G^* g$. If $g \to_{p_1} h' \to_{p_2} h$ and $g \to_{p_2} h'' \to_{p_1} h$, the two derivation steps are *independent*. Two derivations $d_1 = S^\bullet \to_G^* g$ and $d_2 = S^\bullet \to_G^* g$ are *distinct* if they cannot be made equal by reordering of independent derivation steps. Note that our view of derivation is essentially a linearised version of context-free derivation trees, where rule applications in different subtrees are independent, and distinct derivations have derivation trees that are distinguishable. However, the presence of cloning rules makes matters more involved, and Section 4 explains how these are handled.

An OPDG is *bottom-up deterministic* if, for each rule $A \to f$, there is no rule $B \to g$ such that $g \approx f$ and $B \neq A$. Informally, there are no two nonterminals that lead to the same right-hand side.

We conclude this section by sketching a parsing algorithm for OPDGs; for a detailed presentation, formal proofs, and complexity results, see [5]. In short, we can, without looking at the grammar, determine a number of useful properties of the input graph – in particular that there is an appropriate ordering of the leaves – and identify the graphs $g{\downarrow_x}$ for all nodes and edges x. Afterwards, assuming that the grammar is on normal form, we parse the graph bottom-up, marking each non-leaf node or edge x with the the nonterminals that could produce $g{\downarrow_x}$, and checking at each step which right-hand sides match. Finally, we check that the initial nonterminal is in the set of nonterminals that marks the root node.

3.3. Well-ordered DAG series

A commutative semiring is a tuple $\mathcal{C} = (C, +, \cdot, 0, 1)$ such that both $(C, \cdot, 1)$ and $(C, +, 0)$ are commutative monoids, $\cdot$ distributes over $+$, and $0 \cdot c = c \cdot 0 = 0$ for all $c \in C$. If, for every semiring element $c \in C$ except 0, there exists an element $c^{-1} \in C$ such that $c \cdot c^{-1} = 1$, then $\mathcal{C}$ is a commutative semifield. If $\mathcal{C}$ is a semifield and there also exists, for every $c \in C$, an element $-c \in C$ such that $c + (-c) = 0$, then $\mathcal{C}$ is a commutative field. The semiring is *zero-sum free* if there does not exists elements $a, b \in C \setminus \{0\}$ such that $a + b = 0$. It is *zero-divisor free* if there does not exists elements $a, b \in C \setminus \{0\}$ such that $a \cdot b = 0$.

By equipping OPDG rules with weights from a semiring, we can model weighted well-ordered DAG languages, in other words, well-ordered DAG series (WODS). A weighted OPDG (WOPDG) over commutative semiring $\mathcal{C}$ is a structure $G = (\Sigma, N, P, S, w)$, where (Σ, N, P, S) is an OPDG, and $w : P \to \mathcal{C}$ is the weight function.

The A-weight of a derivation $A^\bullet \to_{p_0} g_0 \to_{p_1} \ldots \to_{p_l} g$ is

$$\prod_i w(p_i) \ ,$$

and the weight of a graph is the sum of the weights of all distinct S-derivations that generate it. We generally call S-derivations derivations. The weight distribution thus defined is the

WODS $\mathcal{S}(G) : \mathcal{A}_\Sigma \to \mathcal{C}$. This means that if there is no (S-)derivation of g in G, then $\mathcal{S}(G)(g) = 0$. The *support* of a WOPDG G is the set of graphs $\mathit{support}(G) = \{g \mid \mathcal{S}(G)(g) \neq 0\}$. Note that the support of a WOPDG is a subset of the language of the underlying OPDG. If no rule is assigned weight 0, and the semiring is zero-sum and zero-divisor free, then the support of the WOPDG and the language of the underlying OPDG coincide. A WOPDG is *deterministic* if its underlying OPDG is. A WOPDG is *bottom-up deterministic* if, for every nonterminal A, there is at most one production rule $A \to g$ that has non-zero weight.

4. Initial algebra semantics

In this section, we establish a link between well-ordered DAG series and tree series, from which several results relating to minimisation (Section 5) and learnability (Section 6) immediately follow.

Definition 4.1 (Terms over concat_Σ)**.** *We associate with the set of concatenation operators* concat_Σ *the typed ranked alphabet* $\mathrm{concat}'_\Sigma = \{\hat{f} \mid f \in \mathrm{concat}_\Sigma\}$*, where* $rk(\hat{f})$ *equals the arity of* f *and* $tp(\hat{f})$ *is* $tp(f)$*.*

The terms over concat_Σ *is the set of trees* $\mathcal{T}_{\mathrm{concat}_\Sigma} \subset \mathbb{T}_{\mathrm{concat}'_\Sigma}$ *that are* type-matched *in the sense that for each subterm* $\hat{f}[t_1, \ldots, t_l]$*, the* i*th element of the argument type of* $\hat{f}$ *must match the output type of the root symbol of* t_i*.*

Let X *be the (infinite) typed ranked alphabet* $\{x_k \mid k \in \mathbb{N}\}$*, such that* $rk(x_k) = 0$ *and* $tp(x_k) = (k, \varepsilon)$ *for every* $k \in \mathbb{N}$*. Analogously to the tree case, the set* $\mathcal{T}_{\mathrm{concat}_\Sigma}(X)$ *is the set of type-matched trees over* $\mathrm{concat}'_\Sigma \cup X$

Terms over concat_Σ can be evaluated to yield graphs in $\mathcal{A}_\Sigma$. The construction is as expected. Evaluating a symbol $x_k \in X$ yields a placeholder edge with k targets.

Definition 4.2 (Term evaluation)**.** *The evaluation function* $\mathit{eval} : \mathcal{T}_{\mathrm{concat}_\Sigma}(X) \to \mathcal{A}_\Sigma$ *is defined as follows: For every* $x_k \in X$*,* $\mathrm{eval}(x_k)$ *is a single placeholder edge with exactly* k *targets, all external. For every* $t = \hat{f}[t_1, \ldots, t_k] \in \mathcal{T}_{\mathrm{concat}_\Sigma}(X) \setminus X$*,* $\mathrm{eval}(t) = f(\mathrm{eval}(t_1), \ldots, \mathrm{eval}(t_k))$*.*

The clones in concat_Σ need some special care, since their arguments have no inherent order. In what follows, we will write Cl to denote the set $\{\hat{f} \mid f \in \mathrm{concat}_\Sigma \wedge f \text{ is a clone}\}$ of all clones in concat_Σ.

Definition 4.3 (Top clone positions)**.** *Let* $t \in \mathcal{T}_{\mathrm{concat}_\Sigma}$*. The* top clone positions *of* t *is the set of positions*

$$\mathrm{cln}(t) = \{v \in \mathrm{pos}(t) \mid \textit{ there is a path from } \mathrm{root}(t) \textit{ to } v \textit{ labelled } (Cl)^+(\mathrm{concat}'_\Sigma \setminus \{Cl\})\} \ .$$

The set of subtrees that attaches to the top clone positions in a term t can be freely permuted according to some bijection onto these positions, without affecting the value of t with respect to eval. This invariance induces an equivalence relation on $\mathcal{T}_{\mathrm{concat}_\Sigma}$.

Definition 4.4 (The relation $\sim$)**.** *The binary relation* $\sim$ *on* $\mathcal{T}_{\mathrm{concat}_\Sigma}$ *is defined as follows, for every* $t = \hat{f}[t_1, \ldots, t_k], s = \hat{g}[s_1, \ldots, s_n] \in \mathcal{T}_{\mathrm{concat}_\Sigma}$*:*

$$t \sim s \iff \hat{f} = \hat{g} \textit{ and } \begin{cases} \exists \varphi \in \mathrm{biject}(\mathrm{cln}(t), \mathrm{cln}(s)) : \forall v \in \mathrm{cln}(t) : t/v \sim s/\varphi(v) & \textit{if } \hat{f} \in Cl \\ t_i \sim s_i, \forall i \in [k] & \textit{otherwise.} \end{cases}$$

It is straight-forward to show that $\sim$ *is an equivalence relation on* $\mathcal{T}_{\mathrm{concat}_\Sigma}$*.*

Lemma 4.5. *For every $g \in \mathcal{A}_\Sigma$, there is a tree $t \in \mathcal{T}_{\mathrm{concat}_\Sigma}$ such that $g = \mathrm{eval}(t)$, and t is unique modulo $\sim$.*

Proof. The proof is by induction on the size of $g{\downarrow_x}$, where $x \in V \cup E$. For the base case, assume that x is an edge and $g{\downarrow_x}$ has height one and thus consists of a single edge. There must then be a constant operation $f \in \mathrm{concat}_\Sigma$, such that $g{\downarrow_x} = f = \mathrm{eval}(\hat{f})$.

For the inductive case, first assume that $x \in V$. If x only has a single outgoing edge e, then $g{\downarrow_x} = g{\downarrow_e}$ and, since the inductive case for edges is handled below, we are done.

Assume that x has outgoing edges $\{e_1, \ldots, e_\ell\}$. Then $g{\downarrow_x}$ must be the result of a clone operator f applied to two smaller graphs h and h', which by the induction hypothesis can be uniquely represented (modulo $\sim$) as concatenation terms t and t', respectively. It follows that $g{\downarrow_x}$ can uniquely represented as $\hat{f}[t, t']$, as $\hat{f}[t, t'] \sim \hat{f}[t', t]$.

Next, assume that $x \in E$. Let $\mathrm{tar}(x) = v_1 \cdots v_k$ and let $v_{i_1} \cdots v_{i_\ell}$ be the non-leaf subsequence of $\mathrm{tar}(x)$. For each $j \in [\ell]$, by inductive assumption, the subgraph $g{\downarrow_{v_{i_j}}}$ is represented by a term t_{i_j} such that $\mathrm{eval}(t_{i_j}) = g{\downarrow_{v_{i_j}}}$, so $g{\downarrow_x}$ is represented by a term $\hat{f}[t_{i_j}, \ldots, t_{i_l}]$, for some suitable basic concatenation operator $f \in \mathrm{concat}_\Sigma$. □

Every ranked alphabet suggests a corresponding set of top concatenation operators.

Definition 4.6 (Top concatenation)**.** *Let Γ be a ranked alphabet. We denote by TOP_Γ the Γ-indexed family of* top-concatenations *$(c_\gamma)_{\gamma \in \Gamma}$, where for every $\gamma \in \Gamma$, c_γ is the top-concatenation with respect to γ.*

We extend the notion of top-concatenation to the domain $\mathcal{T}_{\mathrm{concat}_\Sigma}/\!\sim$ by letting

$$c_{\hat{f}}([t_1]_\sim, \ldots, [t_{rk(f)}]_\sim) \mapsto [\hat{f}[t_1, \ldots, t_{rk(f)}]]_\sim \ ,$$

for every $\hat{f} \in \mathrm{concat}'_\Sigma$ and $t_1, \ldots, t_{rk(f)} \in \mathcal{T}_{\mathrm{concat}_\Sigma}$. The function is well-defined, because for every $\hat{f} \in \mathrm{concat}'_\Sigma$, top concatenation with respect to $\hat{f}$ is a congruence with respect to $\sim$.

From Lemma 4.5 it follows that $\mathrm{eval} : \mathcal{A}_\Sigma \to \mathcal{T}_{\mathrm{concat}_\Sigma}/\!\sim$ is a bijection, and this gives us Theorem 4.7.

Theorem 4.7. *The algebras $(\mathrm{concat}_\Sigma, \mathcal{A}_\Sigma)$ and $(\mathsf{TOP}_{\mathrm{concat}'_\Sigma}, \mathcal{T}_{\mathrm{concat}_\Sigma}/\!\sim)$ are isomorphic.*

Theorem 4.7 suggests an alternative definition of ODPG semantics.

Definition 4.8. *Every WOPDG G over the alphabet Σ is a weighted tree grammar (wtg) over the typed ranked alphabet concat'_Σ. We denote by $\mathcal{S}_t(G)$ the tree series generated by G when viewed as a wtg.*

Definition 4.9 (Inital algebra semantics)**.** *Let $G = (\Sigma, N, P, S, w)$ be a WOPDG. The* initial algebra semantics *of G is the tree series $\mathcal{S}'(G) = \{(\mathrm{eval}(t), \mathcal{S}_t(G)(t)) \mid t \in \mathcal{S}_t(G)\}$.*

Observation 4.10. *For every WOPDG G, $\mathcal{S}(G) = \mathcal{S}'(G)$.*

A WOPDG is thus essentially a weighted tree grammar together with an evaluation function. This connection to tree series allows us to transfer a host of results.

5. Minimisation

In this section, we consider the minimisation problem for deterministic WOPDGs. We start by showing that if a grammar is bottom-up deterministic, then each graph in its support has a unique derivation tree. This is immediately implied by the following lemma.

Lemma 5.1. *Let $G = (\Sigma, N, P, S, w)$ be a WOPDG. Then G is bottom-up deterministic if and only if the following property holds. For every graph $g = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext})$ in $\mathit{support}(G)$ and every $x \in V \cup E$ that is not a leaf node, there is a unique nonterminal $A \in N$ such that $A^\bullet \to^*_G g{\downarrow_x}$.*

Proof. The 'if' direction is immediate: if there were two distinct nonterminals that appeared as left-hand sides of rules with isomorphic right-hand sides, then there would be some graph that could be derived from both of them.

The 'only if' direction is proved by induction on, primarily, the height of $g{\downarrow_x}$, and secondarily, the outdegree of the root of $g{\downarrow_x}$. Since x is not a leaf, the base case is that x is an edge and the height of $g{\downarrow_x}$ is 1. Thus $g{\downarrow_x}$ is a single edge, together with its incident nodes. This means that for G to generate g, the subgraph $g{\downarrow_x}$ must be generated by a rule $A \to f$, where f is isomorphic to $g{\downarrow_x}$, for some $A \in N$. Since there cannot be two distinct nonterminals that generate graphs isomorphic to $g{\downarrow_x}$ and our grammars have no unit rules, A is the unique nonterminal such that $A^\bullet \to^*_G g{\downarrow_x}$.

For the inductive case, first assume that $x \in V$. If x has only a single outgoing edge e, then $g{\downarrow_x} = g{\downarrow_e}$ and, as the inductive case for edges are handled in the next paragraph, we are done. If, on the other hand, x has several outgoing edges $e_1, \ldots, e_\ell$, then we reason as follows. The only way for a node in a graph generated by an OPDG to have outdegree larger than one is if at some point in the derivation process, x was the source of a single nonterminal edge that was subsequently cloned. This means that there must be some nonterminal A such that each graph $g{\downarrow_{e_1}}, \ldots, g{\downarrow_{e_\ell}}$ can be generated by A and there is a clone rule in P for A. Furthermore, by inductive assumption, A is the unique nonterminal with this property. Thus A is also the unique nonterminal from which $g{\downarrow_x}$ can be derived.

Assume, finally, that x is an edge. Let $v_1, \ldots, v_\ell$ be the non-leaf targets of x. By inductive assumption, for each $i \in [\ell]$, there is a unique nonterminal A_i that can generate $g{\downarrow_{v_i}}$. Then $g{\downarrow_x}$ must have been generated starting with the application of a rule $A \to f$, where f is isomorphic to the graph obtained from $g{\downarrow_x}$ by replacing each graph $g{\downarrow_{v_i}}$ by a single edge labeled A_i, attached to the sequence of leaf nodes of $g{\downarrow_x}$ that are external for $g{\downarrow_{v_i}}$. Since no distinct nonterminals can appear in rules with isomorphic right-hand sides, A must be the unique such nonterminal. □

Another way of stating Lemma 5.1 is the following. For each $A \in N$, let G_A be the grammar obtained from G by replacing S with A as starting symbol. Then G is bottom-up deterministic if and only if, for each pair A_1 and A_2 of nonterminals from N, $\mathit{support}(G_{A_1}) \cap \mathit{support}(G_{A_2}) \neq \emptyset$ implies $A_1 = A_2$. In other words, the concept of bottom-up determinism coincides with the notion of unambiguity for OPDGs, as defined in [4]. Thus we can restate one of the results from that article:

Theorem 5.2 (cnf. [4])**.** *If $\mathcal{S}$ is a series generated by some deterministic WOPDG over a commutative semifield, then there is a unique (up to isomorphism) minimal deterministic WOPDG G_L such that $\mathcal{S}(G_L) = \mathcal{S}$.*

Theorem 5.2 ensures that the *minimisation problem* for deterministic WOPDGs always has a unique solution, modulo nonterminal names. The problem is stated as follows:

Definition 5.3 (Minimization problem)**.** *Given a deterministic WOPDG G, find the unique minimal deterministic WOPDG for $\mathcal{S}(G)$.*

Rather than formulating a minimisation algorithm that solves Problem 5.3 directly, we show that the problem can be reduced to finding the unique minimal weighted deterministic regular tree grammar for $\mathcal{S}_t(G)$. For this purpose, we note that the forward or backward application of eval does not affect the nonterminal to which a tree or DAG is mapped:

Lemma 5.4. *Let G be a deterministic WOPDG. For every non-terminal A in G and every $t \in \mathcal{T}_{\mathrm{concat}'_\Sigma}$, $\mathcal{S}_t(G_A)(t) = \mathcal{S}(G_A)(\mathrm{eval}(t))$.*

In preparation for the proof of Lemma 5.6, we lift the notion of *contexts* from the tree domain to the graph domain. Intuitively, a context is a graph with a single, appropriately placed placeholder edge.

Definition 5.5 (Graph context)**.** *A* graph context *over Σ is the evaluation of a tree in $\mathcal{T}_{\mathrm{concat}_\Sigma}(X)$ with a single occurrence of a symbol x_k from X.*

This yields a graph context c of some type (m, k) with a single placeholder edge e of rank k. We can substitute a graph $f \in \mathcal{A}_\Sigma$ of appropriate type (k, ε) into c in the standard way, yielding the graph $g = c[\![e : f]\!]$ of type (m, ε). We also write this operation as $c[\![f]\!]$.

It is straightforward to show that taking a graph $g \in \mathcal{A}_\Sigma$ and replacing $g{\downarrow_x}$ for some $x \in V_g \cup E_g$ with a placeholder edge of rank $|g{\downarrow_x}_{\bullet\bullet}|$ yields a graph context in this sense.

Lemma 5.6. *Let G be a deterministic WOPDG over a commutative semifield, and H the minimal deterministic weighted tree grammar for $\mathcal{S}_t(G)$. Then H is the minimal deterministic WOPDG for $\mathcal{S}(G)$.*

Sketch. The nonterminals A and B in G are *distinguishable w.r.t.* $\mathcal{S}(G)$ if there is a graph context c and graphs $g \in \mathit{support}(\mathcal{S}(G_A))$ and $h \in \mathit{support}(\mathcal{S}(G_B))$ such that

$$\mathcal{S}(G)(c[\![g]\!]) \cdot \mathcal{S}(G_A)(g)^{-1} \neq \mathcal{S}(G)(c[\![h]\!]) \cdot \mathcal{S}(G_B)(h)^{-1} \ .$$

Similarly, the pair of nonterminals is *distinguishable w.r.t.* $\mathcal{S}_t(G)$ if there is a tree context c and trees $t \in \mathit{support}(\mathcal{S}_t(G_A))$ and $s \in \mathit{support}(\mathcal{S}_t(G_B))$ such that

$$\mathcal{S}_t(G)(c[\![t]\!]) \cdot \mathcal{S}_t(G_A)(t)^{-1} \neq \mathcal{S}_t(G)(c[\![s]\!]) \cdot \mathcal{S}_t(G_B)(s)^{-1} \ .$$

We first ensure that H is a WOPDG. Since H is minimal for $\mathcal{S}_t(G)$, and both H and G are deterministic, the nonterminals of H can be obtained by merging every set of mutually indistinguishable nonterminals w.r.t. $\mathcal{S}_t(G)$ into a single nonterminal [16]. This means in particular that every clone rule in H can be written in the form $P \to f[P, P]$, where P is an equivalence class of mutually indistinguishable nonterminals. Moreover, to see that the merge respects $\sim$, we argue as follows: From Theorem 4.7 we have that $t \sim s$ implies $\mathrm{eval}(t) = \mathrm{eval}(s)$, and by Lemma 5.4 that there is a nonterminal A such that $t, s \in \mathit{support}(\mathcal{S}_t(G_A))$, so there is a nonterminal $B \in H$ (that is the result of merging A the with indistinguishable nonterminals) such that $t, s \in \mathit{support}(\mathcal{S}_t(H_B))$. It follows that H is a valid WOPDG, and that $\sim$ is a congruence with respect to H.

It is then straight-forward to show that for every WOPDG G and nonterminals A and B in G, A and B are distinguishable w.r.t. $\mathcal{S}(G)$ if and only if they are distinguishable w.r.t. $\mathcal{S}_t(G)$, so H is minimal also for $\mathcal{S}(G)$. A witness context for the distinguishability of a pair of nonterminals in one domain, can be translated to a witness context in the other, by extending eval and eval^{-1} to tree and graph contexts in the expected way. □

Lemma 5.6 means that the minimisation result established in [7] and [16] are directly transferable to our setting. In the statement of these results, we assume a deterministic input WOPDG G over different types of semirings and let r denote the maximal number of non-terminals in the right-hand side of any production of G, m denote the size of the input grammar G, and n denote the number of nonterminals in G. A WOPDG is *all-accepting* if assigns a non-zero value to every graph in its domain.

Theorem 5.7 (cnf. Theorem 4.12 of [7])**.** *The minimisation problem for all-accepting deterministic WOPDGs over commutative fields is solvable in $O(rm \log n)$.*

Theorem 5.8 (cnf. [16])**.** *The minimisation problem for deterministic WOPDGs over commutative semifields is solvable in $O(rmn)$.*

6. MAT Learning

The relation between WODS and tree series established in Section 4 also makes results on grammatical inference for tree languages transferable to our DAG domain. Here, we focus on the Minimal Adequate Teacher (MAT) model due to Angluin [1]. The MAT model supposes two entities – a *learner* and a *teacher*. The teacher already knows the target series $\mathcal{S}$, and it is the objective of the learner to infer $\mathcal{S}$. The learner gathers information about $\mathcal{S}$ by querying the teacher: In a *coefficient query*, the learner gives the teacher a graph g, and the teacher answers with the weight $\mathcal{S}(g)$. In an *equivalence query*, the learner gives the teacher a WOPDG G. If G represents $\mathcal{S}$ correctly, then the teacher confirms the successful inference and the learning ends. If not, the teacher returns a *counterexample* – a graph g that is assigned an erroneous weight by G, i.e., that is such that $\mathcal{S}(G)(g)$ and $\mathcal{S}(g)$ differ.

Definition 6.1 (MAT learning)**.** *A MAT teacher for a WODS $\mathcal{S}$ over a semiring $\mathcal{C}$ is an oracle capable of answering two types of queries:*

- ***Coefficient queries**: Given $g \in \mathcal{A}_\Sigma$, what is $\mathcal{S}(g)$?*
- ***Equivalence queries**: Given a WOPDG G, is $\mathcal{S}(G) = \mathcal{S}$? If yes, the teacher confirms the successful inference of $\mathcal{S}$, if no, the teacher returns a counterexample, that is, a graph $g \in \mathcal{A}_\Sigma$ such that $\mathcal{S}(G)(g) \neq \mathcal{S}(g)$.*

A class of graph series is *MAT learnable* if every series in the class can be inferred within the MAT model. In general, this is true for classes for which there is a Myhill-Nerode theorem, such as recognisable string and tree series [9]. As we shall see, WODLs and WODSs over commutative semifields also meet this description. Rather than providing an explicit MAT-learning algorithm for the latter class, we show that the problem of inferring a target WODS $\mathcal{S}$ over the commutative semifield $\mathcal{C}$ can be reduced to that of inferring a regular tree series $\mathcal{S}_t$ over $\mathcal{C}$, and that $\mathcal{S}_t$ is easily derivable from $\mathcal{S}$.

Definition 6.2 (The series $\mathcal{S}_t$)**.** *Let $\mathcal{S} : \mathcal{A}_\Sigma \to \mathcal{C}$ be a WODS. The regular tree series $\mathcal{S}_t : \mathbb{T}_{\mathrm{concat}'_\Sigma} \to \mathcal{C}$ is given by*

$$\mathcal{S}_t(t) = \begin{cases} \mathcal{S}(g) & \text{if } t \in \mathit{eval}^{-1}(g) \text{ for some } g \in \mathit{support}(\mathcal{S}) \text{ , and} \\ 0 & \text{otherwise.} \end{cases}$$

In preparation for Theorem 6.3, we recall the MAT learner for tree series over commutative semifields given in [15] (there formulated for weighted tree automata, see also [13]). In the inference of a series $\mathcal{S}$, the learner gathers two sets of trees, S and T. Both are subtree-closed in the sense that if they contain a tree t, then they also contain every subtree of t. Additionally, every direct subtree of a tree in T is contained in S. The purpose of S is to collect representatives of the syntactic congruence classes with respect to $\mathcal{S}$, which are in a one-to-one correspondence with the nonterminals of the minimal wtg G_t that generates $\mathcal{S}_t$. To avoid confusion, we write $\langle t \rangle$ to express that a tree t is viewed as a nonterminal.

The learner also maintains an auxiliary set of contexts E that witness (i) that every tree in S is a subtree of some tree in $\mathit{support}(\mathcal{S}_t)$, and (ii) that the trees in S are syntactically distinct. The purpose of T is to represent production rules of the hypothesis grammar, and a tree $f[t_1, \ldots, t_k] \in T$ encodes the production rule $\langle \mathit{rep}(t) \rangle \to f(\langle t_1 \rangle, \ldots, \langle t_k \rangle)$, where $\mathit{rep}(t)$ is the unique tree in S such that $\mathit{rep}(t)$ and $f[t_1, \ldots, t_k]$ are indistinguishable with respect to the contexts in E (if no such tree exists, then $f[t_1, \ldots, t_k]$ is added to S).

From S and T, the learner synthesises a weighted tree grammar H that is passed to the teacher through an equivalence query. The wtg has the property that every tree $t \in T$ is

in $\mathit{support}(\mathcal{S}_t)(H_{\langle \mathit{rep}(t)\rangle})$, where $H_{\langle \mathit{rep}(t)\rangle}$ is the grammar obtained from H by replacing the initial nonterminal by $\langle \mathit{rep}(t)\rangle$.

The learner collects the elements of S and T by processing the teacher's counterexamples through *contradiction-backtracking* [17]. This essentially consists in step-wise simulation of the parsing of a counterexample t with respect to the current hypothesis wtg H. The learner repeatedly selects a subtree of t on the form $f[t_1, \ldots, t_k]$ for some $t_1, \ldots, t_k \in S$. If this subtree is not in T, then it is added to T, and the learner has found a new production rule. If it is in T, it is replaced in t by $\mathit{rep}(f[t_1, \ldots, t_k])$. The learner then uses a coefficient query to verify that the counterexample is still a counterexample. If it is not, then the learner has discovered that $f[t_1, \ldots, t_k]$ and $\mathit{rep}(f[t_1, \ldots, t_k])$ belong to different syntactic congruence classes, i.e., the learner has found a new nonterminal. Since the learner disagrees with the teacher about t, it is guaranteed to find at least one new transition or nonterminal by backtracking, and this guarantees that the overall inference process eventually terminates.

Theorem 6.3. *WODSs over commutative semifields are MAT learnable in polynomial time.*

Proof. We show that the problem of inferring a target WODS $\mathcal{S}$ can be reduced to that of inferring the tree series $\mathcal{S}_t$ defined above, and then applying the existing MAT-learning algorithm for trees series over semifields given in [15]. We henceforth refer to this algorithm as the learner. With this approach, the problem becomes one of finding a way to simulate a MAT teacher for $\mathcal{S}_t$ using the available MAT teacher for $\mathcal{S}$.

For coefficient questions, the simulation is easy. When the learner wants to ask a coefficient question for the tree $t \in \mathbb{T}_{\mathrm{concat}'_\Sigma}$, we first check if it is type matched. If not, we answer the learner that it has weight 0. If it is type matched, then $\mathrm{eval}(t)$ is well-defined, so we ask the teacher a coefficient question for $\mathrm{eval}(t)$, and as the answer holds equally for every $s \in \mathrm{eval}^{-1}(\mathrm{eval}(t))$, it holds in particular for t.

To simulate equivalence queries, we must ensure that the hypothesis grammar H maintained by the learner is not only a well-formed weighted tree grammar, but also a well-formed WOPDG. This requires us to argue that the following invariants hold:

- First, we require the trees produced be type-matched, and that for each rule $A \to f$, that $\mathrm{rank}(A) = \mathit{otp}(\mathit{rep}(f)) = \mathit{otp}(f)$
- Second, all clone rules in H are on the form $A \to f[A, A]$ for some nonterminal A and clone f of appropriate type.

We deal with these in order:

- Note that for every tree $s \in S$, the learning algorithm in [15] is guaranteed to have collected at least one context $c \in E$, such that $\mathrm{eval}(c[\![s]\!]) \in \mathit{support}(\mathcal{S})$, meaning s is type-matched. Moreover, each tree $t \in T$ can be freely substituted for its representative $\mathit{rep}(t) \in S$ in all of the contexts $c \in C$, so by the same reasoning, all trees in T are type-matched. That is, for each tree $f[s_1, \ldots, s_\ell] \in T$ with $s_i \in S$ for all $i \in [\ell]$, $\mathit{otp}(s_i) = \mathit{atp}(f)_i$. Finally, by the substitution of t for s in c, $\mathit{otp}(t) = \mathit{otp}(s)$.
- As described in [15], the learner is reactive in its collection of trees that represent productions, and will therefore only include clone rules in H if it has come across such in the contradiction-backtracking on some tree t on the form $c[\![f[s, s']]\!]$ that is in $\mathit{support}(\mathcal{S})$ but not in $\mathit{support}(\mathcal{S}(H))$. When this happens, the learner uses contradiction backtracking and incrementally replaces larger and larger subtrees of s and s' by trees that is in its set of nonterminal representatives S, but if it any points produces a tree $c'[\![f[r, r']]\!]$ such that exactly one of r or r' is not in the same syntactic

congruence class as s and s', then $c'[\![f[r,r']]\!]$ will fall outside of $\mathit{support}(\mathcal{S})$, and the algorithm will learn that r and r' are not syntactically equivalent.

Hence, by the time the learner has replaced s and s' with trees in S, it has to be the same tree, because it has only one representative per syntactic class, so the example will be on the form $c''[\![f[r,r]]\!]$ for some $r \in S$. Since $f[r,r]$ will then be in the set of representatives for production rules T, since it is in the same syntactic class as r, and since the grammar H produced by the learner is guaranteed to map every tree in T to the correct nonterminal in S, $f[r,r]$ will be taken to r, and thus represents a valid clone rule.

We can thus pass H to the teacher through equivalence queries, and if we are given in return a counterexample g, then any tree in $\mathrm{eval}^{-1}(g)$ is a counterexample for the learner, because H maps each tree in $\mathrm{eval}^{-1}(g)$ to the same nonterminal. □

7. Logical characterisation

Our aim in this section is to prove that OPDL is a set of MSO definable graph languages. Thus, given an OPDG $G = (\Sigma, N, P, S)$, our goal is to construct an MSO formula φ_G such that for every graph g,

$$g \in \mathcal{L}(G) \Leftrightarrow g \models \varphi_G.$$

Let r be the maximal rank of any edge appearing in G. The vocabulary we use for φ_G has the following predicate symbols:

Node $\mathrm{Node}(v)$ is true if v is a node.

Edge $\mathrm{Edge}(e)$ is true if e is an edge.

Src $\mathrm{Src}(e, v)$ is true if e is an edge and v is the source node of e.

Tar^i For every $i \in [r]$, $\mathrm{Tar}^i(e, i)$ is true if e is an edge and v is the ith target node of e.

Lab$_a$ For every $a \in \Sigma \cup N$, $\mathrm{lab}_a(e)$ is true if e is an edge and a is the label of e.

Ext $\mathrm{Ext}(v)$ is true if v is a node and v is external.

We can now define the following useful formulas:

$$\begin{aligned}
\mathrm{Tar}(e,v) &= \bigvee_{i\in[r]} \mathrm{Tar}^i(e,v) \\
\mathrm{Root}(v) &= \mathrm{Node}(v) \wedge \neg\exists e(\mathrm{Tar}(e,v)) \\
\mathrm{Leaf}(v) &= \mathrm{Node}(v) \wedge \neg\exists e(\mathrm{Src}(e,v)) \\
\mathrm{Indegree}(v) = 1 &= \exists e(\mathrm{Tar}(e,v) \wedge (\forall x(\mathrm{Tar}(x,v) \to x = e))) \\
\mathrm{Outdegree}(v) = 1 &= \exists e(\mathrm{Src}(e,v) \wedge (\forall x(\mathrm{Src}(x,v) \to x = e)))
\end{aligned}$$

We leave the somewhat tedious construction of a formula that ensures that the input structure correctly encodes a well-ordered DAG for the Appendix, where the formula WDAG is defined.

For every rule $\rho = A \to f \in P$ that is not a clone rule, we want to define a formula $\mathrm{P}_\rho(e, x_0, x_1, \ldots, x_k)$ that is true for a subgraph $g{\downarrow_e}$ if the following conditions are met:

1. x_0 is the unique root of $g{\downarrow_e}$ and e the unique edge in $g{\downarrow_e}$ connected to x_0.

2. The external nodes of $g{\downarrow_e}$ are $x_0 x_1 \cdots x_k$

3. $g{\downarrow_e}$ can be derived from A in G, starting with an application of ρ.

We next describe how to construct such formulas. In the construction, we use the formula $\mathrm{Reent}^i(x, v)$, that is true if and only if v is the ith reentrant node of edge x (the formula is defined in the Appendix). Further, let P_A be the subset of P having A as its left-hand side, for all $A \in N$.

Given $\rho = A \to f$, we assume without loss of generality that

- v_0 is the unique root of f and f_0 is the unique edge connected to v_0.
- $\mathrm{lab}_f(f_0) = a$
- $f_1, \ldots, f_\ell$ is the ordered sequence of nonterminal edges in f.

We arbitrarily number the non-root nodes in f and call them $v_1, \ldots, v_n$. We start the formula $\mathrm{P}_\rho(e, x_0, x_1, \ldots, x_k)$ by existentially quantifying over $n+1$ variables $y_0, y_1, \ldots, y_n$, corresponding to $v_0, v_1, \ldots, v_n$. Then we simply describe the structure of f with a conjunction of the following kinds of formulas:

- $\mathrm{Src}(e, y_0) \wedge \mathrm{Lab}_f(e)$
- If v_i is the jth target of f_0, then $\mathrm{Tar}^j(e, y_i)$.
- If v_i is the jth node in ext_f, then $y_i = x_j$.
- If v_i is the source of $f' \neq f_0$, and v_t is the jth target of f', then $\mathrm{Reent}^j(v_i, v_t)$.

The last item is motivated by the fact that in g, the nonterminal edges have been replaced by graphs, whose reentrancies are exactly the targets of the nonterminal edges.

We also need to add the recursive requirement that the graphs that have replaced the nonterminals of f could have been derived from them. To this end, let v_i be the source of f_j. We then need to state that for any edge d that has v_i as its source in g, the subgraph $g{\downarrow_d}$ could have been produced from f_j, or, more succinctly, that $g{\downarrow_d} \in \mathcal{L}(G_A)$ for $A = \mathrm{lab}(f_j)$.

Let f_j have rank s and let $\mathrm{tar}_f(f_j) = v_{i_1} \cdots v_{i_s}$. We then require that for any such edge d, the formula $P_{\rho'}(d, y_i, y_{i_1}, \ldots, y_{i_s})$ should hold, for some rule $\rho' \in P_A$. Furthermore, if $\mathrm{lab}_f(f_j)$ is not clonable, we require that y_i has outdegree 1 in g.

Note that the recursion involved in the rules will, for graphs generated by G, always terminate with rules where the right-hand side has no nonterminals.

Now that we have the formulas P_ρ, for all $\rho \in P$, we are ready to define the formula L_G that states that the input structure is a graph in the language of G. Let i be the rank of the start nonterminal S. We want to state that the input is a DAG and that it can be produced from S. In order to do so, we need to identify the root. Depending on whether or not S is clonable, we get a different variant of the formula. In the non-clonable case, we get

$$L_G = \mathrm{WDAG} \wedge \forall v(\mathrm{Root}(v) \to (\mathrm{Outdegree}(v) = 1 \wedge \exists e, u_1, \ldots, u_i(\mathrm{Src}(e, v) \wedge \bigvee_{\rho \in P_S} \mathrm{P}_\rho(e, v, u_1, \ldots, u_i)))).$$

In the case where S is clonable, the formula instead looks as follows.

$$L_G = \mathrm{WDAG} \wedge \forall v(\mathrm{Root}(v) \to (\exists u_1, \ldots, u_i(\forall e(\mathrm{src}(e, v) \to \bigvee_{\rho \in P_S} \mathrm{P}_\rho(e, v, u_1, \ldots, u_i))))).$$

The formula L_G is closed, and will be true for exactly those input structures that represent DAGs generated by G. We note that we have used no second-order quantification in the above, but it is needed in the definitions of WDAG and the formulas Reent^i, stated in the Appendix.

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 & Interoperability with Discourse, Sofia, Bulgaria*, 2013.

[3] Martin Berglund, Henrik Björklund, and Frank Drewes. Single-rooted DAGs in regular DAG languages: Parikh image and path languages. In *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+13)*, pages 94–101, 2017.

[4] H. Björklund, B. Björklund, and P. Ericson. On the regularity and learnability of ordered DAG languages. In Arnaus Carayol and Cyril Nicaud, editors, *22nd International Conference on the Implementation and Application of Automata (CIAA 2017), Marne-la-Vallée, France*, volume 10329 of *Lecture Notes in Computer Science*, pages 27–39. Springer International Publishing, 2017.

[5] H. Björklund, F. Drewes, and P. Ericson. 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 (LATA 2016), Prague, Czech Republic, 2016*, pages 521–532, 2016.

[6] Henrik Björklund, Frank Drewes, Petter Ericson, and Florian Starke. Uniform parsing for hyperedge replacement grammars. Technical Report UMINF 18.13, Umeå University, http://www8.cs.umu.se/research/uminf/index.cgi, 2018. Submitted for publication.

[7] Johanna Björklund, Andreas Maletti, and Jonathan May. Bisimulation minimisation for weighted tree automata. In *Proceedings of the 11th International Conference on Developments in Language Theory (DLT 2007), Turku, Finnland*, volume 4588 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2007. Springer Verlag.

[8] Johannes Blum and Frank Drewes. Properties of regular DAG languages. In Adrian-Horia Dediu, Jan Janousek, Carlos Martín-Vide, and Bianca Truthe, editors, *LATA*, volume 9618 of *Lecture Notes in Computer Science*, pages 427–438. Springer, 2016.

[9] Björn Borchardt. The Myhill-Nerode theorem for recognizable tree series. In Zoltán Ésik and Zoltán Fülöp, editors, *Developments in Language Theory*, pages 146–158, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[10] 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), Sofia, Bulgaria*, pages 924–932, 2013.

[11] David Chiang, Frank Drewes, Daniel Gildea, Adam Lopez, and Giorgio Satta. Weighted DAG automata for semantic graphs. *Computational Linguistics*, 44(1):119–186, 2018.

[12] 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.

[13] Frank Drewes and Heiko Vogler. Learning deterministically recognizable tree series. *Journal of Automata, Languages and Combinatorics*, 12(3):332–354, 2007.

[14] S. Gilroy, A. Lopez, S. Maneth, and P. Simonaitis. (Re)introducing regular graph languages. In *Proceedings of the 15th Meeting on the Mathematics of Language (MOL 2017)*, pages 100–113, 2017.

[15] Andreas Maletti. Learning deterministically recognizable tree series — revisited. In Symeon Bozapalidis and George Rahonis, editors, *Algebraic Informatics*, pages 218–235, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[16] Andreas Maletti. Minimizing deterministic weighted tree automata. *Information and Computation*, 207(11):1284–1299, 2009.

[17] Ehud Y. Shapiro. *Algorithmic Program DeBugging*. MIT Press, Cambridge, MA, USA, 1983.

Appendix

In this section, we define the formulas $\text{Reent}^i(e, v)$ and $\text{Reent}(e, X)$, as well as the formula WDAG, stating that the input structure is a well-ordered DAG.

Basic graph properties

We define formulas stating that there is a single root (and thus that the graph is connected and fully reachable from said root), that each element of the universe is either an edge or a node, and that each edge has a unique label, a unique source, and the correct number of targets.

$$\begin{aligned}
\text{Single-Rooted} &= \exists r(\text{Root}(r) \wedge \forall u(\text{Root}(u) \rightarrow (u = r))) \\
\text{Partition} &= \forall x(\text{Edge}(x) \leftrightarrow \neg\text{Node}(x)) \\
\text{UniqueLabels} &= \forall x(\text{Edge}(x) \rightarrow (\bigvee_{a\in\Sigma} (\text{Lab}_a(x) \bigwedge_{b\in\Sigma\setminus\{a\}} \neg\text{Lab}_b(x))) \wedge \\
&\quad (\neg\text{Edge}(x) \rightarrow \bigwedge_{a\in\Sigma} \neg\text{Lab}_a(x))) \\
\text{UniqueSources} &= \forall e(\text{Edge}(e) \rightarrow \exists v(\text{Src}(e, v) \wedge \forall u(\text{Src}(e, u) \rightarrow (u = v)))) \wedge \\
\text{Targets} &= \forall e(\text{Edge}(e) \rightarrow \bigvee_{a\in\Sigma} (\text{Lab}_a(e) \wedge \\
&\quad \bigwedge_{i\in[\text{rank}(a)]} (\exists v(\text{Tar}^i(e, v) \wedge \forall u(\text{Tar}^i(e, u) \rightarrow (u = v))))) \wedge \\
&\quad \bigwedge_{j\in[r]\setminus[\text{rank}(a)]} (\neg\exists v(\text{Tar}^j(e, v))))
\end{aligned}$$

We also define a formula Externals that makes sure that the root is external, the rest of the external nodes are leafs, and, by enumeration, that they are as many as the rank of the start nonterminal. With this in hand, we define the formula Graph, that simply ensures all of the above properties:

$$\text{Graph} = \text{Single-Rooted} \wedge \text{Partition} \wedge \text{UniqueLabels} \wedge \text{UniqueSources} \wedge \text{Targets} \wedge \text{Externals}$$

Reachability and reentrancies

In order to be able to speak about reachability, we define the notion of a set being closed under the Src and Tar relations:

$$\text{Closed}(S) = \forall x \forall y(x \in S \wedge (\text{Src}(y, x) \vee \text{Tar}(x, y)) \rightarrow y \in S)$$

In other words, if $x \in S$, then any edge or node reachable from x also belongs to S.

Towards the definitions of reentrancies and ordering we now define directed reachability.

$$\text{ReachableSet}(x, Y) = \forall S((\text{Closed}(S) \wedge x \in S) \rightarrow \forall y(y \in Y \rightarrow y \in S))$$

The above formula makes sure that Y is the smallest closed set that contains x, or, in other words, that Y is the set of nodes and edges reachable from x. For convenience, we also define reachability between individual nodes and edges:

$$\text{Reachable}(x, y) = \exists Y(\text{ReachableSet}(x, Y) \wedge y \in Y)$$

We are now ready to define the set of reentrant nodes with respect to a node or edge:

$$\begin{aligned}\text{Reent}(x, X) = \forall y(y \in X \leftrightarrow (\text{Reachable}(x, y) \wedge \text{Leaf}(y) \wedge \\ (\text{Ext}(y) \vee \exists z(\neg\text{Reachable}(x, z) \wedge \neg\text{Reachable}(z, x) \wedge \text{Reachable}(z, y)))))\end{aligned}$$

The formula says that for y to be reentrant with respect to x, it has to be a leaf reachable from x and either be external or also reachable from some z such that z is not reachable from x and x is not reachable from z. This is also a sufficient condition.

We require that all edges have the same set of reentrant nodes as their sources. This neatly covers the requirement that edges sharing the same source have the same set of reentrant nodes.

$$\begin{aligned}\text{ReentClones} = \forall e, v(\text{Src}(e, v) \rightarrow \\ \forall x, X, Y(\text{Reent}(e, X) \wedge \text{Reent}(v, Y) \rightarrow (x \in X \leftrightarrow x \in Y)))\end{aligned}$$

Ordering

We can define the notion of closest common ancestor edges, as follows:

$$\text{CommonAncestor}(x, u, v) = \text{Reachable}(x, u) \wedge \text{Reachable}(x, v)$$

$$\begin{aligned}\text{CCAE}(e, u, v) = \text{Edge}(e) \wedge \text{CommonAncestor}(e, u, v) \wedge \\ \neg\exists w(\text{Tar}(e, w) \wedge \text{CommonAncestor}(w, u, v))\end{aligned}$$

To define the ordering of leaves, we have the following.

$$\begin{aligned}\text{Before}(u, v) = \text{Leaf}(u) \wedge \text{Leaf}(v) \wedge ((u = v) \vee \forall e(\text{CCAE}(e, u, v) \\ \rightarrow \exists x \forall y(\text{Reachable}(x, u) \wedge \text{Reachable}(y, v) \wedge \text{Tar}^i(e, x) \wedge \text{Tar}^j(e, y) \rightarrow i < j)))\end{aligned}$$

This is a shorthand, as we cannot directly compare i and j. However, as the alphabet has bounded rank, the above can be achieved by a disjunction over all relevant pairs of values for i and j.

Now, to make sure we have a consistent ordering, we use the following formula.

$$\text{ConsistentOrdering} = \forall e, u, v((\text{CCAE}(e, u, v) \wedge u \neq v) \rightarrow (\text{Before}(u, v) \leftrightarrow \neg\text{Before}(v, u)))$$

Finally, we need to make sure that the graph is really a DAG:

$$\text{DAG} = \forall x, y(\text{Reachable}(x, y) \wedge \text{Reachable}(y, x) \rightarrow x = y)$$

In conclusion, our precondition amounts to

$$\text{WDAG} = \text{Graph} \wedge \text{ReentClones} \wedge \text{ConsistentOrdering} \wedge \text{DAG}$$

Given a consistent ordering, we can find the ordering of the reentrant nodes, and in particular if a specific leaf v is the ith member of the sequence of reentrant leaves of some edge or node x.

$$\text{Reent}(x, y) = \forall X(\text{Reent}(x, X) \rightarrow y \in X)$$

$$\begin{aligned}\text{Reent}^i(x, v) = \text{Reent}(x, v) \wedge \exists u_1, \ldots u_i(\bigwedge_j \text{Reent}(x, u_j) \wedge \bigwedge_j \text{Before}(u_j, v) \wedge \\ \bigwedge_{j \neq k} (u_j \neq u_k) \wedge \forall y(\text{Reent}(x, y) \wedge \text{Before}(y, x) \rightarrow \bigvee_j (y = u_j)))\end{aligned}$$

The last formula simply enumerates i nodes that are reentrant for x, the last of which is v. This concludes the definition of the formulas used in Section 7.