Parsing Weighted Order-Preserving Hyperedge Replacement Grammars

Henrik Björklund[1], Frank Drewes[1], and Petter Ericson[1]

Department of Computing Science, Umeå University
{henrikb,drewes,pettter}@cs.umu.se

Abstract. We introduce a weighted extension of the recently proposed notion of order-preserving hyperedge-replacement grammars and prove that the weight of a graph according to such a weighted graph grammar can be computed uniformly in quadratic time (under assumptions made precise in the paper).

1 Introduction

The hyperedge-replacement grammar (HRG) is one of the most successful formalisms for describing graph languages; see, e.g., [2, 12, 11, 7]. It is also a promising candidate for modelling semantic representations of natural language such as Abstract Meaning Representation [1]. However, HRGs overshoot the mark in that parsing with respect to them is computationally too expensive. Recently, a suitable restriction called order preservation was proposed [4, 3, 5].

The present article builds upon the *order-preserving HRGs* (OPHGs) of [5]. It was shown in [5] that parsing for OPHGs is efficient, requiring polynomial time even in the uniform case, i.e. the grammar is considered to be part of the input. Here, we define a weighted version of OPHGs, and extend the results of [5] to show that when the weights are taken from a commutative semiring, we can efficiently compute the weight assigned by an OPHG to any input graph. This is an important feature since applications such as semantic modelling require ways to quantify the well-formedness of a generated graph.

Introducing weights for OPHGs requires some care, as the associativity and commutativity of some of the rules complicates the question which derivations of a certain graph are to be considered distinct. For this reason, we introduce a notion of hybrid derivation trees, in which some nodes have a set of children, while others have them ordered in a list. After this, we show how weights can efficiently be computed, and prove the correctness of the algorithm.

Related work. Another type of restricted HRGs for semantic modelling was proposed by Chiang et al. [6], together with a parsing algorithm and a detailed complexity analysis. The complexity is, however, exponential even in the non-uniform case. In particular, it is exponential in the maximum degree of nodes in the input graph. The same holds for the parsing algorithm for *regular graph grammars* presented by Gilroy et al. [10]. We also mention that another technique for efficient HRG parsing was resently developed by Drewes et al. [8, 9].

2 Preliminaries

The set of non-negative integers is $\mathbb{N}$, and $[k] = \{1, \ldots, k\}$. For a set S, S^* is the set of strings over S, while $S^{\circledast}$ is the set of strings in S^* in which no element of S occurs twice. The empty string is ϵ, and we have $S^+ = S^* \setminus \epsilon$ and $S^{\oplus} = S^{\circledast} \setminus \epsilon$. The length of a string w is denoted $|w|$. We use the terms 'string' and 'sequence' interchangably. For a sequence $w = a_1 \cdots a_n$, every sequence $a_{i_1} \cdots a_{i_k}$ with $1 \leq i_1 < \cdots < i_k \leq n$ is a *subsequence* of w, and $[w]$ is the set $\{a_1, \ldots, a_n\}$.

2.1 Hypergraphs

We fix a disjoint, countably infinite supply LAB of labels, such that each $\sigma \in \text{LAB}$ has a rank $\text{rank}(\sigma) \in \mathbb{N}$. A *hypergraph* is a structure $g = (V, E, \text{lab}, \text{att}, \text{ext})$ where V and E are the (finite) sets of *nodes* and *hyperedges*, $\text{lab} : E \to \text{LAB}$ is the *edge labelling*, $\text{att} : E \to V^{\oplus}$ is the *edge attachment* with $|\text{att}(e)| = \text{rank}(\text{lab}(e)) + 1$ for all $e \in E$, and $\text{ext} \in V^{\oplus}$ is the sequence of *external nodes*.

From now on, we simply call hypergraphs graphs, and hyperedges edges. We use the graph as a subscript to identify its components. E.g., E_g refers to the set of edges of g. Also, for $X \subseteq \text{LAB}$, we let $E_g^X = \{e \in E_g \mid \text{lab}_g(e) \in X\}$. For an edge $e \in E_g$ with $\text{att}(e) = v_0 \cdots v_k$, we say that $\text{src}_g(e) = v_0, \text{tar}_g(e) = v_1 \cdots v_k$, and name these the *source* and sequence of *targets*, respectively. Similarly, for $\text{ext}_g = v_0 \cdots v_l$, we say that $v_0 = \dot{g}$ is the *source* of the graph, and $v_1 \cdots v_l = g_{\cdot\cdot}$ its sequence of *targets*. In this paper, we require all targets of a graph to be leaves, i.e. $\text{src}_g(e) \notin [g_{\cdot\cdot}]$ for all $e \in E_g$. For a graph g, $\text{rank}(g) = |g_{\cdot\cdot}|$, and for an edge e, $\text{rank}(e) = \text{rank}(\text{lab}(e)) = |\text{tar}_g(e)|$. Graphs g, h are *isomorphic*, denoted $g \equiv h$, if they are equal up to a bijective renaming of nodes and edges.

For $a \in \text{LAB}$ with $\text{rank}(a) = k$, $a^\bullet$ denotes the graph $(\{v_0, \ldots, v_k\}, \{e\}, (e \to a), (e \to v_0 \cdots v_k), (v_0 \cdots v_k))$, i.e. the graph of one a-labelled edge of the proper rank, with all its attached nodes external.

An alternating sequence $v_1 e_1 \ldots v_k e_k$ of nodes and edges is a *path* in g from v_1 to e_k if $\text{src}_g(e_i) = v_i$ and $v_{i+1} \in [\text{tar}_g(e_i)]$, for each $i \in [k]$. We may optionally terminate the path at v_{k+1} instead of e_k. In either case, the path *passes* all nodes and edges v_i and e_i for $i \in [k]$. If $v_1 = \dot{g}$, it is a *source path*. A node v or edge e is *reachable from* s (in g) if there is a path in g from s to v (e). A node or edge is *reachable in* g if there is a source path to it.

2.2 Hyperedge replacement

For graphs h, f, and an edge $e \in E_h$ such that $\text{rank}(e) = \text{rank}(f)$, we can use *hyperedge replacement* to obtain the graph $g = h[\![e : f]\!]$, *substituting* f for e in h, where $g = ((V_h \cup V_f), (E_h \cup E_f) \setminus \{e\}, \text{att}_g, \text{lab}_g, \text{ext}_h)$ with

$$\text{att}_g(e') = \begin{cases} \text{att}_f(e') \text{ if } e' \in E_f \\ \text{att}_h(e') \text{ if } e' \in E_h \setminus \{e\} \end{cases} \text{ and } \text{lab}_g(e') = \begin{cases} \text{lab}_f(e') \text{ if } e' \in E_f \\ \text{lab}_h(e') \text{ if } e' \in E_h \setminus \{e\}. \end{cases}$$

Clearly, we can always choose isomorphic copies of h and f so that $h[\![e : f]\!]$ is defined. We will generally not make note of this, to avoid irrelevant technicalities.

For the case where $g = h[\![e : f]\!]$ and $i = g[\![e' : j]\!]$ with $e' \notin E_f$, we write $i = h[\![e : f, e' : j]\!]$, and similarly for a larger number of replacements.

We divide LAB into two subsets $\mathrm{LAB_T}$ and $\mathrm{LAB_N}$ of *terminals* and *nonterminals*, and accordingly call edges terminal and nonterminal ones. We sometimes shorten the expressions further to just "terminals" and "nonterminals".

2.3 Hyperedge replacement grammars

A *hyperedge replacement grammar* (HRG) $G = (\Sigma, N, S, R)$ consists of a *terminal alphabet* $\Sigma \subset \mathrm{LAB_T}$, a *nonterminal alphabet* $N \subset \mathrm{LAB_N}$, an *initial nonterminal* $S \in N$, and a set R of *(HR) rules* form $A \to f$, where $A \in N$ and f is a graph over $\Sigma \cup N$ with $\mathrm{rank}(A) = \mathrm{rank}(f)$ and $E_f^N = \{e_1, \ldots, e_\ell\}$ for an $\ell \in \mathbb{N}$. We write $\mathit{arity}(A \to f)$ for ℓ. Note the naming scheme used for nonterminal hyperedges in right-hand sides: they are always named $e_1, \ldots, e_\ell$ for an appropriate ℓ.

If we have a graph h with an edge e with $\mathrm{lab}_h(e) = A \in N$, and $A \to f \in R$, we can *derive* $g = h[\![e : f]\!]$. We call this a *derivation step*, and denote it $h \to_{A \to f} g$. We also write more generally $h \to_G g$ for a derivation step using any rule in R. The reflexive and transitive closure of $\to_G$ is $\to_G^*$. The *language of* G is the set $\mathcal{L}(G)$ of all graphs g over $\mathrm{LAB_T}$ such that $S^\bullet \to_G^* g$.

3 Order-Preserving Hyperedge Replacement Grammars

We now turn to order-preserving HRGs. The first ingredient is a condition called reentrancy preservation. Reentrancies are deeply entwined with the way we identify places in a graph that match the right-hand side of a given rule.

3.1 Reentrancies

Intuitively, the *reentrant nodes* of a node or edge x in a graph g are the first descendants of x that can also be reached on a path that avoids x. As the external nodes of a right-hand side of an HR rule are the ones that, after the replacement, are reachable from "outside" the subgraph, we also consider them as reentrant. The graph delineated by x and its reentrant nodes is the *subgraph rooted at* x.

Definition 1 (Reentrant node). *Given a graph g and $E \subset E_g$, let* $\mathrm{TAR}_g(E)$ *be the union of all sets of targets of edges in E, i.e.* $\bigcup_{e \in E} [\mathrm{tar}_g(e)]$.

Further, for $x \in V_g \cup E_g$, let $\hat{x}$ be x if $x \in V_g$, and $\mathrm{src}_g(x)$ *if $x \in E_g$. Now, let E_g^x be the set of all edges $e \in E_g$ such that all source paths to e pass x.*[1] *Then the* set of reentrant nodes of x in g *is*

$$\mathrm{reent}_g(x) = (\mathrm{TAR}_g(E_g^x) \setminus \{\hat{x}\}) \cap (\mathrm{TAR}_g(E_g \setminus E_g^x) \cup [\mathrm{ext}_g]).$$

Definition 2 (Rooted subgraph). *Given a graph g with $x \in V_g \cup E_g$, the* subgraph $g{\downarrow_x}$ rooted at x *is a graph h such that $E_h = E_g^x$, $V_h = \{\hat{x}\} \cup \mathrm{TAR}_g(E_h)$,* att_h *and* lab_h *are the appropriate restrictions of* att_g *and* lab_g*, respectively, and* ext_h *is $\hat{x}$ followed by* $\mathrm{reent}_h(x)$ *in some order.*

[1] Note that if x is not reachable in g, $E_g^x = \emptyset$

Rooted subgraphs are strictly nested, which is proved in [5] as the following lemma (where $\sim$ is isomorphy modulo the order of $g_{\cdot\cdot}$):

Lemma 1 (Lemma 3.4 in [5]). *Let g be a graph, $h = g{\downarrow_x}$ for some $x \in V_g \cup E_g$. Then $h{\downarrow_y} \sim g{\downarrow_y}$ for all $y \in (V_h \cup E_h) \setminus [\mathrm{ext}_h]$*

3.2 Reentrancy Preservation

Reentrancy preservation formalizes the property that, given a graph h and some edge $e \in E_h$ with $\mathrm{lab}_h(e)$, we can replace e by some graph f according to a rule $A \to f$ without affecting the sets $\mathrm{reent}_g(x)$ for $x \in V_h \cup V_f$.

We achieve this by restricting our grammars to two types of rules, namely *duplication rules* and *deep rules*. Rules of these two kinds are called *reentrancy preserving*. To define duplication rules, consider a graph

$$f = (\{v_0, \ldots, v_n\}, \{e_1, e_2\}, \mathrm{att}, \mathrm{lab}, \mathrm{ext}),$$

where $\mathrm{att}(e_1) = v_0 \cdots v_n = \mathrm{att}(e_2)$, $\mathrm{lab}(e_1) = \mathrm{lab}(e_2) \in \mathrm{LAB_N}$, and ext is a subsequence of $\mathrm{att}(e_1)$ starting with v_0. If $|\mathrm{ext}| < n$ then f (and every graph isomorphic to f) is a *twin*, and if $|\mathrm{ext}| = n$ then it is a *clone*. A rule $A \to f$ is a *twin rule* if f is a twin and a *clone rule* if f is a clone with $\mathrm{lab}(e_1) = \mathrm{lab}(e_2) = A$. A *duplication rule* is either a clone or a twin rule.

A rule $A \to f$ is a *deep rule* if f fulfills the following conditions:

- $V_f \neq [\mathrm{ext}_f]$,
- all nodes in V_f are reachable from $\dot{f}$ and have out-degree ≤ 1, and
- for every nonterminal edge e, $\mathrm{reent}_f(e) = [\mathrm{tar}_f(e)]$.

A HRG is reentrancy preserving if it has only reentrancy-preserving rules. We note here that [5] also permits chain rules, i.e. rules that violate the first condition above. In the present paper we exclude them because they can result in an infinite number of derivations of a given graph, thus making it in general unreasonable to associate a weight with such a graph.

Later on, we will also need the following generalization of duplication rules to the case where $\ell \geq 2$ copies of a nonterminal edge are created: given any duplication rule $r = (A \to f)$ and some $\ell \geq 2$, we denote by r^ℓ the rule $A \to f'$, where f' is obtained from f by replacing e_1, e_2 by ℓ copies. Thus, $r^2 = r$.

Lemma 2 (Adapted from lemma 5.6 in [5]). *Let $g \in \mathcal{L}(G)$ for some reentrancy-preserving HRG G. There is a quadratic algorithm that computes, for every $x \in V_g \cup E_g$, the set $\mathrm{reent}_g(x)$, and thus the subgraph $g{\downarrow_x}$.*

3.3 Ordering nodes

Reentrancy preservation allows us to pinpoint the subgraphs that may have been generated by a specific nonterminal, but as shown in [4], this is not sufficient to achieve efficient parsing, as needing to guess the order of targets in subgraphs

$g{\downarrow_x}$ may still cause NP-hardness. Thus, we require a way to determine the order of nodes, in particular reentrant nodes. This requires an ordering relation that can be efficiently computed, and fulfils some basic requirements, and a set of reentrancy-preserving rules that additionally *preserves that order*. Formally:

Definition 3 (Suitable order). *For a set $\mathcal{G}$ of graphs, a* suitable family of orders *is a family $(\preceq_g)_{g\in\mathcal{G}}$ of binary relations $\preceq_g \subseteq V_g \times V_g$ such that*

- *for all $A \in \mathrm{LAB}_N$, $A^{\bullet}_{\bullet\bullet}$ is ordered by $\preceq_{A^\bullet}$ and*
- *if $i\colon g \to h$ is an isomorphism and $u, v \in V_g$, then $u \preceq_g v$ iff $i_V(u) \preceq_h i_V(v)$.*

Definition 4 (Order preservation). *A reentrancy-preserving set R of HR rules* preserves *a suitable family of orders $\preceq = (\preceq_g)_{g\in\mathcal{G}}$ if, for all $g = h[e : f]$ with $g, h, f \in \mathcal{G}$, $e \in E_h$, and $\mathrm{lab}_h(e) \to f \in R$, we have $\preceq_g|_{V_h} = \preceq_h$ and $\preceq_f|_{V_f} = \preceq_f$.*

An order-preserving HRG (OPHG) *is an HRG (Σ, N, S, R) together with a suitable family $\preceq$ of orders, such that R is both order preserving and preserves $\preceq$.*

4 Weighted Order-Preserving HR Grammars

We now add weights – taken from some semiring – to order-preserving HR grammars. For this, and throughout the rest of this paper, let $\mathcal{S} = (S, +, \cdot, 0, 1)$ be a commutative semiring, meaning that $(S, +, 0)$ and $(S, \cdot, 1)$ are two monoids over the domain S. Thus, $+$ and $\cdot$ are binary operations on S such that

- 1 is the identity element for $\cdot$
- 0 is the identity element for $+$ and the absorbing one for $\cdot$,
- $+$ and $\cdot$ are commutative, and
- $\cdot$ distributes over $+$.

As usual, for every $a \in S$ we let $a^0 = 1$ and $a^{n+1} = a \cdot a^n$ for all $n \in \mathbb{N}$.

A weighted OPHG computes a graph series, i.e. a mapping of graphs to S. As usual, this is achieved by assigning weights to rules.

Definition 5 (weighted OPHG). *A* weighted OPHG *$G = (\Sigma, N, S, R, \omega)$ (over $\mathcal{S}$) consists of an OPHG (Σ, N, S, R) and a* weight assignment *$\omega\colon R \to S$.*

Informally speaking, if several distinct derivations can produce the same graph, we sum up the weights of the individual derivations to obtain the weight of the graph. The weight for a single derivation is the product of the weights of all the rules applied.

It is inconvenient to formalise this based on the derivations themselves because, just as in the case of ordinary context-free grammars, derivations may differ only in the order in which nonterminals are replaced, which yields distinct derivations that should not be distinguished. A standard technique to solve this problem is to consider derivation trees instead of derivations. We can mostly use this standard technique, but we also have to take into account that duplication rules have

certain associativity and commutativity properties that make it inappropriate to sum up over derivation trees that, intuitively, should be considered equivalent.

Let us begin the process of making these notions more precise by recalling the notions of *shallow graphs* and *siblinghoods* from [5].

Definition 6. *A graph g is* shallow *if $\dot{g} = \mathrm{src}_g(e)$ for all $e \in E_g$. A* siblinghood *in g is a set* $\mathrm{Sib} \subseteq E_g$ *such that* $|\mathrm{Sib}| \geq 2$ *and* $\mathrm{tar}_g(e) = \mathrm{tar}_g(e')$ *for all* $e, e' \in \mathrm{Sib}$. *We denote* $\mathrm{tar}_g(e)$, $e \in \mathrm{Sib}$, *by* $\mathrm{tar}_g(\mathrm{Sib})$, *and let* $g(\mathrm{Sib}) = (\{\dot{g}\} \cup [\mathrm{tar}_g(\mathrm{Sib})], \mathrm{Sib}, \mathrm{att}_g|_{\mathrm{Sib}}, \mathrm{lab}_g|_{\mathrm{Sib}}, \mathrm{tar})$, *where* tar *is the subsequence of* $\mathrm{tar}_g(\mathrm{Sib})$ *of nodes that are external in g or targets of edges outside of* Sib, *i.e. that belong to* $\mathrm{TAR}_g(\mathrm{Sib}) \cap (\mathrm{TAR}_g(E_g \setminus \mathrm{Sib}) \cup [g_{\bullet\bullet}])$

For siblinghoods $\mathrm{Sib}, \mathrm{Sib}'$, *we let* $\mathrm{Sib} \leq \mathrm{Sib}'$ *if* $\mathrm{tar}_g(\mathrm{Sib})$ *is a subsequence of* $\mathrm{tar}_g(\mathrm{Sib}')$. *A siblinghood of g is* prime *if it is maximal with respect to both $\leq$ and set inclusion.*

From now on, we shall for technical simplicity assume that the considered OPHG G contains exactly one clone rule for every $A \in N$. This is not a restriction because the definition of the weight of derived graphs to be given below ensures that any number of clone rules for the same nonterminal can be replaced by a single clone rule whose weight is the sum of the weights of the individual rules. In particular, if there is no clone rule for A, this has the same effect as a single clone rule of weight 0. The weight of the unique clone rule for $A \in N$ is denoted by $\omega(A)$, and we write $\to_{\mathrm{cl}}$ for the derivation relation that exclusively uses clone rules, i.e. $g \to_{\mathrm{cl}}^* g'$ if g' is obtained from g by cloning nonterminal edges.

The following is essentially Lemma 5.3 of [5]:

Lemma 3. *Let $A \in N$ and let g be a shallow graph over N with $|E_g| \geq 2$.*

- *If $A^\bullet \to^+ g$, then for every prime siblinghood* Sib *of g we either have $g = g(\mathrm{Sib})$ and $A^\bullet \to_{\mathrm{cl}}^+ g$, or $A^\bullet \to^* h \to h[\![e : f]\!] \to_{\mathrm{cl}}^* h[\![e : f']\!] = g$ where $\mathrm{lab}_h(e) \to f$ is a twin rule and $g(\mathrm{Sib}) = f'$.*
- *Up to reordering of derivation steps, the derivations of these forms are the only ones deriving g from $A^\bullet$.*

Hence, a derivation of a shallow graph can be broken down into an initial series of clonings followed by iterated sub-derivations each consisting of an application of a twin rule $A \to f$ and any number of clonings of the two nonterminal edges e_1, e_2 of f. Note that the result of each such sub-derivation depends only on $A \to f$ and the number of clonings since $\mathrm{att}_f(e_1) = \mathrm{att}_f(e_2)$. Therefore, the following definition of derivation trees uses trees in which the nodes that correspond to derivations of siblinghoods are unordered and unranked. For a tree consisting of a root labelled a and subtrees $t_1, \ldots, t_\ell$, we write $a[t_1, \ldots, t_\ell]$ or $a\langle t_1, \ldots, t_\ell \rangle$ depending on whether $t_1, \ldots, t_\ell$ is to be interpreted as an ordered or unordered list (or a multiset), respectively. We write $a(t_1, \ldots, t_\ell)$ to denote a tree in which the first level of children can be either ordered or unordered.

Definition 7 (derivation tree). *For a weighted OPHG $G = (\Sigma, N, S, R, \omega)$ and $A \in N$, the set of all* A-derivation trees *is the smallest set of trees t such that one of the following holds:*

(1) $t = r[t_1, \ldots, t_\ell]$ *for a deep rule* $r = (A \to f) \in R$ *such that* $\mathrm{arity}(A \to f) = \ell$, *and* t_i *is a* $\mathrm{lab}_f(e_i)$*-derivation tree for every* $i \in [k]$.

(2) $t = r^\ell\langle t_1, \ldots, t_\ell\rangle$ *for a clone rule* $A \to f$, *where* $\ell \geq 2$ *and* t_i *is an* A*-derivation tree that is not of type (2), for every* $i \in [\ell]$.

(3) $t = r^\ell\langle t_1, \ldots, t_\ell\rangle$ *for a twin rule* $A \to f$, *where* $\ell \geq 2$ *and* t_i *is a* $\mathrm{lab}_f(e_1)$*-derivation tree that is not of type (2), for every* $i \in [\ell]$.

We can *evaluate* a derivation tree to yield a graph g in the following way: Given a derivation tree $t = r(t_1, \ldots, t_\ell)$, $\mathit{eval}(t)$ is defined as the right-hand side f of r, with each successive nonterminal e_i replaced with the evaluation of the corresponding subtree of the derivation tree, i.e. $\mathit{eval}((A \to f)(t_1, \ldots, t_\ell)) = f[\![e_1 : \mathit{eval}(t_1), \ldots, e_\ell : \mathit{eval}(t_\ell)]\!]$. Given a graph g, we let $\mathrm{DT}_G(g)$ denote the set of all S-derivation trees such that $\mathit{eval}(t) \equiv g$.

We make the following observation, whose correctness follows from the context-freeness of hyperedge replacement.

Observation 1 For every OPHG $G = (\Sigma, N, S, R, \omega)$,

$$\mathcal{L}(G) = \{\mathit{eval}(t) \mid t \text{ is an } S\text{-derivation tree of } G\}.$$

Now, as mentioned, the weight of a graph is defined to be the sum of the weights of all its derivation trees:

Definition 8 (generated graph series). *Let* $G = (\Sigma, N, S, R, \omega)$ *be a weighted OPHG and* $A \in N$.

1. *For every duplication rule* $r = (A \to f) \in R$ *and every* $\ell \geq 2$, *let* $\omega(r^\ell) = \omega(r) \cdot \omega(\mathrm{lab}_f(e_1))^{\ell-2}$. *(Note that* r^ℓ *corresponds to the application of* r *followed by* $\ell - 2$ *clonings of any of the two resulting nonterminal edges.)*
2. *The weight of an* A*-derivation tree* $t = r(t_1, \ldots, t_\ell)$ *is defined inductively, as*

$$\omega(t) = \omega(r) \cdot \prod_{i \in [k]} \omega(t_i).$$

3. *The graph series* $\omega_G \colon \mathcal{G}_\Sigma \to S$ *generated by* G *is given by*

$$\omega_G(g) = \sum_{t \in \mathrm{DT}_G(g)} \omega(t).$$

(The sum is finite, and thus well defined due to the commutativity of $+$*.)*

Note that given G, the language $\mathcal{L}(G)$ of G seen as an unweighted grammar, is a subset of the *support* of G, i.e. the set of all graphs g such that $\omega_G(g) \neq 0$.

5 Computing Weights

Our algorithm builds upon the unweighted parsing algorithm from [5]. We store in each node and edge nothing more than an $|N|$-vector of weights, which is computed in very much the same way as the sets of nonterminals computed in [5]. We use the distributivity of multiplication over addition to keep our computations efficient (assuming efficient multiplication and addition).

The algorithm exploits Lemma 1, i.e. the property that the subgraphs $g{\downarrow_x}$ are strictly nested in all graphs derivable by an OPHL. Using this, it is possible to process the subgraphs of g in a tree-like "bottom-up" manner, marking each node and edge x with the set of all nonterminals that can generate $g{\downarrow_x}$, after all $g{\downarrow_y}$ properly contained in $g{\downarrow_x}$ have already been processed. Eventually, S belongs to the set $\dot{g}$ is marked with if and only if $g \in \mathcal{L}(G)$.

Order preservation enters the picture as follows: every subgraph h of g which was derived from some nonterminal edge, is of the form $h = g{\downarrow_x}$ for some node or edge x of g. As shown in [5], order preservation guarantees that $h_{\bullet\bullet}$ is ordered by $\preceq_g$. Thus, in the algorithm only those subgraphs $g{\downarrow_x}$ are of interest for which the ordering of targets is uniquely determined by $\preceq_g$. From now on, we will thus assume that, whenever a subgraph $h = g{\downarrow_x}$ is constructed, the order of nodes in $h_{\bullet\bullet}$ is chosen according to $\preceq_g$.

To show how $\omega_G(g)$ can be computed, we describe two algorithms in one: the first computes the derivation trees of g whereas the second computes its weight by summing up over all the derivation trees. In the current paper, we mainly use the first algorithm as a tool to facilitate the correctness proof of the second. The set of derivation trees computed can, however, be represented in a compact fashion as a "packed forest", which is of independent usefulness.

The main procedure of the algorithm computes, in the same bottom-up manner as in [5], a set $\mathcal{D}_x(A)$ of A-derivation trees for each $x \in V_g \cup E_g$ and every $A \in N$. More precisely, $\mathcal{D}_x(A)$ is the set of all A-derivation trees of the input HRG G such that $A^\bullet \to_G^* g{\downarrow_x}$. As the correctness of this procedure was proved in [5] (though not explicitly in terms of derivation trees), it remains to show that the second version of the algorithm computes $\sum_{t \in \mathcal{D}_{\dot{g}}(S)} \omega(t)$.

That second version computes weights $\mathcal{W}_x(A)$ instead of the sets $\mathcal{D}_x(A)$, where $\mathcal{W}_x(A) = \sum_{t \in \mathcal{D}_x(A))} \omega(t)$. In the pseudocode below, we always indicate the changes that must be made to obtain the second version by lines marked by "***alt:***". The corresponding line always replaces its immediate predecessor. For sets of (derivation) trees $D_1, \ldots, D_\ell$ and a rule r of arity ℓ, we furthermore write $r(D_1, \ldots, D_\ell)$ to denote the set $\{r(t_1, \ldots, t_\ell) \mid (t_1, \ldots, t_\ell) \in D_1 \times \cdots \times D_\ell\}$ (i.e. we use that notation in both the ordered and unordered case).

A subroutine used by the algorithm is Algorithm 1, a modified version of the corresponding procedure in [5]. It takes as input a shallow graph h whose edges e are already assumed to be annotated with the respective sets $\mathcal{D}_e(A)$. The algorithm uses Lemma 3 in order to assemble – in a bottom-up manner over the prime siblinghoods of h – the set $\mathcal{D}_{\dot{h}}(A)$. In the algorithm we say that a duplication rule $A \to f$ of G *fits* a siblinghood $\mathrm{Sib} = \{s_1, \ldots, s_\ell\}$ of h

Algorithm 1 Computing Derivation Trees with Duplication Rules

function SHALLOWPARSE(set R of duplication rules, shallow annotated graph h with irrelevant edge labels)
  while $|E_g| > 1$ **do**
    if h does not contain a prime siblinghood **then**
      return $(A \mapsto \emptyset)_{A \in N}$
      alt: **return** $(A \mapsto 0)_{A \in N}$
    choose a prime siblinghood $\mathrm{Sib} = \{s_1, \ldots, s_\ell\}$
    replace Sib in h by a new edge e with $\mathrm{tar}_h(e) = h(\mathrm{Sib})$
    for each $A \in N$ **do**
      $\mathcal{D}_e(A) \leftarrow \bigcup_{r = (A \to B^{\bullet\bullet}) \text{ fits Sib}} r^\ell \langle \mathcal{D}_{s_1}(B), \ldots, \mathcal{D}_{s_\ell}(B) \rangle$
      alt: $\mathcal{W}_e(A) \leftarrow \sum_{r = (A \to B^{\bullet\bullet}) \text{ fits Sib}} \omega(r^\ell) \cdot \prod_{i \in [\ell]} \mathcal{W}_{s_i}(B)$
  return $(A \mapsto \mathcal{D}_e(A))_{A \in N}$ where $\{e\} = E_h$
  alt: **return** $(A \mapsto \mathcal{W}_e(A))_{A \in N}$ where $\{e\} = E_h$

Algorithm 2 Computing Derivation Trees for Order-Preserving HR Grammars

function PARSE(order-preserving HR grammar $G = (\Sigma, N, S, R)$, graph $g \in \mathcal{G}_R$)
  $preProcess(g)$ ▷ Compute $\prec_g$ as well as all $g{\downarrow_x}$ for all $x \in V_g \cup E_g$
  for $x \in V_g \cup E_g$ **do**
    if $g{\downarrow_x}$ is defined **then** $\mathcal{D}_x \leftarrow \bot$
    else
      $\mathcal{D}_x \leftarrow (A \mapsto \emptyset)_{A \in N}$
      alt: $\mathcal{W}_v \leftarrow (A \mapsto 0)_{A \in N}$
  while $\mathcal{D}_g(\cdot) = \bot$ **do**
    let $x \in V_g \cup E_g$ with $\mathcal{D}_x = \bot$ and
      $\mathcal{D}_y \neq \bot$ for all $y \in (V_{g\downarrow_x} \cup E_{g\downarrow_x}) \setminus ([\mathrm{ext}_{g\downarrow_x}] \cup \{x\})$
    if $x \in V_g$ **then** $\text{PARSE}_\text{V}(x)$
    else $\text{PARSE}_\text{E}(x)$
  return $\mathcal{D}_g(S)$
  alt: **return** $\mathcal{W}_g(S)$

if $f \equiv h(\{s_1, s_2\})$ when disregarding edge labels, and we denote f by $B^{\bullet\bullet}$ to indicate that the two edges in f carry the label B.

The reader should note that the result of Algorithm 1 does not depend on the choice of Sib because the prime siblinghoods $\mathrm{Sib}_1, \ldots, \mathrm{Sib}_k$ of h are pairwise disjoint and the replacement of $\mathrm{Sib} = \mathrm{Sib}_i$ by e does not affect the siblinghoods Sib_j, $j \in [k] \setminus \{i\}$ (though it may of course create an additional prime siblinghood).

The main procedure of the parsing algorithm is shown in Algorithm 2. In its while loop, it repeatedly chooses an $x \in V_g \cup E_g$ for which the sets $\mathcal{D}_x(A)$ shall be computed, and calls PARSE_V (Algorithm 3) or PARSE_E (Algorithm 4) depending on whether $x \in V_g$ or $x \in E_g$.

The function MATCHING used in line 5 of Algorithm 4 is described in [5] (using slightly different notation). It is based on the fact that, if $g{\downarrow_e}$ can be derived from a deep right-hand side f, then the mapping ϕ of the nodes in f to their

Algorithm 3 Computing Derivations Trees of $g{\downarrow_v}$ for nodes $v \in V_g$

function PARSE$_\mathrm{V}$(node v such that $\mathcal{D}_e(A) \neq \bot$ for all $e \in E_g$ with $\mathrm{src}_g(e) = v$)
  if v has out-degree 0 **then**
    $\mathcal{D}_v \leftarrow (A \mapsto \emptyset)_{A \in N}$
    alt: $\mathcal{W}_v \leftarrow (A \mapsto 0)_{A \in N}$
  else
    initialize $h = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext})$ as the following shallow graph:
    $E = \{e \in E_g \mid \mathrm{src}_g(e) = v\}$
    $V = \{v\} \cup \bigcup_{e \in E} \mathrm{reent}_g(e)$
    $\mathrm{ext} = \mathrm{ext}_{g{\downarrow_v}}$
    $\mathrm{att}(e) = vw$, where w is $\mathrm{reent}_g(e)$ ordered by $\preceq_g$, for each $e \in E$
    $\mathcal{D}_v \leftarrow$ SHALLOWPARSE$(\{r \in R \mid r \text{ a duplication rule}\},\ h)$
    alt: $\mathcal{W}_v \leftarrow$ SHALLOWPARSE$(\{r \in R \mid r \text{ a duplication rule}\},\ h)$

Algorithm 4 Computing Derivations Trees of $g{\downarrow_e}$ for edges $e \in E_g$

function PARSE$_\mathrm{E}$(edge e s.t. $\mathcal{D}_y \neq \bot$ for all $y \in (V_{g(x)} \cup E_{g(x)}) \setminus ([\mathrm{ext}_{g(x)}] \cup \{x\})$)
  $\mathcal{D}_e(A) \leftarrow \emptyset$ for all $A \in N$
  alt: $\mathcal{W}_e(A) \leftarrow 0$ for all $A \in N$
  for each deep rule $r = (A \to f)$ of arity ℓ **do**
    $\phi \leftarrow$ MATCHING(f, e)
    if $\phi \neq$ null **then**
      $\mathcal{D}_e(A) \leftarrow \mathcal{D}_e(A) \cup r[\mathcal{D}_{\phi(\mathrm{src}_f(e_1))}(\mathrm{lab}_f(e_1)), \ldots, \mathcal{D}_{\phi(\mathrm{src}_f(e_\ell))}(\mathrm{lab}_f(e_\ell))]$
      alt: $\mathcal{W}_e(A) \leftarrow \mathcal{W}_e(A) + \omega(r) \cdot \prod_{i \in [\ell]} \mathcal{W}_{\phi(\mathrm{src}_f(e_i))}(\mathrm{lab}_f(e_i))\}$

images in $g{\downarrow_e}$ is uniquely determined by f and the reentrancies in $g{\downarrow_e}$, due to reentrancy and order preservation. As proved in [5], this makes it furthermore possible to compute $\phi = \text{MATCHING}(f, e)$ in linear time.

As the correctness of the computation of the sets $\mathcal{D}_x(A)$ was essentially shown in [5], we take it for granted here and use it to show inductively that the weights are correctly computed. Below, we assume for the sake of technical simplicity that the operations of the semiring $\mathcal{S}$ are computable in constant time.

Theorem 2. *Let $\prec$ be a suitable family of orders, and let η be a function mapping graphs to $\mathbb{N}$ such that both $\eta(g)$ and $\prec_g$ can be computed in time $\eta(g)$.*[2] *Then there is an algorithm which takes as input a graph g and an OPHG grammar $G = (\Sigma, N, S, R, \omega)$, and computes $\omega_G(g)$ in time $\mathcal{O}(\eta(g) + |g|^2 + |G|^2)$.*

Proof. With straightforward reformulations, the proof of the main theorem in [5] shows that Algorithm 2 computes $\mathrm{DT}_G(g)$ and runs in time $\mathcal{O}(\eta(g) + |g|^2 + |G|^2)$ if the time required for the explicit construction of derivation trees is neglected.[3] Together with the assumption that the operations of $\mathcal{S}$ can be computed in

[2] The function η describes the complexity of computing $\prec_g$, and the condition that it can be executed in time $\eta(g)$ corresponds to the usual requirement of time constructibility.

[3] Instead of computing the sets $\mathcal{D}_x(A)$, the algorithm in [5] only computes, for every $x \in V_g \cup E_g$, the set of all $A \in N$ such that $\mathcal{D}_x(A) \neq \emptyset$.

constant time, the latter means that the weight-computing version of Algorithm 2 runs in time $\mathcal{O}(\eta(g) + |g|^2 + |G|^2)$ as well. To complete the proof, it thus suffices to prove by induction that Algorithms 1–4 maintain the invariant that $\mathcal{W}_x(A) = \sum_{t \in \mathcal{D}_x(A)} \omega(t)$ for those edges and nodes x and those $A \in N$ such that $\mathcal{D}_x(A) \neq \bot$.

In the proof, for a set D of derivation trees, we abbreviate $\sum_{t \in D} \omega(t)$ by $\omega(D)$. We check the algorithms one by one. Note that the induction hypothesis states that the equation $\mathcal{W}_x(A) = \omega(\mathcal{D}_x(A))$ holds when the respective procedure is entered, and we have to show that it still holds afterwards. We use the fact that, by distributivity, for every rule $r = (A \to f)$ of arity ℓ and all sets $D_1, \ldots, D_\ell$ of derivation trees, it holds that

$$\omega(r(D_1, \ldots, D_\ell)) = \omega(r) \cdot \prod_{i \in [\ell]} \omega(D_i). \tag{1}$$

Procedure SHALLOWPARSE*:* We have to show that the two lines in the body of the loop starting in line 8 maintain the invariant. These lines change only $\mathcal{D}_e(A)$ and $\mathcal{W}_e(A)$, and after those two lines we have

$$\begin{aligned}
\mathcal{W}_e(A) &= \sum_{r = (A \to B^{\bullet\bullet}) \text{ fits Sib}} \omega(r^\ell) \cdot \prod_{i \in [\ell]} \mathcal{W}_{s_i}(B) \\
&= \sum_{r = (A \to B^{\bullet\bullet}) \text{ fits Sib}} \omega(r^\ell) \cdot \prod_{i \in [\ell]} \omega(\mathcal{D}_{s_i}(B)) \\
&= \sum_{r = (A \to B^{\bullet\bullet}) \text{ fits Sib}} \omega(r^\ell \langle \mathcal{D}_{s_1}(B), \ldots, \mathcal{D}_{s_\ell}(B) \rangle) \quad \text{(by Equation 1)} \\
&= \omega(\mathcal{D}_e(A)).
\end{aligned}$$

Procedure PARSE*:* Only lines 6 and 7 affect some $\mathcal{D}_x(A)$ and $\mathcal{W}_x(A)$. These lines obviously preserve the invariant.

Procedure PARSE$_{\text{V}}$*:* As before, lines 3 and 4 respect the invariant. Concerning lines 11 and 12, note that the two versions of SHALLOWPARSE return $(A \mapsto \mathcal{D}_e(A))_{A \in N}$ and $(A \mapsto \mathcal{W}_e(A))_{A \in N}$, respectively, for some edge e. By induction hypothesis, $\mathcal{W}_e(A) = \omega(\mathcal{D}_e(A))$ for all $A \in N$, which completes the argument.

Procedure PARSE$_{\text{E}}$*:* Once more, lines 2 and 3 respect the invariant. Furthermore, if $D = \mathcal{D}_e(A)$ and $W = \mathcal{W}_e(A) = \omega(\mathcal{D}_e(A))$ before an execution of lines 7 and 8 then, after those two lines,

$$\begin{aligned}
\mathcal{W}_e(A) &= W + \omega(r) \cdot \prod_{i \in [\ell]} \mathcal{W}_{\phi(\mathrm{src}_f(e_i))}(\mathrm{lab}_f(e_i))\} \\
&= \omega(D) + \omega(r) \cdot \prod_{i \in [\ell]} \omega(\mathcal{D}_{\phi(\mathrm{src}_f(e_i))}(\mathrm{lab}_f(e_i))) \\
&= \omega(D) + \omega(r[\mathcal{D}_{\phi(\mathrm{src}_f(e_1))}(\mathrm{lab}_f(e_1)), \ldots, \mathcal{D}_{\phi(\mathrm{src}_f(e_\ell))}(\mathrm{lab}_f(e_\ell))] \\
&= \omega(\mathcal{D}_e(A)).
\end{aligned}$$

This completes the correctness proof of the theorem. □

As indicated before, it is worthwhile noticing that the first version of the parsing algorithm computes the set $\mathrm{DT}_G(g)$ in time $\mathcal{O}(\eta(g) + |g|^2 + |G|^2)$ if the sets $\mathcal{D}_x(A)$ are represented in a compact way as packed forests. This may be useful for further applications.

References

1. 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: Proc. 7th Linguistic Annotation Workshop, ACL 2013 (2013)
2. Bauderon, M., Courcelle, B.: Graph expressions and graph rewriting. Mathematical Systems Theory **20**, 83–127 (1987)
3. Björklund, H., Björklund, J., Ericson, P.: On the regularity and learnability of ordered DAG languages. In: Proc. 22nd International Conference on the Implementation and Application of Automata (CIAA'17). Lecture Notes in Computer Science, vol. 10329, pp. 27–39. Springer (2017)
4. Björklund, H., Drewes, F., Ericson, P.: Between a rock and a hard place – uniform parsing for hyperedge replacement DAG grammars. In: Dediu, A., Janoušek, J., Martín-Vide, C., Truthe, B. (eds.) Proc. 10th Intl. Conf. on Language and Automata Theory and Applications. Lecture Notes in Computer Science, vol. 9618, pp. 521–532 (2016)
5. Björklund, H., Drewes, F., Ericson, P., Starke, F.: Uniform parsing for hyperedge replacement grammars. Tech. Rep. UMINF 18.13, Umeå University, http://www8.cs.umu.se/research/uminf/index.cgi (2018), submitted for publication
6. Chiang, D., Andreas, J., Bauer, D., Hermann, K.M., Jones, B., Knight, K.: Parsing graphs with hyperedge replacement grammars. In: Proc. 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013), Volume 1: Long Papers. pp. 924–932 (2013)
7. Drewes, F., Habel, A., Kreowski, H.J.: Hyperedge replacement graph grammars. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations, chap. 2, pp. 95–162. World Scientific (1997)
8. Drewes, F., Hoffmann, B., Minas, M.: Predictive top-down parsing for hyperedge replacement grammars. In: Proc. 8th Intl. Conf. on Graph Transformation (ICGT'15). Lecture Notes in Computer Science (2015)
9. Drewes, F., Hoffmann, B., Minas, M.: Predictive shift-reduce parsing for hyperedge replacement grammars. In: de Lara, J., Plump, D. (eds.) Proc. 10th Intl. Conf. on Graph Transformation (ICGT'17). Lecture Notes in Computer Science, vol. 10373, pp. 106–122 (2017)
10. Gilroy, S., Lopez, A., Maneth, S.: Parsing graphs with regular graph grammars. In: Proc. 6th Joint Conf. on Lexical and Computational Semantics (*SEM 2017). pp. 199–208 (2017)
11. Habel, A.: Hyperedge Replacement: Grammars and Languages, Lecture Notes in Computer Science, vol. 643. Springer (1992)
12. Habel, A., Kreowski, H.J.: May we introduce to you: Hyperedge replacement. In: Proceedings of the Third Intl. Workshop on Graph Grammars and Their Application to Computer Science. Lecture Notes in Computer Science, vol. 291, pp. 15–26. Springer (1987)