Journal of Computer and System Sciences 118 (2021) 1–27

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Uniform parsing for hyperedge replacement grammars

Henrik Björklund [a], Frank Drewes [a,*], Petter Ericson [b,a], Florian Starke [c]

[a] Department of Computing Science, Umeå University, Sweden
[b] Digital and Cognitive Musicology Lab, École Polytechnique Fédérale de Lausanne, Switzerland
[c] Faculty of Computer Science, TU Dresden, Germany

ARTICLE INFO

Article history:
Received 24 April 2018
Accepted 26 October 2020
Available online 27 November 2020

Keywords:
Graph grammar
Hyperedge replacement
Uniform parsing
Complexity
Natural Language Processing
Meaning representation

ABSTRACT

It is well known that hyperedge-replacement grammars can generate NP-complete graph languages even under seemingly harsh restrictions. This means that the parsing problem is difficult even in the non-uniform setting, in which the grammar is considered to be fixed rather than being part of the input. Little is known about restrictions under which truly uniform polynomial parsing is possible. In this paper we propose a low-degree polynomial-time algorithm that solves the uniform parsing problem for a restricted type of hyperedge-replacement grammars which we expect to be of interest for practical applications.

1. Introduction

Hyperedge-replacement grammars (HR grammars, for short) are context-free graph grammars that were introduced in [3,18], see also [17,11]. They represent one of the two most successful formal models for the description of graph languages (the other being confluent node-replacement grammars), because of their favorable algorithmic and language-theoretic properties which closely resemble those of context-free string grammars. Unfortunately, the similarities between the string and graph cases fail to extend to one of the most important computational problems in the context of formal languages: the parsing problem. It has been known for a long time that even the non-uniform membership problem for context-free graph languages is intractable (unless $P \neq NP$). In particular, there are hyperedge replacement graph languages which are NP-complete [1,19]. Severe restrictions must be placed on the grammars in order to make at least non-uniform polynomial parsing possible. Early results in this regard can be found in [20,21,10]. In [20] the degree of the polynomial that bounds the running time varies with the language. The algorithm in [21], which considers only edge replacement, and its generalization to hyperedge replacement by [10] are cubic in the size of the input graph, but depend exponentially on the grammar if considered in a uniform setting. Moreover, the restrictions [21] and [10] placed on the considered graph languages are very strong, and it was shown in [9] that even a slight relaxation results in NP-completeness again. For these reasons, these parsing algorithms are mainly of theoretical interest.

In recent years the question of efficiently parsing hyperedge replacement languages received renewed interest, because hyperedge replacement was proposed as a suitable mechanism for describing sentence semantics in natural language processing, and in particular the abstract meaning representation (AMR) proposed in [2]. Regarding the use of hyperedge replacement in this application area, see [7]. The same paper described a general recognition algorithm together with a

* Corresponding author.
E-mail addresses: henrikb@cs.umu.se (H. Björklund), drewes@cs.umu.se (F. Drewes), petter.ericson@epfl.ch (P. Ericson), Florian.Starke@tu-dresden.de (F. Starke).

https://doi.org/10.1016/j.jcss.2020.10.002

detailed complexity analysis. Unsurprisingly, the running time of the algorithm is exponential even in the non-uniform case, one of the exponents being the maximum degree of nodes in the input graph. The same is true for the recent algorithm by [16] which implements parsing for so-called *regular graph grammars*.

Unfortunately, the node degree is one of the parameters one would ideally not wish to limit, since meaning representations do not have bounded node degree. Moreover, natural language processing often has to deal with algorithmic learning situations in which large corpora must be parsed and grammars adjusted in an iterative process. Thus, truly uniform polynomial-time solutions would be valuable, provided that the polynomials have a reasonably low degree and the restrictions on the grammars are "natural".

Parsing a graph G with respect to a given HR grammar $\mathcal{G}$ means to check whether there is a derivation tree in $\mathcal{G}$ that yields G. Thus, the task is to decompose G recursively into subgraphs that can be generated from the nonterminals of $\mathcal{G}$. Intuitively, the NP-completeness of the problem comes from the fact that a graph has exponentially many subgraphs. This is the main difference between graph and string parsing. In the latter case, the well-known dynamic programming approach by Cocke, Kasami, and Younger is efficient because a string has only quadratically many substrings. One way to achieve polynomial parsing in the graph case as well is to make sure that only polynomially many decompositions are possible candidates for well-formed derivation trees. In this paper we achieve this by imposing restrictions on $\mathcal{G}$ which guarantee that the overall shape of a suitable decomposition of G can be "read off" G itself. Intuitively, what remains is to check whether appropriate rules of $\mathcal{G}$ can be assigned to the vertices of this decomposition in order to turn it into a derivation tree.

An attempt at a set of restrictions serving this purpose was made in [5]. Motivated by the fact that meaning representations such as AMR are typically acyclic, HR grammars were considered that generate directed acyclic graphs. However, as acyclicity alone does not make parsing any easier additional conditions were placed on the form of the rules. In the present paper, we generalize the approach: the generated graphs may have cycles, the allowed rules are considerably more general, and the restrictions are fewer and formulated in an axiomatic way which allows for different concretizations. We impose two conditions on our grammars, called *reentrancy preservation* and *order preservation*. The latter is relative to an ordering of the nodes of input graphs that can be instantiated in different ways.

We expect our parsing algorithm to be useful for describing and processing languages of semantic graphs such as AMR. In fact, typical structures occurring in such graphs provided the starting point for the development of the restrictions proposed in this paper. Along with the formal development of the notions and results leading to our parsing algorithm, we try to illustrate the potential usefulness of our grammars by means of a small running example that stretches from Section 2 to Section 6. It shows how to generate a language of AMR-like semantic graphs by an HR grammar that satisfies our conditions for uniform polynomial parsing (Examples 2.4, 4.3 and 5.4, as well as Section 6.1).

Let us briefly describe the idea behind the restrictions we impose on HR grammars to make them efficiently parsable. When working with hyperedge replacement, a nonterminal hyperedge is a placeholder attached to a sequence of nodes. This placeholder will eventually be replaced by a subgraph that shares the attached nodes of the hyperedge (and only those) with the rest of the generated graph. One difficulty parsing has to face is that, after the replacement of a hyperedge, it may not be visible in the resulting graph which nodes the replaced hyperedge had been attached to. Reentrancy preservation, which is the first condition we describe in this paper, makes it possible to recover this set of nodes solely from the structure of the generated graph.

One difficulty remains: even if the attached nodes of a nonterminal hyperedge can uniquely be recovered, it may still be unclear in which order they had been attached to the hyperedge. This is what is avoided by the condition of order preservation. It ensures, for example, that a rule cannot replace a nonterminal hyperedge by another nonterminal hyperedge attached to the same nodes but in a different order.

Thanks to the two restrictions, we obtain a uniform parsing algorithm which is roughly quadratic in both the size of the grammar and that of the input graph.[1]

As a final note on related work, we mention here that another recent approach to efficient parsing for HR grammars was presented in [13,14], where predictive top-down and bottom-up parsers are proposed, generalizing techniques from compiler construction to the graph case. The approach thus differs from ours in that it yields a parser generator which, with only the grammar as input, constructs a quadratic parser for the specific language generated by that grammar. Provided that the grammar analysis can be performed in polynomial time (which depends on the exact variant of the parser generator used), this approach is thus uniformly polynomial as well.

The next section compiles the basic notions relevant to hyperedge replacement grammars. Section 3 and 4 define and study reentrancy and order preservation, respectively. Section 5 presents one possible concretization of our abstract notion of preserved orders. The parsing algorithm and the main result of this paper are presented in Section 6, and Section 7 concludes the paper.

[1] The exact running time depends on how efficiently the chosen order can be computed.

2. Preliminaries

The set of non-negative integers is denoted by $\mathbb{N}$. For $n \in \mathbb{N}$, $[n]$ denotes $\{1, \ldots, n\}$. Given a set S, S^* denotes the set of all finite sequences over S, and $S^{\circledast}$ denotes the set of non-repeating sequences in S^*, i.e. those sequences in which no element of S occurs twice. The empty sequence is denoted by ε, $S^+ = S^* \setminus \{\varepsilon\}$, and $S^{\oplus} = S^{\circledast} \setminus \{\varepsilon\}$. The length of a sequence $w \in S^*$ is denoted by $|w|$, and $[w]$ denotes the smallest subset A of S such that $w \in A^*$. The canonical extensions of a mapping $f\colon S \to T$ to S^* and to the powerset of S are denoted by f as well, i.e., $f(a_1 \cdots a_k) = f(a_1) \cdots f(a_k)$ for $a_1, \ldots, a_k \in S$, and $f(S') = \{f(a) \mid a \in S'\}$ for $S' \subseteq S$. A sequence $sw \in S^*$ with $s \in S$ may also be denoted by (s, w).

2.1. Ordering subsets of a set

As mentioned in the introduction, one of the prerequisites of our parsing algorithm is a way to order various subsets of the nodes of an input graph. Consider an arbitrary binary relation $\prec$ on a set S. Given a sequence $w = s_1 \ldots s_k \in S^*$, we say that w *is ordered by* $\prec$ if $s_i \prec s_{i+1}$ for all $i \in [k-1]$, and moreover, $s_i \prec s_j$ implies $i < j$ for all $i, j \in [k]$. We furthermore say that $\prec$ *orders* a subset $A \subseteq S$ if the elements of A can be arranged in a sequence w which is ordered by $\prec$. Clearly, if $\prec$ orders A, this sequence w is uniquely determined. In the following, we denote this sequence by $[\![A]\!]_{\prec}$ (provided that $\prec$ indeed orders A). Note that, for the sake of generality, we place no further restrictions on $\prec$, and it is thus neither necessarily an order on A (where it may lack transitivity) nor on S (where it may be otherwise entirely arbitrary).

2.2. Hypergraphs

Throughout this paper, we fix a countably infinite supply LAB of symbols called labels, such that every $\sigma \in \mathrm{LAB}$ has a unique *rank* $\mathrm{rank}(\sigma) \in \mathbb{N}$. Similarly, we fix countably infinite supplies $\mathcal{V}$ and $\mathcal{E}$ of vertices and hyperedges, respectively.

Definition 2.1 *(hypergraph).* A (directed hyperedge-labeled) *hypergraph* over $\Sigma \subseteq \mathrm{LAB}$ is a tuple $G = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext})$ with the following components:

- $V \subseteq \mathcal{V}$ and $E \subseteq \mathcal{E}$ are disjoint finite sets of *nodes* and *hyperedges*, respectively.
- The *attachment* $\mathrm{att}\colon E \to V^{\oplus}$ assigns to each hyperedge e a sequence of attached nodes. For $e \in E$ with $\mathrm{att}(e) = (v, w)$ we also denote v by $\mathrm{src}(e)$ and w by $\mathrm{tar}(e)$, calling them the *source* and the sequence of *targets* of e, respectively.
- The *labeling* $\mathrm{lab}\colon E \to \Sigma$ assigns a label to each hyperedge, 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 $\mathrm{ext}_G = (v, w)$, then we denote the node v by $\dot{G}$ and the sequence w of nodes by $G_{\cdot\cdot}$, respectively, and we impose the additional requirement that $\mathrm{src}(e) \notin [G_{\cdot\cdot}]$ for all $e \in E$.[2]

The *size* $|G|$ of G is $\sum_{e \in E} |\mathrm{att}(e)|$.[3]

Note that we forbid $\mathrm{att}(e)$ (for $e \in E$) to contain any node repeatedly. In the following, we simply call hyperedges edges and hypergraphs graphs. Our division of the attachment of every edge into a single source node and any number of target nodes is similar to that used in the literature on term (hyper)graphs. It makes it meaningful to speak about directed paths (defined below). Our graphs are, however, more general than term graphs in that we, for the moment, do not impose further structural conditions on them.

Throughout the paper, if the components of a graph G are not explicitly named, we denote them by V_G, E_G, att_G, etc. If the components of G *are* given explicit names (and thus the subscript is dropped) we extend this in the obvious way to derived notations, dropping the subscript even there. We furthermore use the notation $\mathrm{out}_G(v)$ to denote the set of all outgoing edges of a node $v \in V_G$, i.e., $\mathrm{out}_G(v) = \{e \in E_G \mid \mathrm{src}_G(e) = v\}$.

An *isomorphism* $h\colon G \to H$ is a pair of bijective mappings $(h_V\colon V_G \to V_H, h_E\colon E_G \to E_H)$ such that $\mathrm{att}_H \circ h_E = h_V \circ \mathrm{att}_G$, $\mathrm{lab}_H \circ h_E = \mathrm{lab}_G$, and $\mathrm{ext}_H = h_V(\mathrm{ext}_G)$. If such an isomorphism exists we write $G \equiv H$ and say that the graphs are *isomorphic*.

A *path* of length $k \in \mathbb{N}$ from $u \in V$ to $e \in E$ in G is a sequence $p = e_1 \cdots e_k \in E^+$ where $\mathrm{src}(e_1) = u$, $\mathrm{src}(e_{i+1}) \in [\mathrm{tar}(e_i)]$ for all $i \in [k-1]$, and $e_k = e$. If furthermore v is a node in $[\mathrm{tar}(e_k)]$ then pv is a path from u to v. Both p and pv *pass* the nodes $\mathrm{src}(e_2), \ldots, \mathrm{src}(e_k)$, and we say that p *contains* $e_1, \ldots, e_k$ as well as $\mathrm{src}(e_1), \ldots, \mathrm{src}(e_k)$, while pv additionally contains v. If $\mathrm{src}(e_1) \in [\mathrm{tar}(e_k)]$, the path is a *cycle*. We say that the path is a *source path* if $u = \dot{G}$.

A node v or an edge e is *reachable* from a node u if $u = v$ or there is a path from u to v or from u to e, respectively. We simply say that v and e are *reachable in* G if they are reachable from $\dot{G}$. If G is clear from the context we may just write

[2] Recall that $[G_{\cdot\cdot}]$ denotes the set of nodes occurring in $G_{\cdot\cdot}$.

[3] This simple definition of size is sufficient and appropriate for our purposes as the classes of grammars considered in the paper only generate connected hypergraphs, and by the definition of hypergraphs it holds that external nodes are pairwise distinct and $1 \leq |\mathrm{att}(e)| \leq |V|$ for all hyperedges e. Thus, $|V| \leq |G|$, $|E| \leq |G|$, and $|\mathrm{ext}| \leq |G|$.

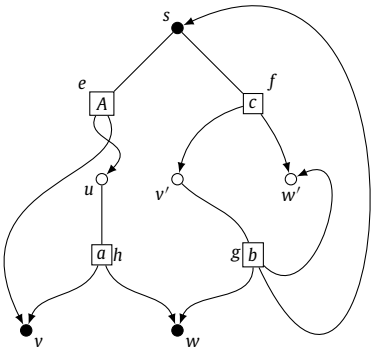

Fig. 1. Example drawing of a graph G.

"reachable" instead of "reachable in G". Note that, by definition, paths are always directed, and thus all of these notions refer to directed paths.

The *rank* of $G = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext})$ is $\mathrm{rank}(G) = |G_{\bullet\bullet}|$ and that of $e \in E$ is $\mathrm{rank}_G(e) = \mathrm{rank}(\mathrm{lab}(e))$. The *in-degree* of a node $u \in V$ is $|\{e \in E \mid u \in [\mathrm{tar}(e)]\}|$ and its *out-degree* is $|\{e \in E \mid \mathrm{src}(e) = u\}|$. A node of out-degree 0 is a *leaf*, and a node v of in-degree 0, such that every other node in V is reachable from v, is a *root*. Thus, the root of a graph is unique if it exists. If it does, we say that G is *rooted*. Note that, if the root is $\dot{G}$, then the whole graph G is also reachable. Note furthermore that, by our general condition on the sources of edges, all nodes in $G_{\bullet\bullet}$ are leaves. The reader should keep this fact in mind because we will occasionally make use of it without explicitly mentioning it.

For a label A of rank k, we let $A^\bullet$ denote the graph $(\{0, \ldots, k\}, \{e\}, \mathrm{att}, \mathrm{lab}, 0 \cdots k)$ such that $\mathrm{att}(e) = 0 \cdots k$, and $\mathrm{lab}(e) = A$.

2.3. Drawing conventions

We draw graphs as shown in Fig. 1: external nodes are depicted as bullets and non-external ones as circles. The node $\dot{G}$ is always the topmost bullet. An edge $e \in E_G$ is depicted as a box with the edge label inscribed, which can be dropped if it is not relevant. The attachment $\mathrm{att}_G(e)$ is indicated by a line drawn from $\mathrm{src}_G(e)$ to (the box representing) e, and arrows pointing from e to the nodes in $\mathrm{tar}_G(e)$. The arrows leave the box in the order in which they appear in $\mathrm{tar}_G(e)$, from left to right. Similarly, the nodes in $G_{\bullet\bullet}$ are arranged from left to right. For example, in the figure we have $\mathrm{tar}_G(e) = uv$, $\dot{G} = s$, and $G_{\bullet\bullet} = vw$.

2.4. Hyperedge replacement

Let H and F be graphs and $e \in E_H$ such that $V_H \cap V_F = [\mathrm{ext}_F]$, $E_H \cap E_F = \emptyset$, and $\mathrm{att}_H(e) = \mathrm{ext}_F$. The result of *substituting* e by F in H is the graph $G = H[e : F]$ such that $G = (V_H \cup V_F, (E_H \cup E_F) \setminus \{e\}, \mathrm{att}_G, \mathrm{lab}_G, \mathrm{ext}_H)$ with

$$\mathrm{att}_G(f) = \begin{cases} \mathrm{att}_H(f) & \text{if } f \in E_H \setminus \{e\} \\ \mathrm{att}_F(f) & \text{if } f \in E_F \end{cases} \qquad \mathrm{lab}_G(f) = \begin{cases} \mathrm{lab}_H(f) & \text{if } f \in E_H \setminus \{e\} \\ \mathrm{lab}_F(f) & \text{if } f \in E_F. \end{cases}$$

For graphs H and F and an edge $e \in E_H$ with $\mathrm{rank}_H(e) = \mathrm{rank}(F)$ it should be clear that we may always choose an isomorphic copy F' of F such that $H[e : F']$ is defined. To avoid the cumbersome technicalities of constantly having to deal with explicit isomorphisms, we shall therefore always assume that F itself fulfills the requirements. If it does not, it is assumed that F is silently replaced by an appropriate isomorphic copy. Note that this is possible by our assumption that neither attachments of edges nor the sequences of external nodes of graphs contain repetitions.

For the remainder of the paper, we assume that LAB is partitioned into two disjoint subsets $\mathrm{LAB_N}$ and $\mathrm{LAB_T}$, both countably infinite, whose elements are called *nonterminals* and *terminals*, respectively. Naturally, a terminal (nonterminal) edge is an edge labeled by a terminal (nonterminal, respectively). We sometimes just call them terminals and nonterminals if there is no danger of confusion. By convention, we use capital letters to denote nonterminals, and lowercase letters for terminal symbols. Furthermore, we refer to the graph H above, in which the replacement takes place, as the *host graph*.

Definition 2.2 *(hyperedge replacement grammar).* A *hyperedge replacement grammar* (HR grammar, for short) is a system $\mathcal{G} = (\Sigma, N, S, R)$ where $\Sigma \subseteq \mathrm{LAB_T}$, $N \subseteq \mathrm{LAB_N}$, $S \in N$ is the *initial nonterminal*, and R is a set of *rules*, also called HR rules. Each rule is of the form $A \to F$ where $A \in N$ and F is a graph over $\Sigma \cup N$ with $\mathrm{rank}(F) = \mathrm{rank}(A)$.

The *size* of $\mathcal{G}$ is $|\mathcal{G}| = \sum_{(A \to F) \in R} |F|$.

For graphs G, H, we let $H \Rightarrow_R G$ if there exist a rule $A \to F \in R$ and an edge $e \in E_H$ with $\mathrm{lab}(e) = A$ such that $G = H[e : F]$. As usual, $\Rightarrow_R^*$ denotes the reflexive transitive closure of $\Rightarrow_R$. If there is no danger of confusion we often write $\Rightarrow$

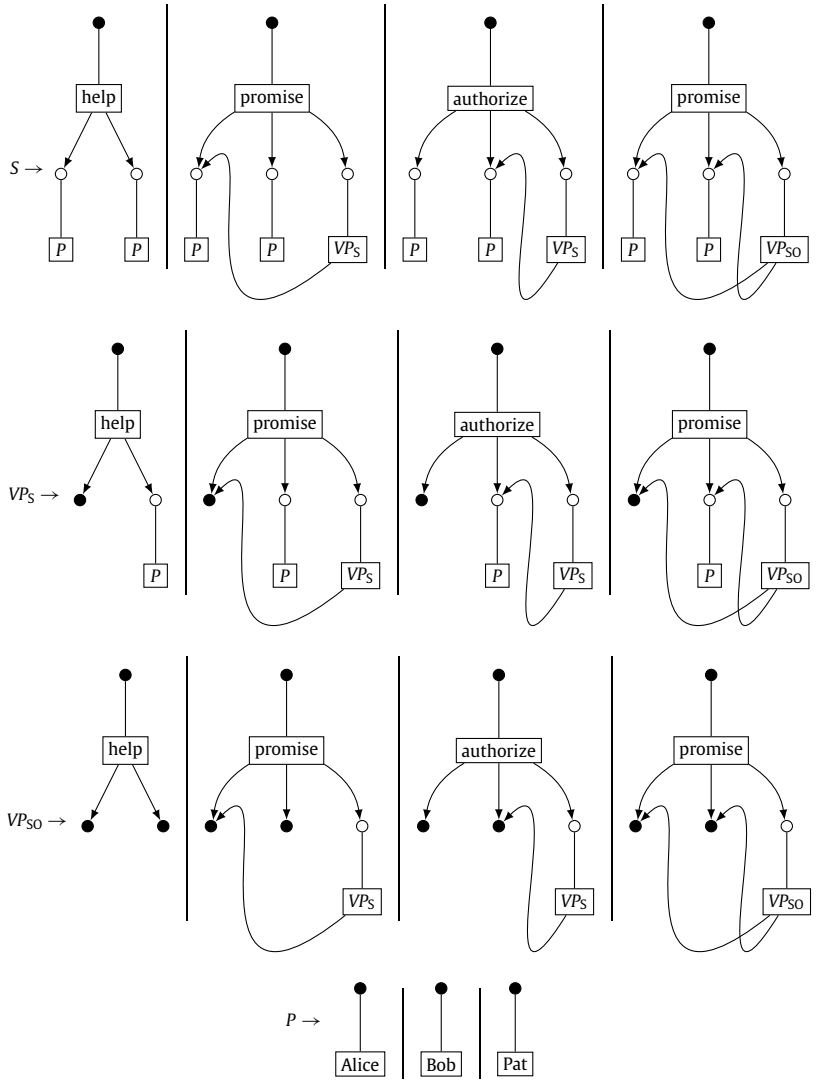

Fig. 2. Example rules of a HR grammar.

and $\Rightarrow^*$ instead of $\Rightarrow_R$ and $\Rightarrow_R^*$, respectively. The *language generated by $\mathcal{G}$ from $A \in \mathrm{LAB_N}$* is the set $\mathcal{L}_A(\mathcal{G})$ of all graphs G over Σ such that $A^\bullet \Rightarrow_R^* G$. The *language generated by $\mathcal{G}$* is $\mathcal{L}(\mathcal{G}) = \mathcal{L}_S(\mathcal{G})$.

For a given set $\mathcal{R}$ of HR rules (usually infinite), we let $\mathbb{G}_\mathcal{R}$ denote the set of all graphs G over LAB such that $A^\bullet \Rightarrow_\mathcal{R}^* G$ for some $A \in \mathrm{LAB_N}$.

Given pairwise distinct edges $f_1, \ldots, f_k \in E_F$ and graphs $G_1, \ldots, G_k$ such that $F[f_1 : G_1] \cdots [f_k : G_k]$ is defined, we may denote the latter by $F[f_1 : G_1, \ldots, f_k : G_k]$. We recall here the so-called context-freeness lemma of hyperedge-replacement grammars:

Lemma 2.3 *([17,11]). Let $\mathcal{G} = (\Sigma, N, S, R)$ be an HR grammar. The sets $\mathcal{L}_A(\mathcal{G})$ $(A \in N)$ are the smallest sets such that the following holds: for every rule $(A \to F) \in R$, if $f_1, \ldots, f_k$ are the nonterminal edges in F and $G_1 \in \mathcal{L}_{\mathrm{lab}_F(f_1)}(\mathcal{G}), \ldots, G_k \in \mathcal{L}_{\mathrm{lab}_F(f_k)}(\mathcal{G})$, then $F[f_1 : G_1, \ldots, f_k : G_k]$ is in $\mathcal{L}_A(\mathcal{G})$.*

Example 2.4. Fig. 2 shows some example rules of a HR grammar that generates semantic graphs akin to Abstract Meaning Representation. The nonterminals are S (of rank 0, being the initial nonterminal, and standing for *semantic graph*), VP_S (of rank 1, and standing for *verb phrase with specified subject*), VP_SO (of rank 2, and standing for *verb phrase with specified subject and object*), and finally P (of rank 0, standing for *person*).

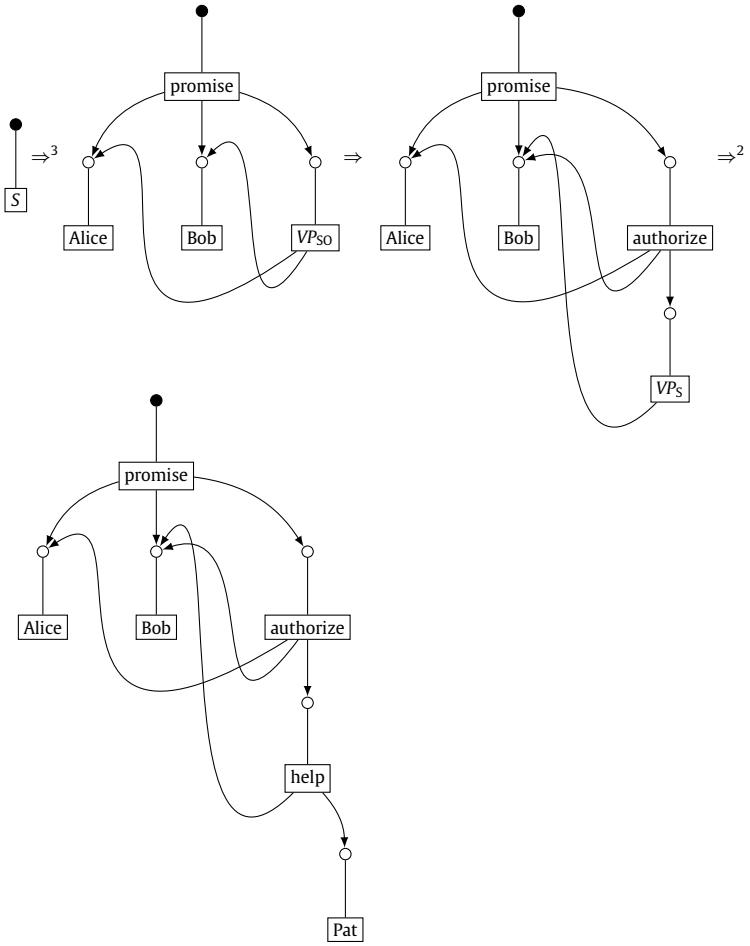

Fig. 3. Deriving the semantic graph of the sentence "Alice promises Bob to authorize him to help Pat" using the rules in Fig. 2.

For the terminal symbols "promise", "authorize", and "help", the first target represents the subject, the second one the object, and the third one (for the former two verbs), the thing which is promised or authorized to be done. To understand the rules, the reader should note that the act of promising exercises what is called subject control in linguistics: if X promises Y to, e.g., help someone, then X is not only the one doing the promising, but also the one doing the helping. Authorization is similar, but exercises object control instead: if X authorizes Y to help someone, then Y is the one doing the helping. Further terminal symbols are the ordinary verb "help"[4] as well as the proper names "Alice", "Bob", and "Pat".

For instance, the second and third rule for S pass the subject and object of "promise" and "authorize", respectively, to the nonterminal labeled VP_S in order to make it possible implement control using the rules for VP_S. The fourth rule has been added to show that anaphora can be implemented as well: In the sentence "Alice promises Bob to authorize him to help Pat", the pronoun "him" refers to Bob, showing that he is the one doing the helping (while subject control implies that Alice is the one providing the authorization). The reader should easily be able to add similar rules for further cases.

Fig. 3 illustrates a derivation of the semantic graph for the sentence mentioned above (where $\Rightarrow^k$ abbreviates a sequence of k derivation steps to replace k nonterminal edges in parallel).

3. Reentrancies

We now start to develop the notions and restrictions that lead to our parsing algorithm. This section focuses on reentrancies while the next section discusses suitable ways to order reentrant nodes.

Imagine starting at a node or edge x in a given graph G and collecting nodes that can be reached from there. Descending through G from x, we may first only encounter some nodes that cannot be reached in other ways, i.e., on source paths not containing x. However, typically we will eventually reach nodes that can *also* be reached on paths avoiding x, or are

[4] We use "help" in the sense "X helps Y" here. In general, it can also act as an object control verb in, e.g., "X helps Y to paint the wall."

external nodes of G (which, intuitively, are nodes that can be reached from outside G). These are the reentrant nodes of x. They determine the "fringe" of a subgraph F such that $G = H[f : F]$, where H is the graph G with F "cut out" of it and f is an edge whose targets are the reentrant nodes of x in G (and thus $F_{\bullet\bullet}$ consists of the reentrant nodes of x as well). The ambiguity inherent in this situation, caused by the fact that the reentrant nodes must be ordered in some way, will be dealt with in Section 4.

The definition below formalizes the notion of reentrant nodes.

Definition 3.1 *(reentrant nodes).* For a graph G and $E \subseteq E_G$, let $\mathrm{TAR}_G(E) = \bigcup_{e\in E}[\mathrm{tar}_G(e)]$ be the set of all targets of edges in E. For $x \in V_G \cup E_G$, let

$$\hat{x} = \begin{cases} x & \text{if } x \in V_G \\ \mathrm{src}_G(x) & \text{if } x \in E_G, \end{cases}$$

and let E^x_G be the set of all reachable edges $e \in E_G$ such that all source paths to e contain x. Then the set of *reentrant nodes of x in G* is

$$\mathrm{reent}_G(x) = (\mathrm{TAR}_G(E^x_G) \setminus \{\hat{x}\}) \cap (\mathrm{TAR}_G(E_G \setminus E^x_G) \cup [\mathrm{ext}_G]). \tag{1}$$

Note that $e \in E^e_G$ for all reachable $e \in E_G$, and $E^x_G = \mathrm{reent}_G(x) = \emptyset$ for all unreachable x. We will not overly concern ourselves with unreachable parts of G in the following, as for the substantial parts of this paper, only graphs are of interest in which all nodes and edges are reachable.

The reentrant nodes with respect to x are those which are targets of edges that can only be reached (from $\dot{G}$) via x and are at the same time targets of other edges (not only reachable through x) or are in $[\mathrm{ext}_G]$. As indicated above, the latter corresponds to the intuition that the external nodes are those nodes which can be reached "from outside G" and thus in particular by edges not in E^x_G.

Example 3.2 *(reentrant nodes).* Some examples of edge sets E^x_G and reentrancies $\mathrm{reent}_G(x)$ for a graph G and various choices of x are shown in Fig. 4:

(a) $E^d_G = \{d, e, f\}$ and thus $\mathrm{reent}_G(d) = \{t, w\}$, because t is external and w is a target of $h \notin E^d_G$.
(b) $E^u_G = E^e_G = \{e\}$ and $\mathrm{reent}_G(u) = \mathrm{reent}_G(e) = \{v, w\}$ because these nodes are also targets of d and h, respectively.
(c) $E^g_G = \{g, h\}$ and $\mathrm{reent}_G(g) = \{w\}$. Node s is not in $\mathrm{reent}_G(g)$ because $s = \hat{g}$.
(d) $E^{v'}_G = E^h_G = \{h\}$ and $\mathrm{reent}_G(v') = \mathrm{reent}_G(h) = \{w, w', s\}$ because $E^g_G = \{g\}$. Note that, in contrast to the preceding case, $s \in \mathrm{reent}_G(x)$ because $s \neq \hat{x}$.)
(e) $E^s_G = E_G$ and thus $\mathrm{reent}_G(s) = \{t, t', t''\}$ (not shown in the figure). This is true in general for $x = \dot{G}$ because of the assumption that all nodes are reachable from the root.

Lemma 3.3. *Let G be a graph with $x \in E_G \cup V_G$, and let $e \in E_G$ be reachable. Then $e \in E^x_G$ if and only if one of the following holds:*

1. $x \in \{e, \mathrm{src}_G(e)\}$ *or*
2. $\mathrm{src}_G(e) \neq \dot{G}$ *and all reachable edges $f \in E_G$ with $\mathrm{src}_G(e) \in [\mathrm{tar}_G(f)]$ are in E^x_G.*

Proof. For the *only if* direction, if $x \notin \{e, \mathrm{src}_G(e)\}$ and $\mathrm{src}_G(e) = \dot{G}$ then the source path e does not contain x and thus $e \notin E^x_G$. Thus, assume that $x \notin \{e, \mathrm{src}_G(e)\}$ and $\mathrm{src}_G(e) \neq \dot{G}$. Consider a reachable edge $f \in E_G$ with $\mathrm{src}_G(e) \in [\mathrm{tar}_G(f)]$ and assume, towards a contradiction, that $f \notin E^x_G$. Then there is a source path p to f not containing x. But then pe is a source path to e not containing x, and hence $e \notin E^x_G$.

We now prove the *if* statement. If $x \in \{e, \mathrm{src}_G(e)\}$, then all source paths to e contain x, and thus $e \in E^x_G$. Suppose now that $\mathrm{src}_G(e) \neq \dot{G}$. If all reachable edges $f \in E_G$ with $\mathrm{src}_G(e) \in [\mathrm{tar}_G(f)]$ are in E^x_G, then all source paths to e contain x as they pass one of those edges (because $\mathrm{src}_G(e) \neq \dot{G}$). Since e is reachable, it follows that $e \in E^x_G$. □

In the following, let $\approx$ be the binary relation on graphs such that $G \approx H$ if the two graphs are equal except that the order of nodes in $G_{\bullet\bullet}$ and $H_{\bullet\bullet}$ may differ. To be precise, $V_G = V_H$, $E_G = E_H$, $\mathrm{att}_G = \mathrm{att}_H$, $\mathrm{lab}_G = \mathrm{lab}_H$, $\dot{G} = \dot{H}$, and $[G_{\bullet\bullet}] = [H_{\bullet\bullet}]$. The following definition formalizes the notion of a subgraph rooted at an edge or a node. These subgraphs are uniquely determined up to $\approx$.

Definition 3.4 *(rooted subgraphs).* Let G be a graph and $x \in V_G \cup E_G$. The *subgraph $G{\downarrow_x}$ rooted at x* is a graph $H = (V, E, \mathrm{att}, \mathrm{lab}, \hat{x}w)$, where

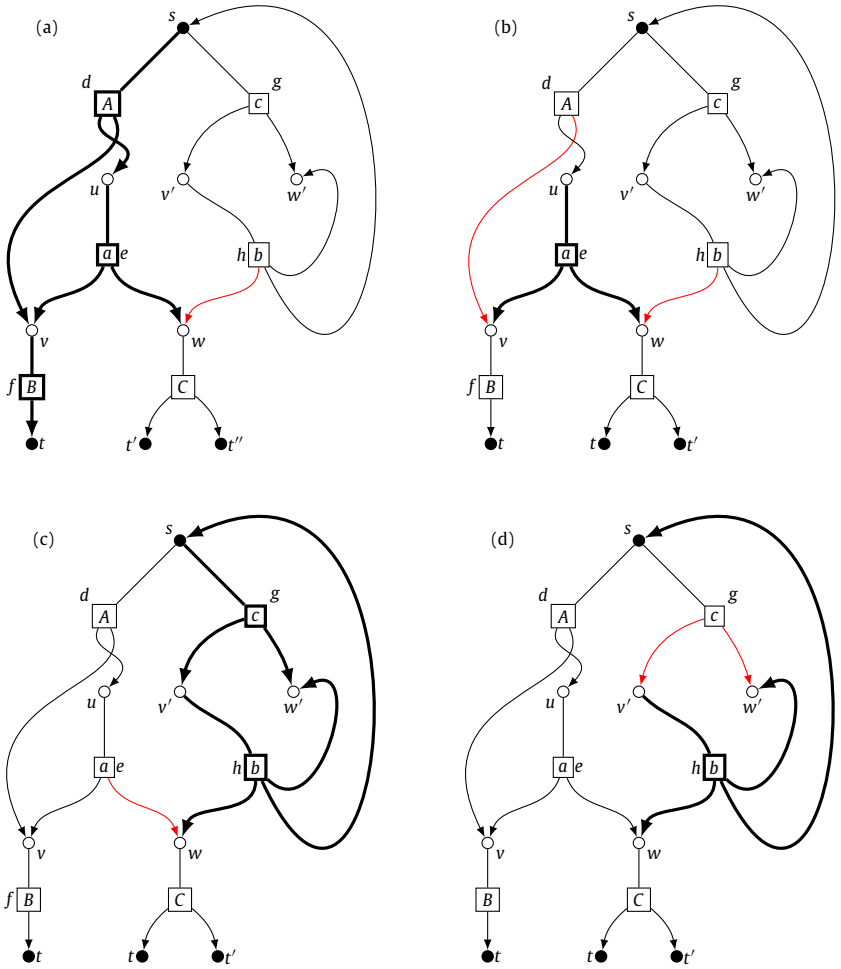

Fig. 4. Examples of edge sets E^x_G (drawn with thick lines) for $x = d$, $x \in \{u, e\}$, $x = g$, and $x \in \{h, v'\}$. Intuitively, starting from x, edges are collected following every path until such a path returns to $\hat{x}$ or reaches a node that is "blocked" by virtue of being external or having an incoming tentacle from the outside (drawn in red or, in a greyscale printing, in grey). The nodes at which the paths are blocked are the reentrant nodes with respect to x. Note that $\hat{x}$ is never considered to be reentrant even if it s a target of one of the edges in E^x_G as in (c). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

- $E = E^x_G$ and $V = \{\hat{x}\} \cup \mathrm{TAR}_G(E)$,
- att and lab are the restrictions of att_G and lab_G to E, and
- $[w] = \mathrm{reent}_G(x)$.

Thus, $G{\downarrow_x}$ is uniquely determined up to $\approx$. We assume in the following that $G{\downarrow_x}$ denotes an arbitrarily chosen element of the corresponding equivalence class of $\approx$.[5]

A slight simplification of the definition of reentrant nodes that is easier to handle in some proofs is

$$\mathrm{ree}_G(x) = \mathrm{TAR}_G(E^x_G) \cap (\mathrm{TAR}_G(E_G \setminus E^x_G) \cup [\mathrm{ext}_G]). \tag{2}$$

Obviously, $\mathrm{reent}_G(x) = \mathrm{ree}_G(x) \setminus \{\hat{x}\}$. Hence, in order to establish equations such as $\mathrm{reent}_G(x) = \mathrm{reent}_H(x)$ it is sufficient (but not necessary) to show that $\mathrm{ree}_G(x) = \mathrm{ree}_H(x)$. Thus, we will frequently show that $\mathrm{reent}_G(x) = \mathrm{reent}_H(x)$ by establishing that $\mathrm{ree}_G(x) = \mathrm{ree}_H(x)$ as the latter relieves us from considering $\hat{x}$ as a special case.

We conclude this section by stating and proving a lemma that essentially says that if y belongs to $G{\downarrow_x}$, then its rooted subgraph in G is the same as in $G{\downarrow_x}$. This will be important for the correctness proof of our parsing algorithm.

Lemma 3.5. *Let G be a graph, $H = G{\downarrow_x}$ for some $x \in V_G \cup E_G$. Then $H{\downarrow_y} \approx G{\downarrow_y}$ for all $y \in (V_H \cup E_H) \setminus [\mathrm{ext}_H]$.*

[5] For unreachable $x \in V_G \cup E_G$, $G{\downarrow_x}$ is the graph consisting of the single external node $\hat{x}$ and no edges.

Proof. The statement is trivially true for $x \in [G_{\cdot\cdot}]$, because these nodes have no outgoing edges, which means that $G{\downarrow_x}$ is the graph consisting of x only. Hence, for the remainder of the proof let $x \notin [G_{\cdot\cdot}]$.

Let us first assume that x and y are both edges and that x is reachable. (As $G{\downarrow_x}$ is a single external node for unreachable x, the lemma is trivially true if x is not reachable.) By Definition 3.4 it suffices to show that

(i) $E_H^y = E_G^y$ and
(ii) $\mathrm{ree}_H(y) = \mathrm{ree}_G(y)$.

We distinguish two sub-cases.

Case 1: $x = y$. Then y is the unique edge in E_H whose source is $\dot{H}$, as all other edges (in E_G) sharing that source can obviously be reached on source paths in G not containing x.

Moreover, as all edges in E_G^x are reachable only through x, all edges in E_H are reachable in H, and all source paths (in H) pass $x = y$, meaning $E_H^y = E_H$. Consequently, $E_H^y = E_H = E_G^x = E_G^y$, completing (i).

For (ii), it suffices to note that

$$\begin{aligned}
\mathrm{ree}_H(y) &= \mathrm{TAR}_H(E_H) \cap [\mathrm{ext}_H] && \text{since } E_H^y = E_H \text{ and thus } E_H \setminus E_H^y = \emptyset \\
&= \mathrm{TAR}_G(E_G^x) \cap (\{\hat{x}\} \cup \mathrm{ree}_G(x)) && \text{by definition of } H = G{\downarrow_x} \\
&= \mathrm{ree}_G(x) \\
&= \mathrm{ree}_G(y).
\end{aligned}$$

Case 2: $x \neq y$. To prove (i), consider first an edge $e \in E_H^y \subset E_G^x$. There is a source path to y in G, and from there to e. Thus, $e \notin E_G^y$ only if e is also reachable in G on a source path not containing y. Then that path contains x (because $e \in E_G^x$), and its sub-path p from x to e cannot be a path in H because all those paths do contain y. Thus $p = p_1 e' p_2$ for some edge $e' \notin E_H = E_G^x$, i.e., e' is reachable on a source path q in G that does not contain x. However, then qep_2 is a source path to e in G, and it does not contain x, which contradicts the assumption that $e \in E_G^x$.

Conversely, for an edge $e \in E_G^y$, all source paths to e in G contain y, and hence they all contain x as well because $y \in E_G^x$. Moreover, at least one such path exists. Thus, $e \in E_G^x = E_H$. Clearly, H cannot contain more paths than G, which shows that all source paths to e in H contain y. It remains to show that at least one such path exists. However, we know that there is a source path to e in G that contains x, i.e., it has a sub-path starting at x. By the same reasoning as in the previous paragraph, this sub-path is a path in H because otherwise there would be a source path to e in G that does not contain x. Hence $e \in E_H^y$, completing the proof of (i).

We now prove (ii), i.e., $\mathrm{ree}_H(y) = \mathrm{ree}_G(y)$ (still for the case where $x, y \in E_G$ and $x \neq y$).

($\mathrm{ree}_H(y) \subseteq \mathrm{ree}_G(y)$) We have to show that

$$\begin{aligned}
&\mathrm{TAR}_H(E_H^y) \cap (\mathrm{TAR}_H(E_H \setminus E_H^y) \cup [\mathrm{ext}_H]) \\
&\subseteq \mathrm{TAR}_G(E_G^y) \cap (\mathrm{TAR}_G(E_G \setminus E_G^y) \cup [\mathrm{ext}_G]).
\end{aligned}$$

We already know that $E_H^y = E_G^y$ and hence $\mathrm{TAR}_H(E_H^y) = \mathrm{TAR}_G(E_G^y)$. Thus, it remains to be shown that $\mathrm{TAR}_H(E_H \setminus E_H^y) \cup [\mathrm{ext}_H] \subseteq \mathrm{TAR}_G(E_G \setminus E_G^y) \cup [\mathrm{ext}_G]$. Since $\mathrm{TAR}_H(E_H \setminus E_H^y) = \mathrm{TAR}_G(E_H \setminus E_G^y) \subseteq \mathrm{TAR}_G(E_G \setminus E_G^y)$, it only needs to be verified that $([\mathrm{ext}_H] \cap \mathrm{TAR}_H(E_H^y)) \setminus [\mathrm{ext}_G] \subseteq \mathrm{TAR}_G(E_G \setminus E_G^y)$, but this is clear because

$$\begin{aligned}
([\mathrm{ext}_H] \cap \mathrm{TAR}_H(E_H^y)) \setminus [\mathrm{ext}_G] &= \mathrm{ree}_G(x) \setminus [\mathrm{ext}_G] \\
&\subseteq (\mathrm{TAR}_G(E_G \setminus E_G^x) \cup [\mathrm{ext}_G]) \setminus [\mathrm{ext}_G] \\
&\subseteq \mathrm{TAR}_G(E_G \setminus E_G^x) \\
&\subseteq \mathrm{TAR}_G(E_G \setminus E_G^y).
\end{aligned}$$

($\mathrm{ree}_G(y) \subseteq \mathrm{ree}_H(y)$) Consider a node $v \in \mathrm{ree}_G(y)$. We already know that $v \in \mathrm{TAR}_G(E_G^y) = \mathrm{TAR}_H(E_H^y)$, so we need to verify that $v \in \mathrm{TAR}_H(E_H \setminus E_H^y) \cup [\mathrm{ext}_H]$. If $v \in [\mathrm{ext}_G]$ then there is nothing left to show, because $\mathrm{ree}_G(y) \cap [\mathrm{ext}_G] \subseteq [\mathrm{ext}_H]$. For $v \in \mathrm{ree}_G(y) \setminus [\mathrm{ext}_G]$ we get

$$\begin{aligned}
&v \in \mathrm{TAR}_G(E_G^y) \cap \mathrm{TAR}_G(E_G \setminus E_G^y) \\
&= \mathrm{TAR}_H(E_H^y) \cap \mathrm{TAR}_G(E_G \setminus E_H^y) \\
&= \mathrm{TAR}_H(E_H^y) \cap \mathrm{TAR}_H(E_H \setminus E_H^y) && (\text{since } E_G \cap E_H^y \subseteq E_H) \\
&\subseteq \mathrm{TAR}_H(E_H \setminus E_H^y),
\end{aligned}$$

as required.

This finishes the reasoning for the case where x, y are edges. To complete the proof, consider the case where at least one of x, y is a node. If $x = \dot{G}$, $y = \dot{G}$, or $x = y$ we obviously have $G{\downarrow_x} = G$, $H{\downarrow_y} = G{\downarrow_x}$, or $H{\downarrow_y} = H$, respectively, and there is nothing to show. Hence, assume that $\{x, y\} \cap [\mathrm{ext}_G] = \emptyset$ and $x \neq y$. Let $\overline{G}$ be the graph obtained from G by doing the following for every node $v \in V_G \setminus [G_{\bullet\bullet}]$:

- add a fresh node $\overline{v}$ and an edge e_v with $\mathrm{att}_{\overline{G}}(e_v) = v\overline{v}$ (the label of e_v does not matter), and
- for every edge $e \in E_G$ with $\mathrm{src}_G(e) = v$, define $\mathrm{src}_{\overline{G}}(e) = \overline{v}$.

The remaining components of $\overline{G}$, including the target attachments of edges, are inherited from G. The graph $\overline{H}$ is defined similarly. Now, since e_v is the unique outgoing edge of v, it holds that $\overline{G{\downarrow_{e_v}}} \approx \overline{G{\downarrow_v}}$, and similarly $\overline{H{\downarrow_{e_v}}} \approx \overline{H{\downarrow_v}}$ for nodes $v \in V_H \setminus [H_{\bullet\bullet}]$. Consequently, the first part of the proof shows that $\overline{H{\downarrow_y}} \approx \overline{G{\downarrow_y}}$. As the mapping $\overline{\ }$ is injective, this yields the result. □

4. Order-preserving hyperedge replacement grammars

Our aim in this section is to define a notion of order-preserving grammars that generalizes the type of HR grammars introduced in [5] and also studied in [4].

The purpose of restricting HR grammars in this way is to make polynomial uniform parsing possible. We achieve this by making sure that derivable graphs can efficiently be decomposed in a way compatible with hyperedge replacement in a way that can be used to guide the parsing process.

We start out with a class of HR rules that satisfy some structural requirements which make it possible to exploit the findings of the preceding section. Such rules are called *reentrancy preserving*. Next, we define the notion of a *suitable family of orders*. Finally, we define what it means for a set of HR rules to be *order preserving* for such a family of orders. The requirement is essentially that an HR replacement does not alter the relative order of any nodes, neither in the host graph nor in the right-hand side inserted into it.

Before giving the definition of reentrancy preservation, we define a type of rule that forms a special case among the reentrancy-preserving ones, the so-called duplication rule.

Definition 4.1. Consider a graph

$$F = (\{v_0, \ldots, v_n\}, \{e, e'\}, \mathrm{att}, \mathrm{lab}, v_0 v_{i_1} \cdots v_{i_k}),$$

where $\mathrm{att}(e) = v_0 \cdots v_n = \mathrm{att}(e')$, $\mathrm{lab}(e) = \mathrm{lab}(e') \in \mathrm{LAB_N}$, and $i_1 < \cdots < i_k$. If $k < n$ then F (and every graph isomorphic to F) is a *twin*, and if $k = n$ 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 $\mathrm{lab}(e) = \mathrm{lab}(e') = A$. A *duplication rule* is either a clone or a twin rule.

Note that the right-hand side of a clone rule is uniquely determined by the left-hand side. A clone rule simply duplicates the nonterminal edge it is applied to, whereas a twin rule replaces a nonterminal edge by two "twins" having some additional targets (and, therefore, a different label).

Definition 4.2 *(reentrancy-preserving rule).* An HR rule $A \to F$ is *reentrancy preserving* if it is a duplication rule, or if F satisfies the following conditions:

(P1) all nodes in V_F are reachable,
(P2) the out-degree of every node is at most 1,
(P3) for every nonterminal edge e, $\mathrm{reent}_F(e) = [\mathrm{tar}_F(e)]$.

Example 4.3 *(Example 2.4 cont'd).* The rules in Example 2.4 are all reentrancy preserving. Conditions (P1) and (P2) are obviously satisfied. Condition (P3) is also satisfied because all targets of nonterminal edges in each right-hand side F are either targets of the outgoing edge of $\dot{F}$ or they are in $[F_{\bullet\bullet}]$.

To illustrate the use of duplication rules (more precisely: of cloning rules), let us assume we want to be able to model the meaning of sentences such as "Alice promises to encourage, support, and defend Bob". Thus, all three verbs share their subjects and objects. We can model this by deriving VP_{SO} into an edge labeled "and", with a cloneable nonterminal VP^{+}_{SO} beneath it that can be cloned into any number of VP_{SO}-labeled edges. The required additional rules are shown in Fig. 5. Fig. 6 shows a derivation of the semantic graph of the aforementioned sentence.

We denote the class of all reentrancy-preserving HR rules by $\mathcal{C}$, and thus the set of graphs that can be generated from $A^{\bullet}$ with $A \in \mathrm{LAB_N}$ using rules in $\mathcal{C}$ by $\mathbb{G}_{\mathcal{C}}$. Before discussing node orderings, let us study a few immediate properties of reentrancy-preserving rules and the graphs they generate.

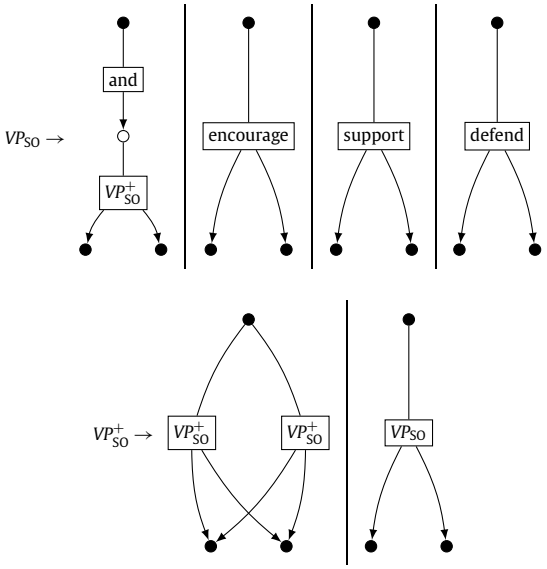

Fig. 5. Adding rules to those of Example 2.4 to make use of cloning.

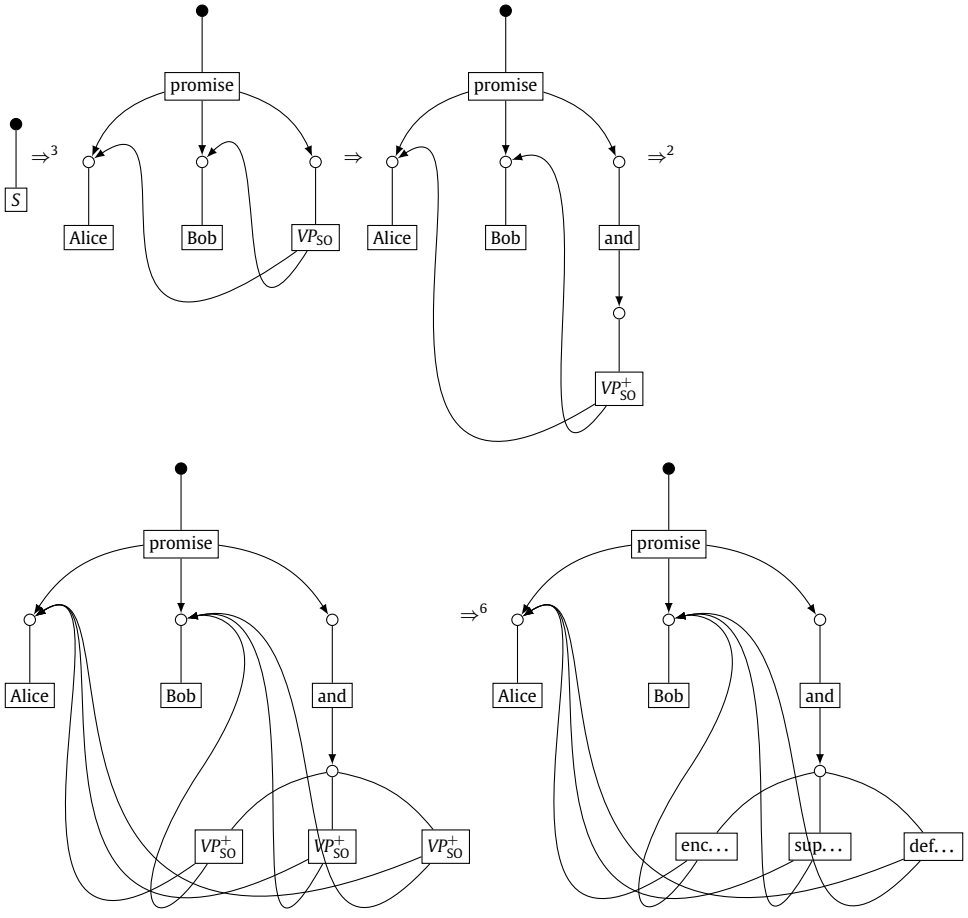

Fig. 6. Deriving the semantic graph of the sentence “Alice promises Bob to encourage, support, and defend him” using the rules in Figs. 2 and 5.

First of all, note that all graphs $A^\bullet$ satisfy (P1)–(P3). Moreover, applying a reentrancy-preserving HR rule to a graph that satisfies (P1) and (P3) preserves these two properties. Thus, it follows by induction on the length of derivations that all graphs in $\mathbb{G}_{\mathcal{C}}$ satisfy (P1) and (P3) (but not necessarily (P2), owing to the existence of duplication rules).

Lemma 4.4 *(reentrancy preservation). Let $G = H[e:F]$, where $H \in \mathbb{G}_{\mathcal{C}}$ and the rule $(\mathrm{lab}_H(e) \to F) \in \mathcal{C}$. For all $x \in E_G \cup V_G$ we have*

$$\mathrm{reent}_G(x) = \begin{cases} \mathrm{reent}_H(x) & \text{if } x \in E_H \cup V_H \\ \mathrm{reent}_F(x) & \text{if } x \in E_F \cup V_F \setminus [\mathrm{ext}_F] \end{cases}$$

We prove the two cases of Lemma 4.4 by establishing a lemma for each, i.e., Lemma 4.4 is the conjunction of Lemmas 4.5 and 4.6 proved next.

Lemma 4.5. *Let $G = H[e:F]$, where $H \in \mathbb{G}_{\mathcal{C}}$ and $(\mathrm{lab}_H(e) \to F) \in \mathcal{C}$. For all $x \in (E_H \cup V_H) \setminus \{e\}$ it holds that $\mathrm{reent}_G(x) = \mathrm{reent}_H(x)$.*

Proof. Observe first that

$$E_G^x = \begin{cases} E_H^x \setminus \{e\} \cup E_F & \text{if } e \in E_H^x \\ E_H^x & \text{otherwise.} \end{cases} \tag{3}$$

This is because all nodes (and thus all edges) in F are reachable from $\dot{F}$ by (P1), and thus every source path in H can be converted into a source path in G by substituting a suitable source path in F for each occurrence of e, and vice versa every source path in G to $e' \in E_G$ can be converted into a source path in H to e' if $e' \neq e$ and to e otherwise, by substituting e for every maximal sub-path which is a path in F.

By the definition of hyperedge replacement, we have

$$\mathrm{TAR}_G(E_F) \cap \mathrm{TAR}_G(E_G \setminus E_F) \subseteq [\mathrm{ext}_F] = [\mathrm{att}_H(e)].$$

Thus, by equation (3), no node in $\mathrm{TAR}(E_F) \setminus [\mathrm{ext}_F]$ belongs to both $\mathrm{TAR}_G(E_G^x)$ and $(\mathrm{TAR}_G(E_G \setminus E_G^x) \cup [\mathrm{ext}_G])$, i.e., to $\mathrm{ree}_G(x)$. In other words, among the nodes in V_F only the external nodes of F can be reentrant for x in G: $\mathrm{ree}_G(x) \cap V_F \subseteq [\mathrm{ext}_F]$. Only nodes in V_H could thus potentially violate the equality $\mathrm{reent}_G(x) = \mathrm{reent}_H(x)$. Hence, as $\hat{x}$ is in neither $\mathrm{reent}_G(x)$ nor $\mathrm{reent}_H(x)$, it remains to show that $v \in \mathrm{ree}_G(x) \iff v \in \mathrm{ree}_H(x)$ for all $v \in V_H \setminus \{\hat{x}\}$.

Recall that

$$\mathrm{ree}_G(x) = \mathrm{TAR}_G(E_G^x) \cap (\mathrm{TAR}_G(E_G \setminus E_G^x) \cup [\mathrm{ext}_G])$$
$$\mathrm{ree}_H(x) = \mathrm{TAR}_H(E_H^x) \cap (\mathrm{TAR}_H(E_H \setminus E_H^x) \cup [\mathrm{ext}_H]).$$

Note that, by the definition of hyperedge replacement, we always have $[\mathrm{ext}_G] = [\mathrm{ext}_H]$.

We first consider the case when $e \notin E_H^x$ and thus $E_G^x = E_H^x$ and $E_G^x \cap E_F = \emptyset$. Then the left arguments of the intersections defining $\mathrm{ree}_G(x)$ and $\mathrm{ree}_H(x)$ are identical, i.e., $\mathrm{TAR}_G(E_G^x) = \mathrm{TAR}_H(E_H^x) \subseteq V_H$. Thus, since $[\mathrm{ext}_G] = [\mathrm{ext}_H]$ we need to show that $v \in \mathrm{TAR}_G(E_G \setminus E_H^x)$ if and only of $v \in \mathrm{TAR}_H(E_H \setminus E_H^x)$ for all $v \in V_H \setminus [\mathrm{ext}_H]$.

By the definition of hyperedge replacement, all edges in $E_H \setminus \{e\}$ keep their targets in G. Thus, only nodes $v \in [\mathrm{att}_H(e)]$ could potentially violate the equality, which yields two cases: either $v \in \mathrm{tar}_H(e) \setminus \mathrm{TAR}_F(E_F)$, which is prevented by (P1), or $v = \mathrm{src}_H(e)$. However, as $e \notin E_H^x$ (by assumption) and $v \notin [\mathrm{ext}_H]$, we know that $\mathrm{src}_H(e) \in \mathrm{TAR}_H(E_H \setminus E_H^x)$, as required.

We next consider the case when $e \in E_H^x$ and thus $E_G^x = E_H^x \setminus \{e\} \cup E_F$. In this case, we have $\mathrm{TAR}_H(E_H^x) \subseteq \mathrm{TAR}_G(E_G^x)$, but also $\mathrm{TAR}_H(E_H \setminus E_H^x) = \mathrm{TAR}_G(E_G \setminus E_G^x)$, due to the definition of hyperedge replacement. This makes the right arguments of the intersections defining $\mathrm{ree}_G(x)$ and $\mathrm{ree}_H(x)$ identical.

Thus, it remains to show that $v \in \mathrm{TAR}_G(E_G^x)$ if and only if $v \in \mathrm{TAR}_H(E_H^x)$ for the relevant cases. Once again, this boils down to whether there can be a node v that is a target of e but not of any edge in E_F, or a target of an edge in E_F but not of e, and the only discrepancy that may occur is if $v = \mathrm{src}_H(e)$ and $\dot{F} \in \mathrm{TAR}_F(E_F)$. However, with $e \in E_H^x$, the only case in which $\mathrm{src}_H(e)$ may be an element of the second but not the first argument of the intersection defining $\mathrm{ree}_H(x)$ is the case $x = e$, which is excluded by the assumptions in the statement of the lemma. □

Lemma 4.6. *Let $G = H[e:F]$, where $H \in \mathbb{G}_{\mathcal{C}}$ and $(\mathrm{lab}_H(e) \to F) \in \mathcal{C}$. For all $x \in E_F \cup V_F \setminus [\mathrm{ext}_F]$ it holds that $\mathrm{reent}_G(x) = \mathrm{reent}_F(x)$.*

Proof. As e is nonterminal and H belongs to $\mathbb{G}_{\mathcal{C}}$ and thus satisfies (P3), $\mathrm{ree}_H(e) = [\mathrm{tar}_H(e)]$. Hence, every node $v \in [\mathrm{att}_H(e)] = [\mathrm{ext}_F]$ is reachable on a source path in H not containing e, or it is in $[\mathrm{ext}_H]$. Thus, in G, v is reachable on a source path not containing any edge in E_F, or it is in $[\mathrm{ext}_G]$. In particular, $E_F^x = E_G^x$ and thus $E_G^x \cap (E_G \setminus E_F) = \emptyset$ for all $x \in E_F \cup V_F \setminus [\mathrm{ext}_F]$.

This further means that $\mathrm{TAR}_G(E_G^x) \subseteq V_F$, and as we previously established that all nodes in $[\mathrm{ext}_F]$ have source paths not passing edges in E_F or are in $[\mathrm{ext}_G]$, and are thus contained in the second argument of the intersection defining $\mathrm{ree}_G(x)$, the lemma follows from the observation that $\mathrm{att}_G(f) = \mathrm{att}_F(f)$ for all edges $f \in E_F$. □

We now formalize the notion of a suitable family of orders. These do not actually have to be orders in the mathematical sense, but are binary relations on the node set of every graph that, in an order-preserving HR grammar, are required to order the target nodes of nonterminal edges and of all right-hand sides. We thus call these relations orders to support the intuition that this is what they are used for.

Definition 4.7 *(suitable family of orders).* A family $\prec = (\prec_G)_{G\in\mathbb{G}_\mathcal{C}}$, where each $\prec_G$ is a binary relation on V_G, is a *suitable family of orders* if the following hold:

(S1) For all $A \in \mathrm{LAB_N}$, $A^\bullet_{\cdot\cdot}$ is ordered by $\prec_{A^\bullet}$.[6]
(S2) For $G, G' \in \mathbb{G}_\mathcal{C}$, if $G' \equiv G$ via an isomorphism $h\colon G \to G'$ then for all $u, v \in V_G$ we have $u \prec_G v$ if and only if $h_V(u) \prec_{G'} h_V(v)$.[7]

We are now ready to define our notion of order preservation.

Definition 4.8 *(order-preserving).* Let $\prec = (\prec_G)_{G\in\mathbb{G}_\mathcal{C}}$ be a suitable family of orders. A set $\mathcal{R} \subseteq \mathcal{C}$ of HR rules *preserves* $\prec$ if, for all $G = H[e:F]$ with $H \in \mathbb{G}_\mathcal{R}$, $e \in E_H$, and $(\mathrm{lab}_H(e) \to F) \in \mathcal{R}$, we have $\prec_G|_{V_H} = \prec_H$ and $\prec_G|_{V_F} = \prec_F$.[8]

From now on, let $(\prec_G)_{G\in\mathbb{G}_\mathcal{C}}$ be a suitable family of orders which is preserved by a set $\mathcal{R} \subseteq \mathcal{C}$ of HR rules. We shall without loss of generality assume that each label in $\mathrm{LAB_N}$ occurs among the left-hand sides of rules in $\mathcal{R}$, since all other nonterminals can be removed from $\mathrm{LAB_N}$ (and the rules whose right-hand sides contain such labels can be removed from $\mathcal{R}$) without changing $\mathbb{G}_\mathcal{R}$. With this we get the following observation as a consequence of (S1) and Definition 4.8 (additionally using (S2)):

Observation 4.9. *For every rule $A \to F$ in $\mathcal{R}$, $F_{\cdot\cdot}$ is ordered by $\prec_F$, and so is $\mathrm{tar}_F(e)$ for every nonterminal edge $e \in E_F$.*

The first property follows from (S1) by choosing $H = A^\bullet$ in Definition 4.8, and the second follows by choosing $G = F[e:F']$ with $(\mathrm{lab}_F(e) \to F') \in \mathcal{R}$, applying the first property to F' and then using Definition 4.8 twice.

5. An example of a suitable family of orders

We now present a family $\lhd$ of orders and a set $\mathcal{R} \subseteq \mathcal{C}$ of rules that preserves $\lhd$. In particular, this generalizes the order and set of rules used in [5].

Definition 5.1 *(path order).* Let G be a graph.

(1) For $e \in E_G$, we define a relation $\lhd^e_G$ on $\mathrm{reent}_G(e)$. Assume that $\mathrm{tar}_G(e) = v_1 \cdots v_k$ for some nodes $v_1, \ldots, v_k \in V_G$. For all $i \in [k]$, let V_i be the set of nodes in $\mathrm{reent}_G(e)$ which are reachable from v_i in $G{\downarrow_e}$ on a path not containing e. Then, for $u, v \in \mathrm{reent}_G(e)$, we let $u \lhd^e_G v$ if and only if $\min\{i \in [k] \mid u \in V_i\} < \min\{i \in [k] \mid v \in V_i\}$.
(2) The *path order* $\lhd_G$ *on* V_G is defined as $\lhd_G = \bigcup_{e\in E_G} \lhd^e_G$.

Lemma 5.2 *($\lhd$ is suitable). The family $\lhd$ fulfills conditions (S1) and (S2), and is thus a suitable family of orders.*

Proof. For $G = A^\bullet$ with $E_G = \{e\}$ and $G_{\cdot\cdot} = v_1 \cdots v_k = \mathrm{att}_G(e)$ we have $\lhd_G = \lhd^e_G = \{(v_i, v_j) \mid 1 \le i < j \le k\}$ (because $V_i = \{v_i\}$ in the definition of $\lhd^e_G$). Thus, (S1) holds. Obviously, (S2) (invariance under isomorphism) also holds. □

Now, call a graph G *well formed* if it is rooted and it holds that $G_{\cdot\cdot}$ and $\mathrm{tar}_G(e)$, for every nonterminal edge $e \in E_G$, are ordered by $\lhd_G$. We let $\mathcal{R}$ be the set of all rules $A \to F$ in $\mathcal{C}$ such that F is well formed.

Example 5.3. Fig. 7 shows an example of a well-formed right-hand side F with two nonterminal edges labeled A and B, respectively. Here, it is assumed that the leftmost leaf, labeled by $(1')$ in the figure, is the first node of $F_{\cdot\cdot}$ and the other filled leaf, labeled (3), is the second. Following our general drawing conventions, the order of the outgoing tentacles of edges, from left to right, indicates the order of nodes in $\mathrm{tar}_F(e)$ for each edge e. We have

[6] Recalling the definition of "ordered by" from Section 2, this means that $\prec_{A^\bullet}$ imposes a unique total order on the target nodes of $A^\bullet$, and this order coincides with the actual order of those nodes in the target sequence $A^\bullet_{\cdot\cdot}$ of the graph (and thus also with the order of targets of its unique edge).
[7] Thus, while $\prec_G$ is defined on the concrete nodes appearing in G, the relation must be determined by the structure of G alone, rather then depending on the names of nodes and edges; see the example in the next section.
[8] Recall our general assumption (made for technical simplification) that hyperedge replacement does not rename nodes and edges. Hence, we can require equality here. Without this simplifying assumption we would have to require equality under the respective graph morphisms mapping H and F to G.

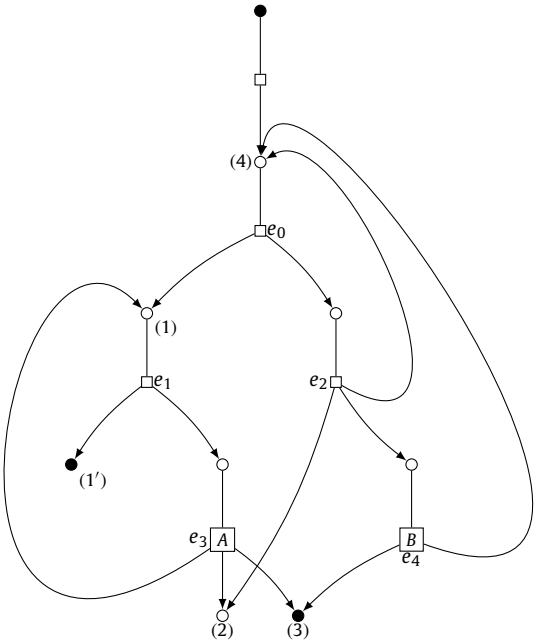

Fig. 7. A well-formed right-hand side.

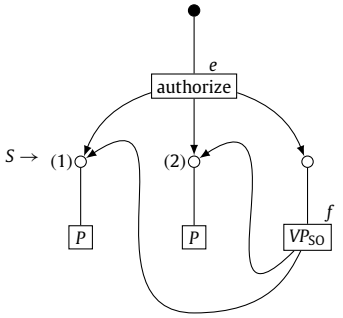

Fig. 8. A rule $S \to F$ that does not preserve $\lhd$.

- $\lhd_F^{e_0} = \emptyset$ because the reentrant nodes (1′) and (3) can both be reached via the same tentacle of e_0 (i.e., via (1)),
- (1′) $\lhd_F$ (2) and (1′) $\lhd_F$ (3) as (1′) can only be reached via the first tentacle of e_1 while (2) and (3) can only be reached via its second (and the fact that the latter two *can* be reached via the same tentacle of e_1 means that $\lhd_F^{e_1}$ does *not* order them),
- (1) $\lhd_F$ (2) $\lhd_F$ (3) because $\mathrm{reent}_F(e_3) = [\mathrm{tar}_F(e_3)] = \{(1), (2), (3)\}$, which are thus ordered according to their appearance in $\mathrm{tar}_F(e_3)$,
- (2) $\lhd_F$ (3) and (2) $\lhd_F$ (4) by virtue of $\lhd_F^{e_2}$ (which does not order (3) and (4)), and finally (3) $\lhd_F$ (4) by virtue of $\lhd_F^{e_4}$.

Example 5.4 *(Example 4.3 cont'd).* Further examples are provided by the rules in Figs. 2 and 5: as the reader may easily check, the edges in each right-hand side F of those rules all order both $F_{\bullet\bullet}$ and $\mathrm{tar}_F(e)$, for every nonterminal edge e, from left to right. Hence, these rules preserve $\lhd$.

It may be instructive to note that the rule $S \to F$ in Fig. 8, which could have extended the first row of rules in Fig. 2 in order to allow for the derivation of semantic graphs for sentences such as "Alice authorized Bob to defend her", does not preserve $\lhd$. This is because (1) $\lhd_F^e$ (2), but (1) is the second target node of f while (2) is its first. Hence, $\mathrm{tar}_F(f)$ is not ordered by $\lhd_F$.

The proof showing that $\mathcal{R}$ preserves $\lhd$ makes use of the following easy lemma.

Lemma 5.5. *Let $G = H[e : F]$ for a graph H and a rule $(\mathrm{lab}_H(e) \to F) \in \mathcal{C}$, and let $u, v \in V_H$. For every path p from u to v in H there is a path p' from u to v in G containing the same nodes of H as p does. Conversely, for every path p' from u to v in G, there is a path p from u to v in G containing the same nodes of H as p' does.*

Proof. Since F is either a duplication rule or satisfies (P1), it contains a (simple) source path p_w to every $w \in [F_{\bullet\bullet}]$, and as all nodes in $F_{\bullet\bullet}$ have out-degree 0, this p_w does not pass any of the nodes in $F_{\bullet\bullet}$. Thus, as a path in G, p_w does not pass any node of H. It follows for the first direction that every occurrence of e in p can be replaced by a suitable p_w to obtain p'. Conversely, since $u, v \in V_H$, in p' every maximal sub-path formed by edges in E_F is of the form p_w for some w, and can thus be replaced by e to obtain p. □

Theorem 5.6. *The family $\lhd$ is preserved by $\mathcal{R}$.*

Proof. Let $G = H[e:F]$ for $H \in \mathbb{G}_{\mathcal{R}}$, $e \in E_H$ with $\mathrm{lab}_H(e) = A$, and $(A \to F) \in \mathcal{R}$. Since $H \in \mathbb{G}_{\mathcal{R}}$, there is a derivation $A^\bullet \Rightarrow^n_{\mathcal{R}} H$ for some $n \in \mathbb{N}$. We show by induction on n that G is well formed and that, as required by Definition 4.8, $\lhd_G|_{V_H} = \lhd_H$ and $\lhd_G|_{V_F} = \lhd_F$.

As F is well formed, this holds for $n = 0$, because then $G = F$. For the induction step, we can thus assume that H is well formed. In particular, $\mathrm{tar}_H(f)$ is ordered by $\lhd_H$ (and we have $\mathrm{reent}_H(e) = [\mathrm{tar}_H(e)]$ as $H \in \mathbb{G}_{\mathcal{R}} \subseteq \mathbb{G}_{\mathcal{C}}$ satisfies (P3)).

First, let $I = V_F \setminus [\mathrm{ext}_F]$ be the set of internal nodes of F. Then, for all $v \in I$ and $f \in E_G$, $v \in \mathrm{reent}_G(f)$ only if $f \in E_F$ (by Lemma 4.4). Therefore, v appears in $\lhd^f_G$ only if $f \in E_F$. Furthermore, for $f \in E_F$, $\lhd^f_G = \lhd^f_F$ because $G{\downarrow_f} = F{\downarrow_f}$. Consequently, $\lhd_G \cap ((V_G \times I) \cup (I \times V_G)) = \lhd_F$. Moreover, since F is rooted, $\dot{F}$ occurs in none of the $\lhd^f_F$ ($f \in E_F$). As we furthermore know that $F_{\bullet\bullet}$ is ordered by $\lhd_F$ and $\mathrm{tar}_H(e)$ is ordered by $\lhd_H$, it remains to be shown that $\lhd^f_G = \lhd^f_H$ for all $f \in E_H \setminus \{e\}$. (Note that this also establishes well-formedness of G because H and F are well formed.)

Let $\mathrm{tar}_G(f) = v_1 \cdots v_k$. We know from Lemma 4.4 that $\mathrm{reent}_G(f) = \mathrm{reent}_H(f)$. Thus, by Definition 5.1(1), it suffices to show for all $i \in [k]$ and $v \in \mathrm{reent}_G(f)$ that v is reachable in $G{\downarrow_f}$ on a path p' not containing $\mathrm{src}_G(f)$ if and only if v is reachable in $H{\downarrow_f}$ on a path p not containing $\mathrm{src}_H(f)$. This, however, is true by Lemma 5.5. □

To end this section, we discuss how to compute $\lhd_G$. For this, we mainly have to explain how to compute $\lhd^e_G$ for every edge e. Below, let us say that the *inner rank* of a graph G is the maximum of the ranks of its edges and of all $|\mathrm{reent}_G(x)|$, $x \in V_G \cup E_G$. Note that if $G \in \mathcal{L}(\mathcal{G})$ for a reentrancy-preserving HR grammar $\mathcal{G} = (\Sigma, N, S, R)$, then its inner rank r is bounded by the maximal inner rank of the right-hand sides of R. Hence, in the estimations below we can always assume that r is bounded in this way because the condition can be checked during the pre-computation of all $\mathrm{reent}_G(x)$. If it is not fulfilled, the algorithm can immediately reject the input.

Lemma 5.7. *Let G be a graph. If the reentrancies of e have already been computed for every edge $e \in E_G$, $\lhd_G$ can be computed in $O(r|G|^2)$ additional steps, where r is the inner rank of G.*

Proof. It suffices to show that $\lhd^e_G$ can be computed in time $O(r|G|)$ for every edge $e \in E_G$, provided that $\mathrm{reent}_G(e)$ is known. As in Definition 5.1, let $\mathrm{tar}_G(e) = v_1 \cdots v_k$ and, for every $i \in [k]$, let V_i be the set of nodes in $\mathrm{reent}_G(e)$ which are reachable from v_i in $G{\downarrow_e}$ on a path not containing e. Since all nodes in $\mathrm{reent}_G(e)$ have out-degree zero in $G{\downarrow_e}$, we can collect the V_i by simple depth-first search in $O(|G|)$ steps, and thus $O(r|G|)$ steps in total because $k \leq r$. Afterwards, we compute for each $v \in \mathrm{reent}_G(e)$ the numbers $\mathit{lower}(v) = \min\{i \in [k] \mid v \in V_i\}$ and $\mathit{upper}(v) = \max\{i \in [k] \mid v \in V_i\}$. This takes $O(r)$ steps for every v and thus $O(r^2)$ steps in total. Finally, we obtain $\lhd^e_G$ from $\mathrm{reent}_G(e) \times \mathrm{reent}_G(e)$ in $O(r^2)$ steps by deleting all pairs (u, v) such that $\mathit{upper}(u) \geq \mathit{lower}(v)$.

In summary, since $r \leq |G|$, all parts of the computation of $\lhd^e_G$ run in time $O(r|G|)$, which proves the lemma. □

We shall prove in the next section (Lemma 6.6) that the reentrancies for all edges of a given graph can be computed in quadratic time. Hence, the assumption in Lemma 5.7 can in fact be dropped.

6. The parsing algorithm

We are now ready to develop our parsing algorithm and prove its correctness as well as analyze its running time.

In the following, given a graph G and some $x \in E_G \cup V_G$ such that $\prec_G$ orders $\mathrm{reent}_G(x)$, we let $G(x)$ denote the graph such that $G(x) \approx G{\downarrow_x}$ and $G(x)_{\bullet\bullet}$ is ordered by $\prec_G$. If $\prec_G$ does not order $\mathrm{reent}_G(x)$, then $G(x)$ is undefined. Note that, if G is given, then $G(x)$ can efficiently be represented by simply storing x and $G(x)_{\bullet\bullet}$, and in this representation it can easily be traversed in, e.g., a depth-first manner. In the following, we will thus assume that $G(e)$ is represented in this way when passed as a parameter to algorithms. Observe also that, for nodes $v \in V_G$, $G(v) = (\{v\}, \emptyset, \emptyset, \emptyset, v)$ if v is a leaf, and $G(v) = G(e)$ if v has out-degree 1 and $e \in E_G$ is the unique edge with $\mathrm{src}_G(e) = v$.

6.1. Intuition first

We start by illustrating how our algorithm would parse a small example graph, namely the graph G generated by the derivation in Fig. 6, using the rules in Figs. 2 and 5. To be able to illustrate an additional point, we furthermore add the rules in Fig. 9.

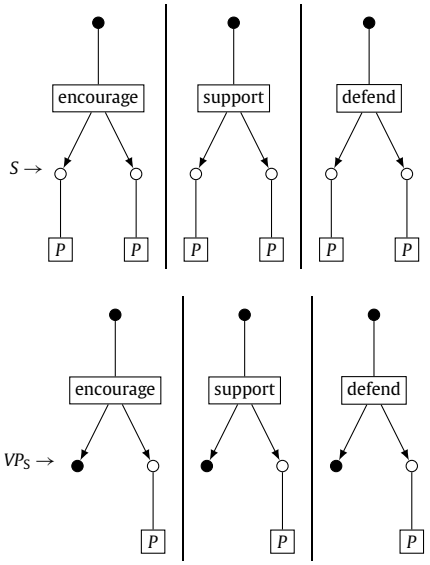

Fig. 9. Some additional reentrancy and order preserving rules.

The general idea is to compute, for every $x \in V_G \cup E_G$, the set $\mathrm{NT}(x)$ of all nonterminals A such that $A^\bullet \Rightarrow^* G(x)$. If $G(x)$ is undefined, then $\mathrm{NT}(x) = \emptyset$. Thus, similar to the well-known algorithm by Cocke, Kasami, and Younger, the input graph is found to be generated by the HR grammar if, at the end, the initial nonterminal belongs to $\mathrm{NT}(\dot{G})$.

The algorithm first determines $G(x)$ for every $x \in V_G \cup E_G$. If $G(x)$ is defined, it sets $\mathrm{NT}(x) = \bot$, otherwise $\mathrm{NT}(x) = \emptyset$. The former signifies that $\mathrm{NT}(x)$ still needs to be computed. In the example at hand, this means that $\mathrm{NT}(x) = \bot$ for all $x \in V_G \cup E_G$ (Fig. 10(a)).

Now, we repeatedly look for some $x \in V_G \cup E_G$ such that $\mathrm{NT}(y) \neq \bot$ for all non-external nodes and edges y in $G(x)$ other than x itself. Let us call such an x eligible. In the first step, there are five eligible x, namely the edges labeled "Alice" and "Bob" as well as those labeled "encourage", "support", and "defend". The algorithm processes eligible items x in any order. Suppose that it first chooses the former two. Both graphs $G(x)$, intuitively applying the corresponding rules backwards, can be generated only from an edge labeled P. We thus replace $\mathrm{NT}(x)$ by $\{P\}$ for each, leading to Fig. 10(b).[9] Now the source nodes of these two edges become eligible as well, and can thus similarly be marked with $\{P\}$ (Fig. 10(c)).

Now, let us turn to the graphs $G(y_i)$ where y_1, y_2, y_3 are the edges labeled "encourage", "support", and "defend". Note that each of them consists of y_i and its attached nodes alone, where the leftmost node is the first and the rightmost one is the second target (cf. Example 4.3). Again applying the three rules backwards, we arrive at Fig. 10(d). Note that, despite the rules in Fig. 9, $\mathrm{NT}(y_i)$ contains neither S nor VP_S, because their right-hand sides do not match $G(y_i)$. If, for example, the leftmost node would not be reentrant (i.e., only one of the three edges would have it as its target), the corresponding right-hand side of VP_S would match instead. (The matching process will be discussed below.)

Now it is time to determine $\mathrm{NT}(x)$, where x is the common source node of the edges y_1, y_2, y_3. The graph $G'(x)$ obtained from $G(x)$ by replacing each subgraph $G(y_i)$ by an edge labeled with the set of labels $\mathrm{NT}(y_i)$ (in this case only singletons) is a so-called shallow graph: it consists of three parallel edges attached to the same sequence of targets. As we shall see in Section 6.2, we can efficiently determine the set of nonterminal labels A such that $A^\bullet$ derives some instance of $G'(x)$ obtained by choosing a label from $\mathrm{NT}(y_i)$ for each y_i. In the present case, the result is obvious: only $A = VP_{\mathrm{SO}}^+$ accomplishes this, using the clone rule twice and then the last rule in Fig. 5. This yields Fig. 10(e).

Next, the edge y labeled "and" becomes eligible. We try to match the right-hand side F of each rule with a deep right-hand side (a right-hand side which is not shallow) against $G(y)$. Recall that every node in such a right-hand side has out-degree at most 1. This allows us to match it against $G(y)$ in a top-down fashion, as follows: starting from the source nodes x_0 and x of F and $G(y)$, respectively, an outgoing terminal edge of x_0 matches the outgoing edge of x if the latter (exists and) has the same label. If so, the matching process continues at the targets of the matching edges. A nonterminal edge, in contrast, is matched by checking whether its nonterminal label is contained in $\mathrm{NT}(x)$. If so, the matching process continues at the nodes in $[\![\mathrm{reent}_G(x)]\!]_{\prec_G}$. If matching succeeds all the way to the external nodes of F and $G(y)$ (which must be mapped onto each other), the left-hand side of the rule is included in $\mathrm{NT}(y)$. In the present case, this means that only the left-hand side VP_{SO} of the first rule in Fig. 5 is included in $\mathrm{NT}(y)$. Since $G(y) = G(x)$, the next step will propagate this to the source node x of y, yielding the situation in Fig. 10(f).

[9] Technically, our algorithm would actually do this in two steps.

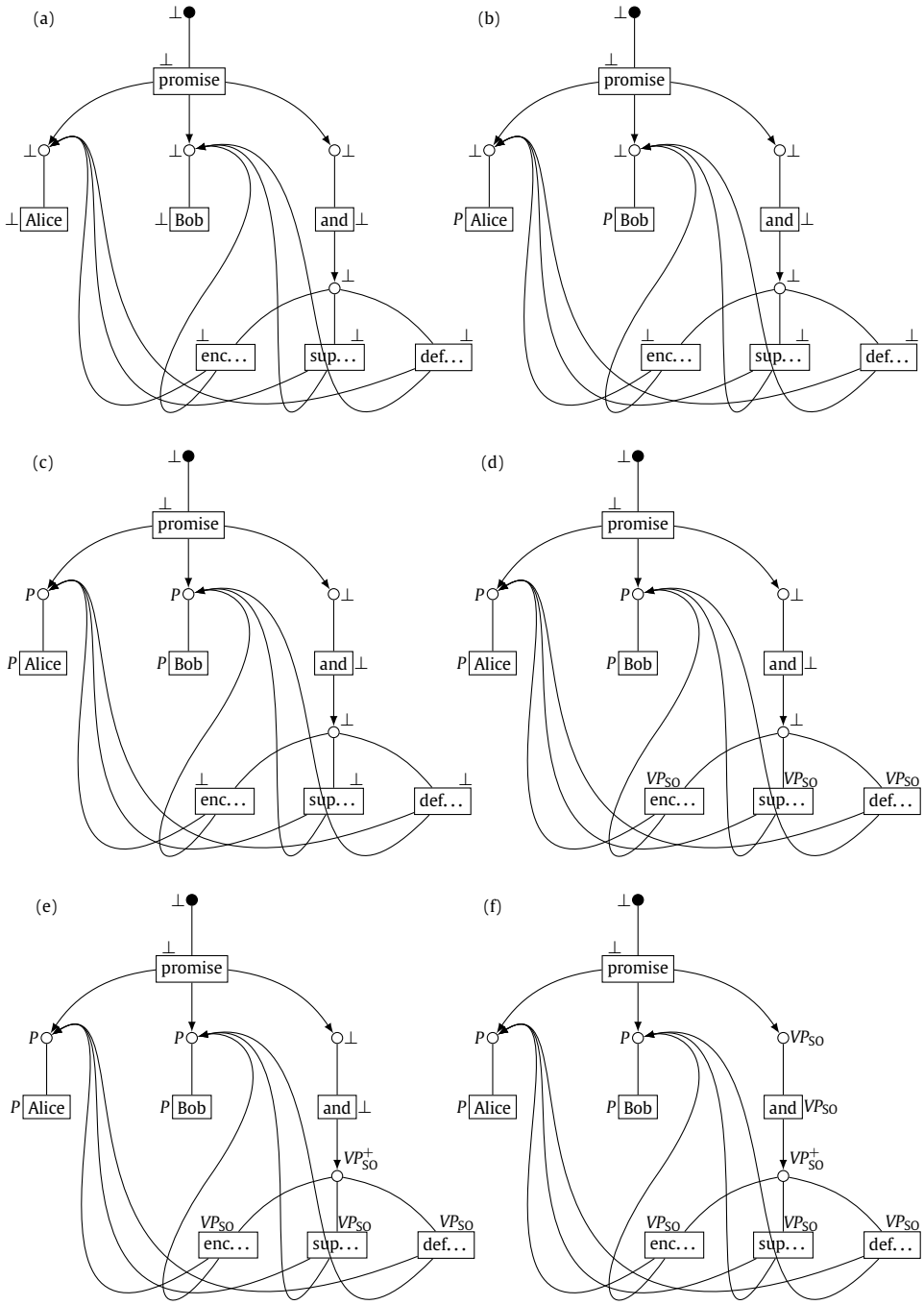

Fig. 10. Different stages of the parsing algorithm illustrated.

Finally, the edge y labeled "promise" has become eligible. Since, for this y, $\mathrm{reent}_G(y) = \emptyset$, the only right-hand side that matches is the last one in the top row of Fig. 2, which means that $\mathrm{NT}(y) = \{S\}$ and thus also $\mathrm{NT}(\dot{G}) = \{S\}$. In other words, G can be derived from $S^\bullet$.

6.2. Shallow graphs

We start our formal description of the parsing algorithm by discussing how to parse when the grammar contains only duplication rules. This will provide some insights into how our general parsing algorithm handles cases where a node has

more than one outgoing edge. Since duplication rules only use nonterminal labels in their right-hand sides, we only consider nonterminal labels in this section.

Definition 6.1. A graph G is *shallow* if $\dot{G}$ is its root and no path in G has length more than one. Additionally, we require G to only have nonterminal labels. A graph which is not shallow is *deep*. A rule is said to be shallow or deep depending on whether its right-hand side is.

Note that graphs containing terminal labels are always considered to be deep, even if they do not contain a path of length greater than one.

It should be clear that shallow rules only produce shallow graphs. In particular, duplication rules do. For the most part of Section 6.2, we shall restrict our attention to duplication rules. Note that, using exclusively duplication rules, every derivation of a graph G from a graph $A^\bullet$ consists of $|E_G| - 1$ steps.

Definition 6.2. A *siblinghood* in a shallow graph G is a set Sib of edges such that $\mathrm{tar}_G(e) = \mathrm{tar}_G(e')$ for all $e, e' \in Sib$. Given a siblinghood Sib we write $\mathrm{tar}_G(Sib)$ for this sequence, where $\mathrm{tar}_G(\emptyset) = G_{\bullet\bullet}$. The *size* of a siblinghood is the number of edges in it.

The next lemma gathers some useful properties of siblinghoods in graphs produced by duplication rules.

Lemma 6.3. *Let R be a set of duplication rules and let $A^\bullet \Rightarrow_R^* G$ be a derivation. Then the following holds for every siblinghood Sib in G:*

1. *If $|E_G| > 1$ then G contains a siblinghood of size 2.*
2. *$G_{\bullet\bullet}$ is a subsequence of $\mathrm{tar}_G(Sib)$.*
3. *$\mathrm{lab}_G(e)$ is the same for all edges $e \in Sib$.*
4. *Let T_1 and T_2 be siblinghoods in G. Then the overlap of the targets of one of them with Sib is a subset of the overlap of the other with Sib. Formally, $[\mathrm{TAR}_G(Sib)] \cap [\mathrm{TAR}_G(T_1)]$ is a subset of $[\mathrm{TAR}_G(Sib)] \cap [\mathrm{TAR}_G(T_2)]$ or vice versa.*
5. *If Sib is a siblinghood of size 2, then there is a derivation of G of the form $A^\bullet \Rightarrow_R^* H \Rightarrow H[e : F]$, where $E_F = Sib$ (i.e., the very last step of the derivation introduces the siblinghood Sib).*

Proof. Property (1) is obvious: by the definition of siblinghoods, every duplication rule introduces a siblinghood of size 2. For the remaining properties, we proceed by induction on the length of derivations (or, equivalently, the size of E_G). To show that property (2) holds, it is clearly sufficient to show property (2′): $G_{\bullet\bullet}$ is a subsequence of $\mathrm{tar}_G(e)$ for every edge $e \in E_G$.

Let $A^\bullet = G_0 \Rightarrow G_1 \Rightarrow \cdots \Rightarrow G_n = G$ be a derivation by rules in R. The base case is $G = A^\bullet$. Then G contains only one edge e, and $\mathrm{tar}_G(e) = G_{\bullet\bullet}$.

For the inductive case, we assume that the properties hold for $G_0, \ldots, G_k$. Let $G_{k+1} = G_k[e : F]$ for some edge e in G_k and a rule $A \to F$ in R with $A = \mathrm{lab}_{G_k}(e)$, where $E_F = \{f_1, f_2\}$. By the definition of duplication rules, $\mathrm{tar}_{G_k}(e)$ is a subsequence of $\mathrm{tar}_{G_{k+1}}(f_i)$, and thus property (2′) holds by the induction hypothesis.

To show properties (3)–(5), assume first that F is a clone rule and consider a siblinghood Sib. The inductive assumption that G_k satisfies property (4) immediately yields that G_{k+1} does so as well, because $\{\mathrm{tar}_{G_{k+1}}(T) \mid T \text{ is a siblinghood of } G_{k+1}\}$ is equal to $\{\mathrm{tar}_{G_k}(T) \mid T \text{ is a siblinghood of } G_k\}$, using the fact that $\mathrm{tar}_{G_{k+1}}(f_i) = \mathrm{tar}_{G_k}(e)$. We furthermore have that $\mathrm{lab}_F(f_1) = \mathrm{lab}_f(f_2) = \mathrm{lab}_{G_k}(e)$. If $Sib \cap \{f_1, f_2\} = \emptyset$, then property (3) holds by the induction hypothesis, because Sib is a siblinghood in G_k. To see that this indeed also establishes property (5) note that, if $G_k = G_{k-1}[e' : F']$ with $E_{F'} = Sib$, then $G_{k+1} = G_{k-1}[e : F][e' : F']$, i.e., the last two steps of the derivation can be interchanged. If $Sib \cap \{f_1, f_2\} = \{f_1\}$ (the other case being symmetric), then $Sib' = (Sib \setminus \{f_1\}) \cup \{e\}$ is a siblinghood in G_k, and thus again the induction hypothesis proves (3) since $\mathrm{lab}_F(f_i) = \mathrm{lab}_{G_k}(e)$. To see that property (5) holds in this case as well, suppose additionally that Sib is of size 2, say $Sib = \{f_1, f_2'\}$. Then all of f_1, f_2, and f_2' have identical attachments and labels, and we may assume by induction that $\{f_1, f_2'\}$ is introduced in G_k. However, then property (5) holds by simply applying an isomorphism that interchanges the roles of f_2 and f_2'. Finally, if $\{f_1, f_2\} \subseteq Sib$, property (3) follows by the same reasoning as before, and if Sib is additionally of size 2 and thus equal to $\{f_1, f_2\}$, then property (5) holds immediately.

To finish, assume now that F is a twin, i.e., $E_F = \{f_1, f_2\}$ and $\mathrm{lab}_F(f_1) = B = \mathrm{lab}_F(f_2)$ for a nonterminal B such that $\mathrm{rank}(B) > \mathrm{rank}(A)$. Then the only siblinghoods Sib containing f_1 or f_2 are subsets of the siblinghood $\{f_1, f_2\}$. Property (4) also holds, because $[\mathrm{TAR}_{G_{k+1}}(\{f_1, f_2\})] \setminus [\mathrm{TAR}_{G_k}(\{e\})]$ are new vertices, not connected to any edge in G_k and thus $[\mathrm{TAR}_{G_{k+1}}(T)] \cap [\mathrm{TAR}_{G_{k+1}}(Sib)]$ is equal to $[\mathrm{TAR}_{G_k}(T)] \cap [\mathrm{TAR}_{G_k}(\{e\})]$ for all siblinghoods T of G_k.[10] Property (3) clearly holds for Sib, and by the induction hypothesis also for all other siblinghoods, as those are siblinghoods in G_k. Finally, if any siblinghood Sib of G_{k+1} is of size 2, it is either equal to $\{f_1, f_2\}$, in which case property (5) holds immediately, or it is a

[10] For the sake of completeness, note additionally that $[\mathrm{TAR}_{G_{k+1}}(\emptyset)] \cap [\mathrm{TAR}_{G_{k+1}}(Sib)] = [G_{k+1\bullet\bullet}]$ by property (2). By a second application of property (2), this establishes property (4) for the case where $T_1 = \emptyset$ or $T_2 = \emptyset$.

Algorithm 1 Parsing with Duplication Rules.

```
1: function ShallowParse(set R of duplication rules, shallow abstract graph G)
2:     while |E_G| > 1 do
3:         if G contains no siblinghood of size 2 then
4:             return ∅
5:         choose a siblinghood Sib of size 2
6:         replace Sib in G with a new edge e with target sequence itar_G(Sib)
7:         lab(e) ← {A | ∃B ∈ lab_G(Sib): A → B(G↓_Sib)}
8:     return lab_G(e) where {e} = E_G
```

siblinghood in G_k. In the latter case, we can again apply the induction hypothesis to the derivation of G_k and then swap the last two rule applications, thus introducing *Sib* in the last step, as above. □

In particular, property (4) of Lemma 6.3 implies that for every siblinghood *Sib*, either $\mathrm{tar}_G(Sib) = G_{\bullet\bullet}$ or there is a unique maximal subsequence $\mathrm{itar}_G(Sib)$ (for *inherited targets*) of $\mathrm{tar}_G(Sib)$ such that $\mathrm{itar}_G(Sib)$ is also a subsequence of $\mathrm{tar}_G(T)$ for some other siblinghood $T \neq Sib$. If $\mathrm{tar}_G(Sib) = G_{\bullet\bullet}$, we define $\mathrm{itar}_G(Sib)$ to be $\mathrm{tar}_G(Sib)$. If *Sib* is a siblinghood of G, we write $G{\downarrow_{Sib}}$ for the subgraph of G induced by *Sib*, where $G{\downarrow_{Sib}}_{\bullet\bullet} = \mathrm{itar}_G(Sib)$. We write $B(G{\downarrow_{Sib}})$ for the graph obtained from $G{\downarrow_{Sib}}$ by setting $\mathrm{lab}(e) = B$ for every edge e in $G{\downarrow_{Sib}}$.

In the upcoming algorithms, we use intermediate graphs that are *abstract* in the sense that each edge carries a set of labels rather than a single label:

Definition 6.4. An *abstract graph* is a tuple $G = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext})$, where V, E, att, and ext are as for graphs, but lab is a function from E to finite sets of labels. For a siblinghood *Sib* in a shallow abstract graph G, we define $\mathrm{lab}_G(Sib) = \bigcap_{e \in Sib} \mathrm{lab}_G(e)$. A *concretization* of G is a graph $H = (V_G, E_G, \mathrm{att}_G, \mathrm{lab}, \mathrm{ext}_G)$ such that $\mathrm{lab}(e) \in \mathrm{lab}_G(e)$ for all $e \in E_G$.

Given a graph G, we implicitly identify it with the abstract graph where $\mathrm{lab}(e)$ is a singleton set for each node, and vice versa.

A parsing algorithm handling the derivation of graphs by duplication rules is given as Algorithm 1. We argue its correctness below. More specifically, we argue that, given an input graph G, it returns the set of all nonterminals A such that $A^\bullet \Rightarrow^* G$.

First, we note that if the condition on line 3 ever becomes true, then this is because property 1 of Lemma 6.3 does not hold, and thus G cannot be derived.

If this does not happen, we can view Algorithm 1 as producing a sequence $G_0, \ldots, G_n$ of abstract graphs such that $G_0 = G$ and G_n has only one edge. Let G_i and G_{i+1} be two graphs in the constructed sequence. Then there is a siblinghood *Sib* of size 2 in G_i and an edge e in G_{i+1} such that G_i is isomorphic to $G_{i+1}[e : G_i{\downarrow_{Sib}}]$. We argue that $\mathrm{lab}_{G_{i+1}}(e)$ is the set of all nonterminals A such that there exists a nonterminal B in $\mathrm{lab}_{G_i}(Sib)$ with $A \to B(G_i{\downarrow_{Sib}})$. To see this, suppose first that a nonterminal A was included in $\mathrm{lab}_{G_{i+1}}(e)$. Line 6 of ShallowParse in the algorithm replaces the siblinghood *Sib* in question with a single edge e with $\mathrm{tar}_{G_{i+1}}(e) = \mathrm{itar}_{G_i}(Sib)$. On line 7, the label set of e is set to those nonterminals that have an appropriate duplication rule for some B in $\mathrm{lab}_{G_i}(Sib)$. (Note that B is equal to A if and only if the rule is a clone rule if and only if $\mathrm{tar}_{G_i}(Sib) = \mathrm{itar}_{G_i}(Sib)$.) Thus, starting with a single edge e with label A, we can use this rule to produce a twin or clone G'_i. Clearly, this graph is isomorphic to $B(G_i{\downarrow_{Sib}})$. Conversely, a similar reasoning gives us that for every $B \in \mathrm{lab}_{G_i}(Sib)$, Algorithm 1 includes a nonterminal A in $\mathrm{lab}_{G_{i+1}}(e)$ such that $A \to B(G_i{\downarrow_{Sib}})$.

We now consider the case where G_0, the input to the algorithm, is a concrete graph in the sense that every edge has a label set of size exactly one. We argue that if Algorithm 1 eventually arrives at a graph G_n with $E_{G_n} = \{e\}$, then $\mathrm{lab}_{G_n}(e)$ is the set of all nonterminals A such that G'_0, the unique concretization of G_0 can be derived from $A^\bullet$. First, let $G_0, \ldots, G_n$ and A be as above. We argue that there are graphs $G'_n, G'_{n-1}, \ldots, G'_0$ such that

$$A^\bullet = G'_n \Rightarrow G'_{n-1} \Rightarrow \cdots \Rightarrow G'_0$$

and for each $i \in \{0, \ldots, n\}$, G'_i is a concretization of G_i. In particular, this means that G'_0 can thus be derived from $A^\bullet$.

For the induction basis, let $Sib = \{e\}$ be the unique siblinghood in G_n. Since A is in $\mathrm{lab}_{G_n}(e)$, $A^\bullet = G'_n$ is a concretization of G_n.

For the inductive case, consider G_i and G_{i+1}. As above, let e be the edge in G_{i+1} and $Sib = \{f_1, f_2\}$ the siblinghood in G_i such that $G_i = G_{i+1}[e : G_i{\downarrow_{Sib}}]$. By the induction hypothesis, there is an edge e' in G'_{i+1} such that $\mathrm{lab}(e') \in \mathrm{lab}(e)$. Let $\mathrm{lab}(e') = A$. By the reasoning above, there is a B in $\mathrm{lab}_{G_i}(Sib)$ such that $A \to B(G_i{\downarrow_{Sib}})$. This means that from G'_{i+1} we can derive a graph G'_i that is a concretization of G_i. In particular, siblinghood *Sib* will have label B in G'_i, which belongs to $\mathrm{lab}_{G_i}(Sib)$.

For the other direction, assume that $A^\bullet = G'_n \Rightarrow G'_{n-1} \Rightarrow \cdots \Rightarrow G'_0 = G$. By Lemma 6.3(5) we can assume that every step in the derivation introduces a siblinghood of size 2. Choosing this siblinghood on line 5 of the algorithm and proceeding by an induction similar to the one above, yields the required graphs $G_1, \ldots, G_n$.

Algorithm 1 handles only shallow graphs generated by duplication rules. However, the derivation of a shallow graph by a reentrancy-preserving grammar may also make use of shallow rules $A \to F$ where $E_F = \{f\}$ for a nonterminal edge f. Since

we only consider order-preserving rules, these rules are always of the form $A \to B^\bullet$ with $\mathrm{rank}(A) = \mathrm{rank}(B)$, called chain rules in the following. Their effect is essentially the same as the effect of chain rules in context-free string grammars: they only relabel a nonterminal. Consequently, standard techniques can be used to extend Algorithm 1 appropriately, as follows. For the given HR grammar and an abstract graph G, define the (backwards) closure of G to be

$$cl(G) = (V_G, E_G, \mathrm{att}_G, \mathrm{lab}, \mathrm{ext}_G)$$

where

$$\mathrm{lab}(e) = \{A \in N \mid A^\bullet \Rightarrow^* B^\bullet \text{ for some } B \in \mathrm{lab}_G(e)\}$$

for all $e \in E_G$. Then the only two changes that must be made to Algorithm 1 in order to handle chain rules are:

1. On line 7 of ShallowParse, replace "$B \in \mathrm{lab}_G(Sib)$" by "$B \in \mathrm{lab}_{cl(G)}(Sib)$".
2. On line 8 of ShallowParse, replace "$\mathrm{lab}_G(e)$" by "$\mathrm{lab}_{cl(G)}(e)$".

In effect, this incorporates the application of all relabellings permitted by the grammar into the derivations constructed by Algorithm 1, in the standard way known from context-free string grammars.

Summarizing, we get the following lemma:

Lemma 6.5. ShallowParse(R, G), *extended to rules of the form* $A^\bullet \to B^\bullet$ *as discussed above, runs in time* $\mathcal{O}(|R|^2 + |G|^2)$ *for every set* R *of shallow rules and every shallow abstract graph* G*, and it holds that*

$$\text{ShallowParse}(R, G) = \{A \in \mathrm{LAB_N} \mid A^\bullet \Rightarrow_R^* H \text{ for some concretization } H \text{ of } G\}.$$

Proof. The required correctness arguments were given above. To implement the generalization efficiently, note that the set of all pairs (A, B) with $A^\bullet \Rightarrow^* B^\bullet$ can be precomputed in time $|R|^2$ by a standard technique. The algorithm itself runs in time $|G|^2$ because choosing *Sib* on line 5 requires only linear time and the loop terminates after at most $|E_G|$ executions. □

The reader should note for later use that the term $|R|^2$ in the running time estimation is a one-time investment if ShallowParse(R, G) is run several times with differing G but the same set R of rules.

6.3. The general algorithm

We now present our parsing algorithm, show that it is correct, and determine its worst-case running time. Throughout this section, let $\mathcal{G} = (\Sigma, N, S, R)$ denote the order-preserving HR grammar which, together with a graph G over Σ, is the input to the parsing algorithm. Let $P \subseteq R$ be the set of deep rules in R, and $Q \subseteq R$ be the set of duplication rules (i.e., $P \cap Q = \emptyset$ and $R \setminus (P \cup Q)$ is the set of chain rules in R).

We want to parse the input graph G with respect to $\mathcal{G}$. Throughout most of the section, we will assume that $\prec = (\prec_G)_{G \in \mathbb{G}_\mathcal{C}}$ is a suitable family of orders and that $G \in \mathbb{G}_\mathcal{R}$ for a set $\mathcal{R}$ that preserves $\prec$. We generalize the reasoning to arbitrary input graphs at the end of the section. Thus, we furthermore assume that $\prec_G$ has been computed beforehand, and that G fulfills condition (P1) in Definition 4.2 (i.e., all nodes and edges are reachable on source paths), while (P3) is trivially fulfilled since G is terminal.

As the following lemma states, if $G \in \mathbb{G}_\mathcal{R}$ and $x \in V_G \cup E_G$, then $\mathrm{reent}_G(x)$ can be efficiently computed.

Lemma 6.6. *Let* $G \in \mathbb{G}_\mathcal{R}$*. 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 graph* $G(x)$*, provided that it is defined (since we assume that* $\prec_G$ *has been precomputed).*

Proof. For each $x \in V_G \cup E_G$, we can make a depth-first search starting at $\dot{G}$ to determine the set of nodes reachable without passing x. Adding $[\mathrm{ext}_G]$, this yields $V = \mathrm{TAR}_G(E_G \setminus E_G^x) \cup [\mathrm{ext}_G]$. Afterwards, we perform a similar search along paths starting at x and not passing any node in V. This takes linear time for each individual x and reveals the nodes in $\mathrm{reent}_G(x)$ as they are exactly the nodes in V which can be reached from x on such paths. Performing this procedure for all $x \in V_G \cup E_G$ thus takes quadratic time. □

Before we present the parsing algorithm, let us formulate and prove a lemma that will enable us to match deep right-hand sides against (subgraphs of) the input graph. This lemma captures one of the central reasons why order-preserving HR grammars can be parsed efficiently: the mapping of nodes in a candidate right-hand side to corresponding nodes in the host graph is uniquely determined (if it exists). For the lemma and the rest of the section, recall from Section 2 that $\mathrm{out}_G(v)$ denotes the set of all outgoing edges of a node v in a graph G.

Lemma 6.7. *Let $H = F[f_1 : G_1, \ldots, f_k : G_k]$ for graphs $F, G_1, \ldots, G_k \in \mathbb{G}_{\mathcal{R}}$, where F is a deep graph satisfying (P1)–(P3) and $f_1, \ldots, f_k$ are the nonterminal edges in F. If H is isomorphic to $G(e)$ via an isomorphism $h\colon H \to G(e)$ for a graph $G \in \mathbb{G}_{\mathcal{R}}$ and an edge $e \in E_G$, then the restriction ϕ of h_V to V_F is the unique mapping defined recursively as follows, for every node $v \in V_F$:*

(i) if $v = \dot{F}$, then $\phi(v) = \mathrm{src}_G(e)$;
(ii) if $v = \mathrm{src}_F(f)$ for a terminal edge $f \in E_F$, then $\mathrm{out}_{G(e)}(\phi(v))$ is a singleton $\{f'\}$ with $\mathrm{lab}_G(f') = \mathrm{lab}_F(f)$ and it holds that $\phi(\mathrm{tar}_F(f)) = \mathrm{tar}_G(f')$; and
(iii) if $v = \mathrm{src}_F(f_i)$ for some $i \in [k]$, then $\phi(\mathrm{tar}_F(f_i)) = [\![\mathrm{reent}_G(\phi(v))]\!]_{\prec_G}$.

Proof. Consider any isomorphism h as in the statement of the lemma, let ϕ be its restriction to V_F, and $v \in V_F$.

If $v = \dot{F}$, then (i) holds by the definition of hyperedge replacement and isomorphism, because $\phi(\dot{F}) = \phi(\dot{H}) = \mathrm{src}_G(e)$.

If $v = \mathrm{src}_F(f)$ where f is terminal, since v has out-degree (at most) one, we have $\mathrm{out}_H(v) = \mathrm{out}_F(v) = \{f\}$. Hence by the definition of isomorphisms $\mathrm{out}_{G(e)}(\phi(v))$ is the singleton $\{f'\}$ with $f' = h_E(f)$, and we also have $\mathrm{lab}_G(f') = \mathrm{lab}_F(f)$. Again by the definition of hyperedge replacement and isomorphisms $\phi(\mathrm{tar}_F(f)) = \phi(\mathrm{tar}_H(f)) = \mathrm{tar}_G(f')$.

Finally, assume that $f = f_i$ for some $i \in [k]$. As F is deep and f nonterminal, it follows that $\mathrm{src}_F(f) \neq \dot{F}$. This is so because $\dot{F}$ has out-degree 1, and thus $\mathrm{src}_F(f) = \dot{F}$ implies $[\mathrm{tar}_F(f)] = \mathrm{reent}_F(f) = \mathrm{reent}_F(\dot{F}) = [F_{\bullet\bullet}]$ (where the first equality is due to (P3)), meaning that F is shallow as all nodes in $[F_{\bullet\bullet}]$ have out-degree 0. But, by the precondition of the lemma, F is not shallow.

Let $H = I[f : G_i]$ were I is the graph in $\mathbb{G}_{\mathcal{R}}$ given by

$$I = F[f_1 : G_1, \ldots, f_{i-1} : G_{i-1}, f_{i+1} : G_{i+1}, \ldots, f_k : G_k].$$

We get

$$\begin{aligned}
\mathrm{reent}_H(v) &= \mathrm{reent}_I(\mathrm{src}_F(f)) && \text{(by Lemma 4.4 and since } v = \mathrm{src}_F(f)) \\
&= \mathrm{reent}_F(\mathrm{src}_F(f)) && \text{(by } (k-1)\text{-fold application of Lemma 4.4)} \\
&= \mathrm{reent}_F(f) && \text{(because, by (P1), } \mathrm{src}_F(f) \text{ has out-degree 1)} \\
&= [\mathrm{tar}_F(f)] && \text{(by (P3))}
\end{aligned}$$

Since $\mathcal{R}$ preserves $\prec$, each of these steps furthermore preserves the order of nodes, and by Observation 4.9 $\mathrm{tar}_F(f)$ is ordered by $\prec_F$, which yields $\mathrm{tar}_F(f) = [\![\mathrm{reent}_H(v)]\!]_{\prec_H}$. As the definition of reentrancies is obviously invariant under isomorphism, and $\prec$ is so by (S2), the latter means that $\phi(\mathrm{tar}_F(f)) = [\![\mathrm{reent}_{G(e)}(\phi(v))]\!]_{\prec_{G(e)}}$.

It remains to be verified that $[\![\mathrm{reent}_{G(e)}(\phi(v))]\!]_{\prec_{G(e)}} = [\![\mathrm{reent}_G(\phi(v))]\!]_{\prec_G}$. As we saw above, $v = \dot{F}$ would contradict the fact F is deep (because $v = \mathrm{src}_F(f)$ and f is nonterminal). Thus, $v \neq \dot{F}$ and we have $\phi(v) \neq \mathrm{src}_G(e)$, and hence $\phi(v) \in V_{G(e)} \setminus [\mathrm{ext}_{G(e)}]$. This shows that Lemma 3.5 applies with $x = e$ and $y = \phi(v)$, telling us that $G{\downarrow_e}{\downarrow_{\phi(v)}} = G{\downarrow_{\phi(v)}}$. In particular, $\mathrm{reent}_{G(e)}(\phi(v)) = \mathrm{reent}_G(\phi(v))$ and thus $[\![\mathrm{reent}_{G(e)}(\phi(v))]\!]_{\prec_{G(e)}} = [\![\mathrm{reent}_G(\phi(v))]\!]_{\prec_G}$, as required. □

Note that ϕ in Lemma 6.7 does not depend on $G_1, \ldots, G_k$. Thus, given F, G, and e, and assuming that $\mathrm{reent}_G(v) = \mathrm{reent}_{G(e)}(v)$ has been precomputed for all $v \in V_G$, the mapping ϕ (viewed as a subset of $V_F \times V_G$) can be computed in linear time by an obvious recursion along the cases (i)–(iii). This yields the following corollary.

Corollary 6.8. *Let $H = F[f_1 : G_1, \ldots, f_k : G_k]$ for graphs $F, G_1, \ldots, G_k \in \mathbb{G}_{\mathcal{R}}$, where F is a deep graph satisfying (P1)–(P3) and $f_1, \ldots, f_k$ are the nonterminal edges in F. If H is isomorphic to $G(e)$ via an isomorphism $h\colon H \to G(e)$ for a graph $G \in \mathbb{G}_{\mathcal{R}}$ and an edge $e \in E_G$, then the restriction ϕ of h_V to V_F can be computed by Algorithm 2 in time $O(|F|)$, provided that $\prec_G$ and $\mathrm{reent}_G(u)$ are known for every node $u \in V_G$.*

Note that the computation of ϕ fails if it reveals an inconsistency, i.e., if $\mathrm{out}_J(\phi(v))$ is not a singleton $\{f'\}$ with $\mathrm{lab}_J(f') = \mathrm{lab}_F(f)$ in case (ii), there happen to be conflicting assignments of values to $\phi(v)$ for some v (i.e., ϕ is not a function), ϕ is not injective, or $\phi(F_{\bullet\bullet}) \neq [\![\mathrm{reent}_G(e)]\!]_{\prec_G}$.

The two main routines of the parser are $\textsc{parse}_{\mathrm{V}}(v)$ and $\textsc{parse}_{\mathrm{E}}(e)$, which are called alternately by PARSE to augment the nodes and edges of G with sets of nonterminals:

1. $\textsc{parse}_{\mathrm{V}}(v)$ augments a node $v \in V_G$ with $\mathrm{NT}(v) = \{A \in N \mid A^\bullet \Rightarrow^* G(v)\}$.
2. $\textsc{parse}_{\mathrm{E}}(e)$ augments an edge $e \in E_G$ with $\mathrm{NT}(e) = \{A \in N \mid A^\bullet \Rightarrow^* G(e)\}$.

In both cases, $\mathrm{NT}(x) = \emptyset$ if $G(x)$ is undefined.

The pseudocode, $\textsc{parse}_{\mathrm{V}}$ and $\textsc{parse}_{\mathrm{E}}$ is given as Algorithms 3, 4, and 5. $\textsc{parse}_{\mathrm{E}}$ additionally calls the subroutine MATCH, given as Algorithm 6. The following is a high-level description of how the algorithms work:

Algorithm 2 Computing the mapping ϕ (using a nested recursive procedure).

function ϕ-COMPUTATION(deep right-hand side F, graph G, edge $e \in E_G$)
  let $\phi(v)$ be undefined for all $v \in V_F$
  ϕ-COMPUTATION_REC($\mathrm{src}_F, \mathrm{src}_{G(e)}, \emptyset$)
  if $\phi(F_{\bullet\bullet}) = [\![\mathrm{reent}_G(e)]\!]_{\prec_G}$ **then return** ϕ
  else failure ▷ $F_{\bullet\bullet}$ mismatch
  where
  procedure ϕ-COMPUTATION_REC(nodes $v \in V_F$, $v' \in V_{G(e)}$, set $V \subseteq V_{G(e)}$)
    if $\phi(v)$ is defined **then**
      if $\phi(v) \neq v'$ **then** failure ▷ ϕ not a function
      else return ▷ have been here earlier
    else if $v' \notin V$ **then** $\phi(v) \leftarrow v'$
    else failure ▷ ϕ not injective
    if $|\mathrm{out}_F(v)| = 1$ **then**
      let $\mathrm{out}_F(v) = \{f\}$ and $\mathrm{tar}_F(f) = v_1 \cdots v_k$
      if f is terminal and $|\mathrm{out}_{G(e)}(v')| = 1$ **then**
        let $\mathrm{out}_{G(e)}(v') = \{f'\}$ where $\mathrm{tar}_{G(e)}(f') = v'_1 \cdots v'_\ell$
        if $\mathrm{lab}_F(f) = \mathrm{lab}_G(f')$ (and thus $k = \ell$) **then**
          for $i = 1, \ldots, k$ **do** ϕ-COMPUTATION_REC($v_i, v'_i, V \cup \{v'\}$)
        else failure ▷ edge mismatch
      else
        let $[\![\mathrm{reent}_G(v')]\!]_{\prec_G} = v'_1 \cdots v'_\ell$
        if $k = \ell$ **then**
          for $i = 1, \ldots, k$ **do** ϕ-COMPUTATION_REC($v_i, v'_i, V \cup \{v'\}$)
        else failure ▷ incorrect number of targets
    else if $\mathrm{out}_{G(e)}(v') \neq \emptyset$ **then** failure ▷ edges in $G(e)$ unaccounted for

Algorithm 3 Parsing for Order-Preserving HR Grammars.

function PARSE(order-preserving HR grammar $\mathcal{G} = (\Sigma, N, S, R)$, graph $G \in \mathbb{G}_{\mathcal{R}}$)
  preProcess(G) ▷ Compute $\prec_G$ as well as all $G(x)$ for all $x \in V_G \cup E_G$
  for $x \in V_G \cup E_G$ **do**
    if $G(x)$ is defined **then** $\mathrm{NT}(x) \leftarrow \bot$
    else $\mathrm{NT}(x) \leftarrow \emptyset$
  while $\mathrm{NT}(\dot{G}) = \bot$ **do**
    let $x \in V_G \cup E_G$ with $\mathrm{NT}(x) = \bot$ and
      $\mathrm{NT}(y) \neq \bot$ for all $y \in (V_{G(x)} \cup E_{G(x)}) \setminus ([\mathrm{ext}_{G(x)}] \cup \{x\})$
    if $x \in V_G$ **then** PARSE$_V(x)$
    else PARSE$_E(x)$
  return $S \in \mathrm{NT}(\dot{G})$

Algorithm 4 Parsing of Vertices for Order-Preserving HR Grammars.

procedure PARSE$_V$(node v)
  if $\mathrm{out}_G(v) = \emptyset$ **then**
    $\mathrm{NT}(v) \leftarrow \{A \in N \mid A^\bullet \Rightarrow^*_{\mathcal{G}} \bullet\}$ ▷ All A s.t. $A^\bullet$ can derive a single leaf
  else initialize $G_{\mathrm{sh}} = (V, E, \mathrm{att}, \mathrm{lab}, \mathrm{ext})$ as the following shallow graph:
    $E = \{e \in E_G \mid \mathrm{src}_G(e) = v\}$
    $V = \{v\} \cup \bigcup_{e \in E} \mathrm{reent}_G(e)$
    $\mathrm{ext} = \mathrm{ext}_{G(v)}$
    $\mathrm{lab}(e) = \mathrm{NT}(e)$ and $\mathrm{att}(e) = v \cdot [\![\mathrm{reent}_G(e)]\!]_{\prec_G}$ for each $e \in E$
    $\mathrm{NT}(v) \leftarrow$ SHALLOWPARSE($\{r \in R \mid r$ shallow$\}$, G_{sh})

After PARSE has completed the preprocessing of the input graph, it repeatedly finds either a node or an edge x to be processed. Once it has processed x, $\mathrm{NT}(x)$ will be equal to the set of nonterminals A such that $A^\bullet \Rightarrow^* G(x)$. To be able to process x, the requirement is that $\mathrm{NT}(y)$ has already been determined for all nodes and edges y of $G(x)$ except for the external nodes and x itself. Intuitively, the algorithm proceeds bottom-up on the graph, starting by processing leaves, and moving on with any edges whose targets are all either leaves or members of $\mathrm{reent}_G(x)$, moving on with "higher" nodes and edges only when everything "below" them has been processed. (This intuitive view should be taken with a grain of salt, because G can be cyclic.)

When parsing a node v with PARSE$_V$, there are two cases:

- The node is a leaf, which is equivalent to saying that $G(v)$ is the graph $\bullet$ consisting of a single external node and no edges. In this case $G(v)$ can only be derived from a nonterminal of rank zero by applying a series of chain rules, i.e., $\mathrm{NT}(v) = \{A \in N \mid A^\bullet \Rightarrow^* \bullet\}$.
- The node has out-degree $d > 0$, which means that the derivations $A^\bullet \Rightarrow^* G(v)$ are (up to reordering derivation steps) those which begin with applying $d - 1$ duplication rules yielding d nonterminal edges from which the individual graphs

Algorithm 5 Parsing of Edges for Order-Preserving HR Grammars.

```
1: procedure PARSE_E(edge e)
2:     NT ← ∅
3:     for each nonterminal A ∈ N do
4:         for each deep production A → F do
5:             if MATCH(F, e) then                          ▷ G(e) is derivable from F
6:                 NT ← NT ∪ {A}                            ▷ ...and thus from A•
7:     NT(e) ← {A ∈ N | A• ⇒* B• for some B ∈ NT}            ▷ finally, apply closure
```

Algorithm 6 Matching a deep right-hand side F against $G(e)$.

```
1: function MATCH(right-hand side F, edge e ∈ E_G)
2:     φ ← φ-COMPUTATION()F, G, e (return FALSE if this computation fails)
3:     if lab(f) ∈ NT(φ(src(f))) for all nonterminal edges f ∈ E_F then
4:         return TRUE
5:     else
6:         return FALSE
```

$G(e)$ with $\mathrm{src}_G(e) = v$ have been derived. This requires us to identify and "reverse" the expansions. In practice we construct a temporary graph G_{sh} that represents the relevant structures, with one edge for each of the graphs $G(e)$ such that $\mathrm{src}_G(e) = v$. We then call SHALLOWPARSE on this graph to calculate the set of possible nonterminals A such that $A^\bullet$ can yield a suitable collection of edges to derive $G(v)$ from.

PARSE$_\mathrm{E}$ mainly identifies potential matching right-hand sides to the graph $G(e)$ and uses the function MATCH to check their relationship in detail. In particular, we know that the rank of any nonterminal $A^\bullet$ generating $G(e)$ must match the number of reentrant nodes. In fact, since the grammar is order preserving, the ith node in $A^\bullet_{\cdot\cdot}$ corresponds to the ith node in $G(e)_{\cdot\cdot}$. Further, since PARSE$_\mathrm{V}$ handles any applications of duplication rules, PARSE$_\mathrm{E}$ does not have to deal with them.

Checking whether F matches $G(e)$ is easily done, using the mapping ϕ computed by Algorithm 2 (see Corollary 6.8): assuming that the sets $\mathrm{NT}(v)$ have already been computed for all internal nodes of $G(e)$, we just have to check for all nonterminal edges f of F that $\mathrm{lab}_F(f) \in \mathrm{NT}(\phi(\mathrm{src}_F(f)))$. This is done by the procedure MATCH in Algorithm 6. Lemma 6.9 confirms formally that MATCH works correctly.

Lemma 6.9. *Let $e \in E_G$ and assume that $\mathrm{NT}(v) = \{A \in N \mid A^\bullet \Rightarrow^* G(v)\}$ for all nodes $v \in V_{G(e)} \setminus [\mathrm{ext}_{G(e)}]$. For a rule $(A \to F) \in P$, MATCH(F, e) returns TRUE if and only if there is a derivation $F \Rightarrow^* G(e)$.*

Proof. Let $f_1, \ldots, f_k$ be the pairwise distinct nonterminal edges of F. Assume first that MATCH(F, e) returns TRUE, and let $v_i = \phi(\mathrm{src}_F(f_i))$ and $G_i = G(e)(v_i)$ for all $i \in [k]$. By Lemma 6.7(iii), $\mathrm{ext}_{G_i} = \phi(\mathrm{att}_F(f_i))$ for all $i \in [k]$. By the definition of hyperedge replacement, this yields $G(e) \equiv F[f_1 : G_1, \ldots, f_k : G_k]$, and by the test on line 3 of MATCH it holds that $\mathrm{lab}_F(f_i) \in \mathrm{NT}(v_i)$ and thus $\mathrm{lab}_F(f_i)^\bullet \Rightarrow^* G_i$ for all $i \in [k]$. Consequently, by context-freeness (Lemma 2.3) $F \Rightarrow^* F[f_1 : G_1, \ldots, f_k : G_k] \equiv G(e)$, as required.

Conversely, assume that $F \Rightarrow^* G(e)$. Again by context-freeness, this means that there are graphs G_i such that $\mathrm{lab}_F(f_i)^\bullet \Rightarrow^* G_i$ (for all $i \in [k]$) and $H = F[f_1 : G_1, \ldots, f_k : G_k]$ is isomorphic to $G(e)$ by some isomorphism $h \colon H \to G(e)$. In particular, $G_i \in \mathbb{G}_{\mathcal{R}}$, and thus MATCH computes the restriction ϕ of h_V to V_F on line 2, by Corollary 6.8. The condition on line 3 is satisfied because $\mathrm{lab}_F(f_i)^\bullet \Rightarrow^* G_i$ and thus, by assumption, $\mathrm{lab}_F(f_i) \in \mathrm{NT}(\phi(\mathrm{src}_F(f_i)))$ for all $i \in [k]$. Hence, MATCH returns TRUE. □

We are now ready to show that PARSE (Algorithm 3) works correctly on graphs in $\mathbb{G}_{\mathcal{R}}$.

Lemma 6.10. *A call PARSE$(\mathcal{G}, G)$ with $G \in \mathbb{G}_{\mathcal{R}}$ to Algorithm 3 returns true if and only if $G \in \mathcal{L}(\mathcal{G})$.*

Proof. We first argue that, whenever the condition on line 6 is satisfied, there exists an appropriate x on line 7 of PARSE. We use Lemma 3.5, which applies to the subgraphs $G(x)$ because $G{\downarrow_x} \approx G(x)$ (if the latter is defined). Intuitively, since Lemma 3.5 states that the subgraphs $G(x)$ are properly nested, we can always recurse down into them to find an appropriate x. For a more precise argument, choose some $x \in V_G \cup E_G$ with $\mathrm{NT}(x) = \bot$ that minimizes the cardinality of the set Y_x of all $y \in (V_{G(x)} \cup E_{G(x)}) \setminus ([\mathrm{ext}_{G(x)}] \cup \{x\})$ for which $\mathrm{NT}(y) = \bot$. We have to show that $Y_x = \emptyset$. Assume for a contradiction that there is a $y \in Y_x$. If $x \in E_G$, then $G(y) = G(x)(y)$ does not contain x (because $x \notin E^y_{G(x)}$ as it can be reached in $G(x)$ without passing y). As, furthermore, $[\mathrm{ext}_{G(x)}] \cap V_{G(y)} \subseteq [\mathrm{ext}_{G(y)}]$, this yields the contradiction that $|Y_y| < |Y_x|$. The second case is that $x \in V_G$. If y is an edge with $\mathrm{src}_G(y) = x$, then $x \in [\mathrm{ext}_{G(y)}]$ and thus $Y_y \subseteq Y_x \setminus \{x, y\}$, again yielding a contradiction. Otherwise y is a node, or it is an edge with $\mathrm{src}_G(y) \neq x$. Hence x is either not in $V_{G(y)}$ at all, or else it belongs to $G(y)_{\cdot\cdot}$, which in both cases yields $Y_y \subseteq Y_x \setminus \{y\}$ and thus again a contradiction.

Thus, PARSE never gets stuck on line 6. To prove the statement of the lemma, we now show by induction on the number of loop executions the slightly stronger claim that $\mathrm{NT}(x) \neq \bot$ implies

$$\mathrm{NT}(x) = \begin{cases} \{A \in N \mid A^\bullet \Rightarrow^* G(x)\} & \text{if } G(x) \text{ is defined} \\ \emptyset & \text{otherwise.} \end{cases} \tag{4}$$

In particular, since $G = G(\dot{G})$, this means that $S \in \mathrm{NT}(\dot{G})$ if and only if $G \in \mathcal{L}(G)$.

The base case, before the first iteration of the loop, is given on line 5 of PARSE, i.e., $G(x)$ is undefined and $\mathrm{NT}(x) = \emptyset$. Thus, the claim holds in this case.

For the inductive case, assume now that $G(x)$ is defined and x fulfills the condition on line 7 of PARSE. We have to show that, after the execution of $\textsc{parse}_\mathrm{V}(x)$ (if $x \in V_G$) or $\textsc{parse}_\mathrm{E}(x)$ (if $x \in E_G$), it holds that $\mathrm{NT}(x) = \{A \in N \mid A^\bullet \Rightarrow^*_{\mathcal{G}} G(x)\}$.

Assume first that $x \in V_G$. If $\mathrm{out}_G(x) = \emptyset$, then $G(x)$ is the graph $\bullet$ consisting of a single node and no edge, and hence (4) reduces to $\mathrm{NT}(x) = \{A \in N \mid A^\bullet \Rightarrow^* \bullet\}$, which is ensured on line 3 of $\textsc{parse}_\mathrm{V}$. Otherwise, if $\mathrm{out}_G(x) = \{e_1, \ldots, e_k\}$ with $k > 0$, then the derivations $A^\bullet \Rightarrow^* G(x)$ are (by context-freeness and the induction hypothesis) exactly those which can be written in the form $A^\bullet \Rightarrow^* H \Rightarrow^* G(x) = H[f_1 : G(e_1), \ldots, f_k : G(e_k)]$, where H is a shallow graph with $E_H = \{f_1, \ldots, f_k\}$ and $\mathrm{lab}_H(f_i) \in \mathrm{NT}(e_i)$ for all $i \in [k]$. Since $\mathrm{tar}_H(f_i) = G(e_i)_{\bullet\bullet} = \mathrm{reent}_G(e_i)$ for all $i \in [k]$, the graph H is (isomorphic to) a concretization of the graph G_{sh} constructed in $\textsc{parse}_\mathrm{V}$. Conversely, if a concretization H of G_{sh} with $\mathrm{tar}_H(f_i) = G(e_i)_{\bullet\bullet} = \mathrm{reent}_G(e_i)$ for all $i \in [k]$ satisfies $A^\bullet \Rightarrow^* H$, then $A^\bullet \Rightarrow^* H[f_1 : G(e_1), \ldots, f_k : G(e_k)]$. Hence, Lemma 6.5 proves that $\mathrm{NT}(x) = \{A \in N \mid A^\bullet \Rightarrow^*_{\mathcal{G}} G(x)\}$ when line 9 of $\textsc{parse}_\mathrm{V}$ has been executed.

Finally, consider the case where $x \in E_G$. Since $G(x)$ is deep (it contains the terminal edge x), and no edge in $G(x)$ shares its source with x, all derivations $A^\bullet \Rightarrow^* G(x)$ have the form

$$A^\bullet \Rightarrow^* B^\bullet \Rightarrow F \Rightarrow^* G(x) \tag{5}$$

for some deep rule $B \to F$ in R. By the condition on line 7 of PARSE, x satisfies $\mathrm{NT}(v) = \{A \in N \mid A^\bullet \Rightarrow^* G(v)\}$ for all nodes $v \in V_{G(x)} \setminus [\mathrm{ext}_{G(x)}]$, i.e., on line 5 of $\textsc{parse}_\mathrm{E}$ the condition for applying Lemma 6.9 is fulfilled. Consequently, on line 7 of $\textsc{parse}_\mathrm{E}$, *NT* equals the set of all $B \in N$ such that there is a deep rule $B \to F$ with $F \Rightarrow^* G(x)$. Hence, by (5), the assignment on line 7 of $\textsc{parse}_\mathrm{E}$ yields $\mathrm{NT}(x) = \{A \in N \mid A^\bullet \Rightarrow^* G(x)\}$, as claimed. □

We are now ready to generalize our result to graphs that do not necessarily belong to $\mathbb{G}_{\mathcal{R}}$ and give time bounds. This yields the main result of the paper. The time bounds given are relative to the time it takes to compute $\prec_G$. In Section 5 we saw that there are suitable and natural families of orders that can efficiently be computed.

Theorem 6.11. *Let $\prec$ be a suitable family of orders that is preserved by a set $\mathcal{R} \subseteq \mathcal{C}$ of HR rules, and let $f : \mathbb{G}_{\mathcal{R}} \to \mathbb{N}$ be a function such that both $f(G)$ and $\prec_G$ can be computed in time $f(G)$ for every graph $G \in \mathbb{G}_{\mathcal{R}}$.*[11] *Then there is an algorithm which takes as input a graph G and an HR grammar $\mathcal{G} = (\Sigma, N, S, R)$ with $R \subseteq \mathcal{R}$, and decides in time $\mathcal{O}(f(G) + |G|^2 + |\mathcal{G}|^2)$ whether $G \in \mathcal{L}(\mathcal{G})$.*

Proof. Assume first that $G \in \mathbb{G}_{\mathcal{R}}$. By Lemma 6.10 and the definition of PARSE, $\textsc{parse}(\mathcal{G}, G)$ returns true if and only if $G \in \mathcal{L}(\mathcal{G})$. By assumption, we can compute $\prec_G$ in time $f(G)$ and by Lemma 6.6 the family $(\mathrm{reent}_G(e))_{e \in E_G}$ can be computed in time $\mathcal{O}(|G|^2)$. As pointed out after (the proof of) Lemma 6.5, we can precompute $\mathit{RELAB} = \{(A, B) \in N^2 \mid A^\bullet \Rightarrow^* B^\bullet\}$ in time $|\mathcal{G}|^2$. Algorithm 6 (MATCH) runs in linear time in the size of the right-hand side F. $\textsc{parse}_\mathrm{V}$ and $\textsc{parse}_\mathrm{E}$ are called once for each node and edge, respectively. In particular, the calls of $\textsc{parse}_\mathrm{E}$ invoke MATCH at most $|E_G| \cdot |R|$ times in total, thus altogether taking at most $\mathcal{O}(|E_G| \cdot |\mathcal{G}|)$ computation steps if we make use of the precomputed set *RELAB* to implement line 7 of $\textsc{parse}_\mathrm{E}$ efficiently. Moreover, $\textsc{parse}_\mathrm{V}$ runs in constant time, except for the construction of G_{sh} and the call to SHALLOWPARSE. The total size of the graphs that are passed to SHALLOWPARSE cannot exceed $|G|$ and SHALLOWPARSE itself runs in time $|G|^2$ by Lemma 6.5 (where we again make use of the precomputed set *RELAB*).[12] Hence, the total contribution is $\mathcal{O}(|G|^2)$. Altogether, this yields the upper bound $\mathcal{O}(f(G) + |G|^2 + |\mathcal{G}|^2)$ on the running time of $\textsc{parse}(\mathcal{G}, G)$ for $G \in \mathbb{G}_{\mathcal{R}}$.

Now, let us drop the assumption that $G \in \mathbb{G}_{\mathcal{R}}$. Let c be such that PARSE runs in time at most $g(\mathcal{G}, G) = c \cdot (f(G) + |G|^2 + |\mathcal{G}|^2))$ on input graphs in $\mathbb{G}_{\mathcal{R}}$. We use the old yardstick trick of computational complexity to make sure that the algorithm terminates in time $\mathcal{O}(g(\mathcal{G}, G))$: initially, $M = g(\mathcal{G}, G)$ is computed (which is possible because $f(G)$ can be computed in time $f(G)$), and a counter is initialized to zero. Then the original algorithm is executed, incrementing the counter after each computation step. If the counter reaches the value M before the algorithm terminates, the input is rejected (because this, by the correctness of the algorithm on $\mathbb{G}_{\mathcal{R}}$, proves that $G \notin \mathbb{G}_{\mathcal{R}}$).

This makes sure that the running time stays within $\mathcal{O}(g(\mathcal{G}, G)) = \mathcal{O}(f(G) + |G|^2 + |\mathcal{G}|^2)$ even for graphs not in $\mathbb{G}_{\mathcal{R}}$. However, we still have to make sure that such graphs are indeed rejected. For this, note that our parsing procedure, if it accepts G, actually determines a concrete derivation, as MATCH determines the mapping ϕ of nodes and edges of right-hand sides to those in G, i.e., it yields the isomorphic copies $A \to F'$ of rules $(A \to F) \in R$ that need to be applied in order to generate the concrete graph G. A similar mapping is obtained in SHALLOWPARSE because its parameter G_{sh}, constructed in $\textsc{parse}_\mathrm{V}$, uses the actual nodes of V_G (see line 6 of $\textsc{parse}_\mathrm{V}$). Thus, we can extend the algorithm in such a way that it, for all $A \in \mathrm{NT}(e)$, stores the concrete isomorphic copy $A \to F'$ of the rule $(A \to F) \in R$ applied to e. Now, having this information,

[11] The first condition corresponds to the usual condition of time-constructibility, but here for a function that takes graphs as input, rather than for an ordinary complexity function.

[12] It may be worth noting that this is a very pessimistic estimation because the graphs G_{sh} will typically have considerably fewer edges than G.

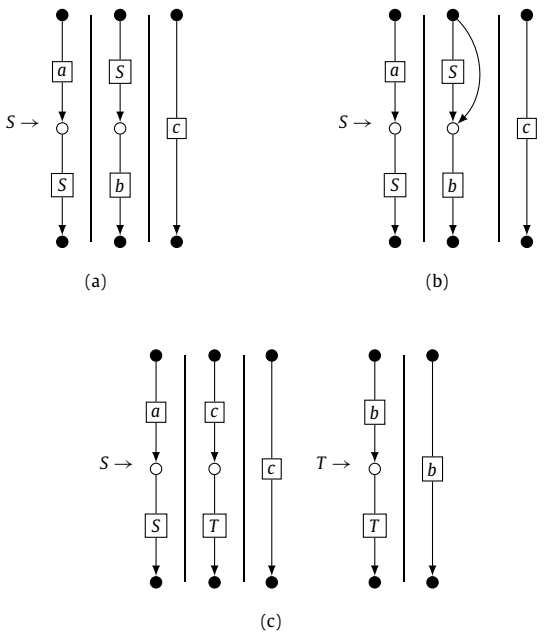

Fig. 11. A string graph grammar G_{HR} obtained from a CFG G (a), a reentrancy-reserving variant G_{HR^+} with additional terminal edges (b), and the string graph grammar $G'_{\mathrm{HR}} = G'_{\mathrm{HR}^+}$ resulting from a corresponding right-linear grammar G' (c).

upon successful termination of PARSE it is easy to reconstruct in time $\mathcal{O}(f(G) + |G| \cdot |\mathcal{G}|)$ the concrete derivation, thereby checking that it is indeed a consistent derivation and rejecting if it is not. Upon success, we finally check the derived graph and G for equality (rather than isomorphism!). If they are indeed equal, then by definition $G \in \mathcal{L}(\mathcal{G})$. If they are not, then the correctness of PARSE on graphs in $\mathbb{G}_{\mathcal{R}}$ implies that $G \notin \mathbb{G}_{\mathcal{R}}$ and thus, in particular, $G \notin \mathcal{L}(\mathcal{G})$. □

We note here that, depending on the choice of $\mathcal{R}$ and $\prec$, it may often be possible to handle the case where the input graph is not an element of $\mathbb{G}_{\mathcal{R}}$ in another way, thus avoiding the explicit use of the yardstick argument and the construction of the derivation with the final equality check. In fact, it may be interesting to study the case where PARSE returns a positive result and thus a derivation of a graph $G' \in \mathcal{L}(\mathcal{G})$ even though $G \notin \mathcal{L}(\mathcal{G})$. This could be seen as a best effort to derive a graph G' similar to G. We will not further pursue this line of thought in the current paper, however.

For the case of the family $\lhd$ defined and discussed in Section 5, Theorem 6.11 in combination with Lemma 5.7 yields the following corollary by setting $f(G) = r|G|^2$ and observing that $f(G)$ can certainly be computed in time $f(G)$.

Corollary 6.12. *Algorithm 3, instantiated with the family $\lhd$ of orders (with $\mathcal{R}$ as in Section 5), runs in time $O(r|G|^2 + |\mathcal{G}|^2)$, where G is the input graph, r its inner rank, and $\mathcal{G}$ the input HR grammar.*

As parsing for context-free string grammars (CFGs) is already cubic, and HR grammars generalize CFGs, one might wonder how it is possible to parse "our" hypergraph grammars in quadratic time. The answer is that, while unrestricted HR grammars generalize CFGs, reentrancy-preserving ones do not. We illustrate this with an example.

Example 6.13. To simplify this example, we will consider HR grammars which satisfy (P1) and (P3), but not necessarily (P2). We can do so because the purpose of this example is to illustrate the *limitations* of order-preserving HR grammars.

Consider the CFG given by the rules $S \to aS \mid Sb \mid c$, where S is nonterminal and a, b, c are terminal. It generates the (regular) language denoted by the regular expression a^*cb^*. Representing strings as chains of edges of rank 1, it is well-known how to turn such a CFG G into an HR grammar G_{HR} that generates so-called string graphs. Taking the present CFG to be G, G_{HR} is shown in Fig. 11(a). However, the second rule of this HR grammar violates (P3): the target node of the nonterminal edge is not reentrant. (Note that the first rule does *not* violate (P3).) A way to make it reentrant is to accompany the nonterminal edge by a parallel terminal one, as shown in Fig. 11(b) (where we omit the label of the additional edge because it is not relevant). This yields what we may call an extended string graph grammar G_{HR^+}. The string *aaacbb*, for instance, will then be represented by several graphs in the generated graph language, such as the one in Fig. 12 (which we have turned by 90 degrees). The additional edges reveal the basic structure of the derivation that has produced it: $S \to Sb \to aSb \to aaSb \to aaSbb \to aaaSbb \to aaacbb$.

If we turn the CFG into the equivalent right-linear grammar G' given by the rules $S \to aS \mid cT \mid c$, $T \to bT \mid b$, the very same construction yields the reentrancy-preserving HR grammar in Fig. 11(c): since every nonterminal edge e in a

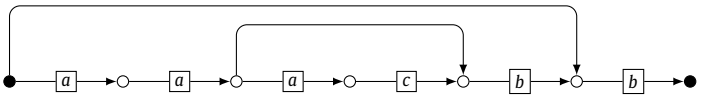

Fig. 12. A graph generated by G_{HR^+} revealing the derivation $S \to Sb \to aSb \to aaSb \to aaSbb \to aaaSbb \to aaacbb$.

right-hand side F satisfies $\mathrm{tar}_G(e) = F_{\bullet\bullet}$, no additional edges as in Fig. 11(b) result, and thus $G_{\mathrm{HR}} = G_{\mathrm{HR}^+}$. Clearly, these observations generalize, showing that, for every CFG G, G_{HR} is reentrancy preserving and thus equal to G_{HR^+} if and only if G is right-linear.

7. Conclusions and future work

Having developed an axiomatic notion of order-preserving HR grammars that enables uniform polynomial-time parsing, and having discussed a particular instantiation in the form of a suitable family of orders and a set of HR rules preserving it, several possible directions for future work remain. One is the study of suitable orders, their formal properties, and the sets of rules which preserve them. A better understanding of what characterizes suitable orders and order-preserving types of rules could make it easier to find additional ones. A related question is whether and in which cases it may be possible to infer a suitable order for a given set of rules "on the fly". Suppose, for example, that a given set of rules does not preserve the family $\lhd$ defined in Section 5. The reason may simply be that the targets of edges must be permuted depending on the labels. Is there an efficient way to determine such a permutation if it exists?

Our running example illustrates that order-preserving HR grammars can capture nontrivial structures of semantic graphs from Natural Language Processing, where they are used to represent sentence meaning. An example is Abstract Meaning Representation, AMR [2]. At the same time, it is clear that the imposed restrictions are substantial. Moreover, not even unrestricted hyperedge replacement can generate all structurally correct AMR graphs. This is obvious from the fact that HR grammars can only generate graph languages of bounded treewidth, whereas the AMR definition allows for the construction of graphs of arbitrary treewidth. For this reason, an interesting practical line of further inquiry would be to determine the empirical coverage of AMR that can be achieved with order-preserving HR grammars. In this context, it is worth mentioning that [8] report the maximum treewidth of graphs in the AMR bank to be 4, in sharp contrast to the above-mentioned theoretical unboundedness of the treewidth of AMR graphs. Future theoretical work with the aim to extend the coverage of the present work to graph languages of unbounded treewidth may try to extend it to the so-called contextual HR grammars [12,15].

We note finally that the results of this paper have already been extended to hyperedge replacement grammars with weights in [6]. Being able to incorporate weights in order to model probabilities is essential for many applications in Natural Language Processing. The results in the mentioned paper show that our approach carries over to the weighted case without efficiency loss.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

We are very grateful to the reviewers whose comments have – as we believe – led to a significant improvement of the readability of this paper.

References

[1] I.J. Aalbersberg, A. Ehrenfeucht, G. Rozenberg, On the membership problem for regular DNLC grammars, Discrete Appl. Math. 13 (1986) 79–85.
[2] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, N. Schneider, Abstract meaning representation for sembanking, in: Proc. 7th Linguistic Annotation Workshop, ACL 2013, 2013.
[3] M. Bauderon, B. Courcelle, Graph expressions and graph rewriting, Math. Syst. Theory 20 (1987) 83–127.
[4] H. Björklund, J. Björklund, P. Ericson, On the regularity and learnability of ordered DAG languages, in: Proc. 22nd International Conference on the Implementation and Application of Automata, CIAA'17, Springer, 2017, pp. 27–39.
[5] H. Björklund, F. Drewes, P. Ericson, Between a rock and a hard place – uniform parsing for hyperedge replacement DAG grammars, in: A. Dediu, J. Janoušek, C. Martín-Vide, B. Truthe (Eds.), Proc. 10th Intl. Conf. on Language and Automata Theory and Applications, 2016, pp. 521–532.
[6] H. Björklund, F. Drewes, P. Ericson, Parsing weighted order-preserving hyperedge replacement grammars, in: Proc. 16th Meeting on the Mathematics of Language, MoL 2019, 2019, pp. 1–11, https://www.aclweb.org/anthology/W19-5701/.
[7] D. Chiang, J. Andreas, D. Bauer, K.M. Hermann, B. Jones, K. Knight, Parsing graphs with hyperedge replacement grammars, in: Proc. 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013), vol. 1: Long Papers, 2013, pp. 924–932.
[8] D. Chiang, F. Drewes, D. Gildea, A. Lopez, G. Satta, Weighted DAG automata for semantic graphs, Comput. Linguist. 44 (2018) 119–186.
[9] F. Drewes, NP-completeness of k-connected hyperedge-replacement languages of order k, Inf. Process. Lett. 45 (1993) 89–94.
[10] F. Drewes, Recognising k–connected hypergraphs in cubic time, Theor. Comput. Sci. 109 (1993) 83–122.

[11] F. Drewes, A. Habel, H.J. Kreowski, Hyperedge replacement graph grammars, in: G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations, World Scientific, 1997, pp. 95–162, chapter 2.
[12] F. Drewes, B. Hoffmann, Contextual hyperedge replacement, Acta Inform. 52 (2015) 497–524.
[13] F. Drewes, B. Hoffmann, M. Minas, Predictive top-down parsing for hyperedge replacement grammars, in: Proc. 8th Intl. Conf. on Graph Transformation, ICGT'15, 2015.
[14] F. Drewes, B. Hoffmann, M. Minas, Predictive shift-reduce parsing for hyperedge replacement grammars, in: J. de Lara, D. Plump (Eds.), Proc. 10th Intl. Conf. on Graph Transformation, ICGT'17, 2017, pp. 106–122.
[15] F. Drewes, A. Jonsson, Contextual hyperedge replacement grammars for abstract meaning representations, in: 13th Intl. Workshop on Tree-Adjoining Grammar and Related Formalisms, TAG+13, Association for Computational Linguistics, 2017, pp. 102–111, http://aclweb.org/anthology/W17-6211.
[16] S. Gilroy, A. Lopez, S. Maneth, Parsing graphs with regular graph grammars, in: Proc. 6th Joint Conf. on Lexical and Computational Semantics, *SEM 2017, 2017, pp. 199–208.
[17] A. Habel, Hyperedge Replacement: Grammars and Languages, Lecture Notes in Computer Science, vol. 643, Springer, 1992.
[18] A. Habel, H.J. Kreowski, May we introduce to you: hyperedge replacement, in: Proceedings of the Third Intl. Workshop on Graph Grammars and Their Application to Computer Science, Springer, 1987, pp. 15–26.
[19] K.J. Lange, E. Welzl, String grammars with disconnecting or a basic root of the difficulty in graph grammar parsing, Discrete Appl. Math. 16 (1987) 17–30.
[20] C. Lautemann, The complexity of graph languages generated by hyperedge replacement, Acta Inform. 27 (1990) 399–421.
[21] W. Vogler, Recognizing edge replacement graph languages in cubic time, in: H. Ehrig, H.J. Kreowski, G. Rozenberg (Eds.), Proceedings of the Fourth Intl. Workshop on Graph Grammars and Their Application to Computer Science, in: Lecture Notes in Computer Science, vol. 532, Springer, 1991, pp. 676–687.