Polynomial Graph Parsing with Non-Structural Reentrancies

Johanna Björklund
Dept. Computing Science
Umeå University
Umeå, Sweden
johanna@cs.umu.se

Frank Drewes
Dept. Computing Science
Umeå University
Umeå, Sweden
drewes@cs.umu.se

Anna Jonsson
Dept. Computing Science
Umeå University
Umeå, Sweden
aj@cs.umu.se

Abstract

Graph-based semantic representations are valuable in natural language processing, where it is often simple and effective to represent linguistic concepts as nodes, and relations as edges between them. Several attempts has been made to find a generative device that is sufficiently powerful to represent languages of semantic graphs, while at the same allowing efficient parsing. We add to this line of work by introducing graph extension grammar, which consists of an algebra over graphs together with a regular tree grammar that generates expressions over the operations of the algebra. Due to the design of the operations, these grammars can generate graphs with *non-structural reentrancies*, a type of node-sharing that is excessively common in formalisms such as abstract meaning representation, but for which existing devices offer little support. We provide a parsing algorithm for graph extension grammars, which is proved to be correct and run in polynomial time.

Index Terms

Algorithmic Graph Theory, Semantic Representations, Hyperedge Replacement Grammars, Natural Language Processing

I. Introduction

Semantic representations are a key component in algorithmic solutions for high-level language processing. Word embeddings such as BERT [14] and GPT-3 [9] have become a popular choice of representation, enabling rapid progress on many tasks. Due to the descriptive power of the neural networks that generate them, they rarely submit to analysis by deductive means. However, for some applications the transparency gained by formal methods is a must. In such cases, we believe that graph-based semantic representations offer a useful complement. These generally represent objects as nodes, and relations as directed edges. Probabilities and other weights can be added to reflect quantitative aspects such as likelihoods and uncertainties.

In this work, we model languages of graph-based semantic representations as the combination of a graph algebra $\mathcal{A}$ together with a tree grammar g whose generated trees are well-formed

This work has been supported in part by the Swedish Research Council under Grant No 2020-03852.

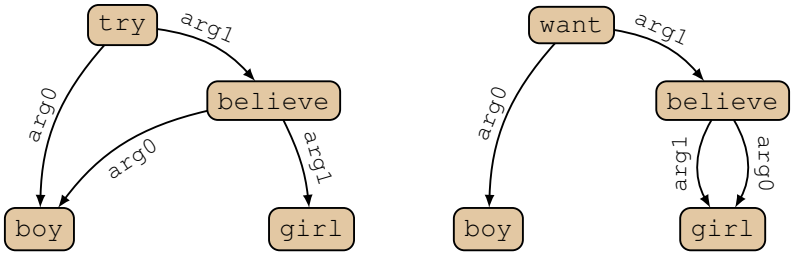


Fig. 1. AMR graphs for the pair of sentences "The boy tries to believe the girl" and "The boy wants the girl to believe herself". The nodes represent the concepts 'boy', 'girl', 'try', 'want', and 'believe'. The edges labelled 'arg0' give the agent for each verb, and the edges labelled 'arg0' the patient.

expressions over the operations of $\mathcal{A}$. Each tree translates into a graph when evaluated by $\mathcal{A}$, meaning that the tree language generated by g evaluates to a graph language. If we are careful about how we construct and combine g and $\mathcal{A}$, we can make parsing efficient. In other words, given a graph G, we can in polynomial time find a tree in the language of g that evaluates to G under $\mathcal{A}$ (or decide that no such tree exists). In the literature, this approach is known as tree-based generation [15]. The novelty in the present work lies in the design of the algebra.

For inspiration, we look at a particular instance of graph-based representations, namely abstract meaning representation (AMR) [30], [2]. It is characterised by its graphs being directed, acyclic, and having unbounded node degree. AMR was first introduced by [30] based on a semantic abstraction language by [28]. The notion was refined and popularised by [2] and instantiated for a limited domain by [8]. To ground AMR in formal language theory, Chiang et al. [11] analyse the AMR corpus of Banarescu et al [2]. They note that even though the node degree is generally low in practice, this is not always the case, which speaks in favour of models that allow an unbounded node degree. Regarding the treewidth of the graphs in the corpus, they find that it never exceeds 4 and conclude that an algorithm can depend exponentially on this parameter and still be feasible in practice.

In the context of semantic graphs, it is common to talk about *reentrancies*. Figure 1 illustrates this concept with a pair of AMR graphs, both of which require node sharing, or *reentrant edges*, to express the correct semantics. We propose to distinguish between two types of reentrancies: In the first graph, the reentrancy is *structural*, meaning that the boy must be the agent of whatever he is trying, so the 'arg0' edge can only point to him. In the second graph, the reentrancy is *non-structural* in the sense that although in this particular case the girl believes in herself, there is nothing to prevent her believing in someone else, or that someone else believes in her. In general, we speak of (i) structural reentrancies when they are syntactically governed, e.g., by control or coordination, and of (ii) non-structural reentrancies when they represent coreferences which can in principle refer to any antecedent, essentially disregarding the structure of the graph. Another example of (i) is subject control as in "They persuaded him to talk to her", where the person who talks and the person who was persuaded must be one and the same. In contrast, "[. . .], but she liked them" is an example of (ii) since antecedent of "them" may be picked from

anywhere in "[...]" for the semantic representation to be valid.

Another important characteristic of AMR and other notions of semantic graphs in natural language processing is that certain types of edges, like those in Figure 1, point to *arguments* of which there should be at most one. For example, the concept 'try' in Figure 1 must only have one outgoing 'arg0' edge. Other edges, of types not present in Figure 1, can for example represent modifiers, such as believing *strongly* and *with a passion*. Outgoing edges of these types are usually not limited in number.

We have previously shown that contextual hyperedge-replacement grammars (CHRGs), an extension of the well-known context-free hyperedge-replacement grammars (HRGs), can handle both structural and non-structural reentrancies [22]. Moreover, a parser generator has been provided by Drewes, Hoffmann, and Minas for CHRGs that, when it succeeds in producing a parser, guarantees that the parser will run in quadratic (and in the common case, linear) time in the size of its input graph [19], [21]. However, similarly to common LL- and LR-parsers for context-free languages the parser generator may discover a parsing conflict, thus failing to output a parser. Compared to the string case, the possible reasons for conflicts are much subtler, which makes grammar construction a complex and error-prone endeavour.

For this reason, we are now proposing a CHRG variant that by construction allows for polynomial parsing while retaining the ability to specify graph languages with reentrancies of both type (i) and (ii). One key property of these grammars that enables polynomial parsing is that, intuitively, nodes are provided with all of their outgoing edges the moment they are created. Using ordinary contextual hyperedge replacement, this would thus result in graph languages of bounded out-degree, the bound being given by the maximal number of outgoing edges of the nodes in the right-hand sides of production rules. To be able to generate semantic graphs in which a concept can have an arbitrary number of edges representing modifiers, we use a technique from [17] known as *cloning*. To incorporate both contextuality and cloning in a natural way without sacrificing efficient parsability, we formalise our grammars using the concept of *tree-based graph generation*. While tree-based hyperedge-replacement grammars are well-known to be equivalent to ordinary ones (see below), the tree-based formulation does make a difference in the contextual case as it avoids the problem of cyclic dependencies that has not yet been fully characterised and can make parsing intractable [20].

A. Related work

Tree-based generation dates back to the seminal paper by Mezei and Wright [34] which generalises context-free languages to languages over arbitrary domains, by evaluating the trees of a regular tree language with respect to an algebra.[1] Operations on graphs were used in this way for the first time by Bauderon and Courcelle [3]. See the textbook [13] by Courcelle and Engelfriet for the eventual culmination of this line of work. Graph operations similar to those used here (though without the contextual extension) appeared first in Courcelle's work on the monadic-second order logic of graphs [12, Definition 1.7]. Essentially, if the right-hand side of a production contains k nonterminal hyperedges, it is viewed as an operation that takes k

[1]Mezei and Wright formulate this in terms of systems of equations, but the essential ideas are the same.

hypergraphs as arguments (corresponding to the hypergraphs generated by the k subderivations) and returns the hypergraph obtained by replacing the nonterminals in the right-hand side by those k argument hypergraphs. By context-freeness, evaluating the tree language that corresponds to the set of derivation trees of the grammar (which is a regular tree language) yields the same set of hypergraphs as is generated by the original HRG.

Several formalisms have been put forth in the literature to describe graph-based semantic representations in general and AMR in particular. Most of these can be seen as variations of HRGs; see [27], [3], [23]. It was established early that unrestricted HRGs can generate NP-complete graph languages [1], [29], so restrictions are needed to ensure efficient parsing. To this end, Lautemann [31] proposed a CYK-like membership algorithm and proved that it runs in polynomial time provided that the language satisfies the following condition: for every graph G in the language, the number of connected components obtained by removing s nodes from G is in $O(\log n)$, where n is the number of nodes of G and the constant s depends on the grammar. Lautemann's algorithm was refined by Chiang et al. [10] to make it more suitable for NLP tasks, but the algorithm is exponential in the node degree of the input graph.

In a parallel line of work, Quernheim and Knight [35] define automata for directed acyclic graphs (DAGs) to process feature structures in machine translation. Chiang et al. [11] propose an extended model of DAG automata, focusing on semantic representations such as AMR. Towards this end, they extend DAG automata by allowing the left- and right-hand sides to have the form of a restricted regular expression. Blum and Drewes [7] complement this work by studying language-theoretic properties of the DAG automata. They establish, among other things, that equivalence and emptiness are decidable in polynomial time.

Various types of hyperedge replacement graph algebras for AMR parsing are described in the work by Groschwitz et al., see, e.g. [25], [24], [26], [32], [33]. A central objective is to find linguistically motivated restrictions that can efficiently be trained from data. An algorithm based on such an algebra for translating strings into semantic graphs was presented in [26]: operations of arity zero denote graph fragments, and operations of arity two denote binary combinations of graph fragments into larger graphs. The trees over the operations of this algebra mirror the compositional structure of semantic graphs. The approach differs from ours in that the graph operations are entirely deterministic, and that neither context nodes nor cloning are used. Moreover, as is common in computational linguistics, evaluation is primarily empirical.

Through another set of syntactic restrictions on HRGs, Björklund et al. [6], [5] arrive at order-preserving dag grammars (OPDGs) – with or without weights – which can be parsed in linear time. Intuitively, the restrictions combine to ensure that each generated graph can be uniquely represented by a tree in a particular graph algebra. Despite their restrictions, OPDGs can describe central structural properties of AMR, but their limitation lies in the modelling of reentrancies. Out of the previously discussed types of reentrancies, type (i) can to a large extent be modelled using OPDGs. Modelling type (ii) cannot be done (except in very limited cases) since it requires attaching edges to the non-local context in an stochastic way, which cannot be achieved using hyperedge replacement alone.

CHRGs [18] extend the ordinary HRGs with so-called *contextual* rules, which allow for

isolated nodes in their left-hand-sides. Contextual rules can reference previously generated nodes, that are not attached to the replaced nonterminal hyperedge, and add structure to them. Even though this formalism is strictly more powerful than HRG, it inherits several of the nice properties of HRG. In particular, there are useful normal forms and the membership problem is in NP [16]. Paralleling previous work on HRG, we set out to find a syntactically restricted subclass of CHRG, enriched by a benign form of cloning, that is sufficiently descriptive to model AMR, while having a membership problem in P.

II. Graphs and Graph Extension Grammars

We first recall some standard definitions and notations from discrete mathematics, automata theory, and algorithmic graph theory. The set of natural numbers (including 0) is denoted by $\mathbb{N}$, and for $n \in \mathbb{N}$, $[n]$ denotes $\{1, \dots, n\}$. In particular, $[0] = \emptyset$. For a set S, the set of all finite sequences over S is denoted by S^*; the subset of S^* containing only those sequences in which no element of S occurs twice is denoted by $S^{\circledast}$. Both contain the empty sequence ε. For $w \in S^*$, $[w]$ denotes the set of all elements of S occurring in w.[2] The canonical extensions of a function $f\colon S \to T$ to S^* and to $\wp(S)$ are denoted by f as well, i.e., $f(s_1 \cdots s_n) = f(s_1) \cdots f(s_n)$ for $s_1, \dots, s_n \in S$ and $f(S') = \{f(s) \mid s \in S'\}$ for all $S' \subseteq S$. The restriction of $f\colon S \to T$ to $S' \subseteq S$ is denoted by $f|_{S'}$. The power set of S is denoted by $\wp(S)$. For functions $f_1\colon S_1 \to T_1$ and $f_2\colon S_2 \to T_2$ by $f_1 \cup f_2$, we let $f_1 \sqcup f_2\colon S_1 \cup S_2 \to T_1 \cup T_2$ be given by

$$(f_1 \sqcup f_2)(s) = \begin{cases} f_1(s) & \text{if } s \in S_1 \\ f_2(s) & \text{if } s \in S_2 \setminus S_1 \end{cases} .$$

Trees and Algebras

As sketched in the introduction, we generate trees over an algebra $\mathcal{A}$ of graph operations and then evaluate each tree t in a non-deterministic fashion to a set of graphs $val_{\mathcal{A}}(t)$. To be able to ensure that the graph operations are always well defined on their arguments, we work with *typed* trees. Let $\mathcal{T}$ be a set of *types*. A *$\mathcal{T}$-typed signature* is a pair $A = (\Sigma, \sigma)$ such that Σ is a finite set of symbols and σ is a function that assigns to every $f \in \Sigma$ a pair $\sigma(f) \in \mathcal{T}^* \times \mathcal{T}$. We usually keep σ implicit, thus identifying A with Σ, and write $f\colon \tau_1 \cdots \tau_k \to \tau$ to indicate that $\sigma(f) = (\tau_1 \cdots \tau_k, \tau)$. The number k is said to be the *rank* of f in Σ. Furthermore, we may drop the epithet $\mathcal{T}$-typed when speaking of signatures in case $\mathcal{T}$ is of little relevance or understood from context.

The family $\mathrm{T}_\Sigma = (\mathrm{T}_\Sigma^\tau)_{\tau \in \mathcal{T}}$ of all well-formed trees over a signature Σ is defined inductively, as usual. It is the smallest family of sets T_Σ^τ of formal expressions such that, for every $f\colon \tau_1 \cdots \tau_k \to \tau$ and all $t_1 \in \mathrm{T}_\Sigma^{\tau_1}, \dots, t_k \in \mathrm{T}_\Sigma^{\tau_k}$, we have $f[t_1, \dots, t_k] \in \mathrm{T}_\Sigma^\tau$. Note that, for every tree $t \in \bigcup_{\tau \in \mathcal{T}} \mathrm{T}_\Sigma^\tau$ there is a unique type τ such that $t \in \mathrm{T}_\Sigma^\tau$. Therefore, we may simply write $t \in \mathrm{T}_\Sigma$ and call τ the type of t.

We refer to the subtrees of a tree by their Gorn addresses. Thus, we define the set $addr(t) \subseteq \mathbb{N}^*$ of *addresses* in a tree t, and the subtrees they designate, in the usual way: for $t = f[t_1, \dots, t_k]$,

[2]The similarity to the notation $[n]$, $n \in \mathbb{N}$, is intentional, since both operations turn single elements into sets.

$addr(t) = \{\varepsilon\} \cup \bigcup_{i\in[k]}\{i\alpha \mid \alpha \in addr(t_i)\}$ (note that the basis of the recursion is the case $k = 0$). Moreover, for all $\alpha \in addr(t)$

$$t/\alpha = \begin{cases} t & \text{if } \alpha = \varepsilon \\ t_i/\alpha' & \text{if } \alpha = i\alpha' \text{ for some } i \in [k] \text{ and } \alpha' \in addr(t_i) \end{cases} .$$

Let Σ be a $\mathcal{T}$-typed signature. A Σ-*algebra* is a pair $\mathcal{A} = ((\mathbb{A}_\tau)_{\tau\in\mathcal{T}}, (f_\mathcal{A})_{f\in\Sigma})$ where $\mathbb{A}_\tau$ is a set for each $\tau \in \mathcal{T}$ and $f_\mathcal{A}$ is a function $f_\mathcal{A} \colon \mathbb{A}_{\tau_1} \times \cdots \times \mathbb{A}_{\tau_k} \to \mathbb{A}_\tau$ for every $f \colon \tau_1 \cdots \tau_k \to \tau$ in Σ. Given a tree $t \in \mathrm{T}_\Sigma$, the result of evaluating t with respect to $\mathcal{A}$ is denoted by $val_\mathcal{A}(t)$. It is defined recursively: if $t = f[t_1, \ldots, t_k]$ then

$$val_\mathcal{A}(t) = f_\mathcal{A}(val_\mathcal{A}(t_1), \ldots, val_\mathcal{A}(t_k)) .$$

Regular Tree Languages

To generate trees (i.e., expressions) over the operations of an algebra, we use *regular tree grammars*. Our interest is in algebras over graph domains, but to illustrate the general idea, we can look at an algebra over the rational numbers $\mathbb{Q}$. In this setting, the operations could be the standard arithmetic ones, such as $+, -, /, \times$, all binary, together with constant-valued operations in $\mathbb{Q}$, all of arity zero. It is hopefully easy to see how the trees $+[\times[2, \frac{1}{2}], 2]$ and $/[10, -[10, 4]]$ evaluate to 2 and $\frac{5}{3}$, respectively. Moreover, by introducing types, we may work with an algebra over a heterogeneous domain, e.g., natural numbers and rationals, and use the types to ensure that when a tree is evaluated, operations are always applied to suitable arguments.

Definition 1 (regular tree grammar)**.** *Let $\mathcal{T}$ be a set of types. A $\mathcal{T}$-typed* regular tree grammar *(or, for brevity, regular tree grammar) is a system $g = (N, \Sigma, P, S)$ consisting of the following components:*

- *N is a $\mathcal{T}$-typed signature of symbols of rank 0 called* nonterminals*; we write N_τ to denote the set of all nonterminals $A \in N$ such that $A \colon \varepsilon \to \tau$, where $\tau \in \mathcal{T}$.*
- *Σ is a $\mathcal{T}$-typed signature of* terminals *which is disjoint with N.*
- *P is a finite set of* productions *of the form $A \to f[A_1, \ldots, A_k]$ where, for some types $\tau, \tau_1, \ldots, \tau_k \in \mathcal{T}$, we have $f \colon \tau_1 \cdots \tau_k \to \tau$, $A_i \in N_{\tau_i}$ for all $i \in [k]$, and $A \in N_\tau$.*
- *$S \in N_\tau$ is the* initial nonterminal *for some type $\tau \in \mathcal{T}$.*

We also say that g is a regular tree grammar over *Σ.*

Definition 2 (regular tree language)**.** *Let $g = (N, \Sigma, P, S)$ be a regular tree grammar. The family $(L_A(g))_{A\in N}$ is the smallest family of sets of trees such that, for all $A \in N$, $L_A(g)$ is the set of all trees $f[t_1, \ldots, t_k]$ such that there is a production $(A \to f[A_1, \ldots, A_k]) \in P$ for some $k \in \mathbb{N}$ and $A, A_1, \ldots, A_k \in N$, and $t_i \in L_{A_i}(g)$ for all $i \in [k]$. The* regular tree language *generated by g is $L(g) = L_S(g)$.*

Note that, as can be shown by a straightforward induction, $L_A(g) \subseteq \mathrm{T}^\tau_\Sigma$ for all $A \in N_\tau$. In particular $L(g) \subseteq \mathrm{T}^\tau_\Sigma$ where τ is the type of S, i.e., $S \in N_\tau$.

Graphs and Graph Operations

We now define the type of graphs used in this paper, and the operations used to construct them. In short, we work with node- and edge-labelled directed graphs, each equipped with a finite set of ports. From a graph operation point of view, the sequence of ports is the "interface" of the graph; its nodes are the only ones that can individually be accessed. To ensure that the resulting graph is well-formed, we need to pay attention to the number of ports, which determines the *type* of the graph.

Definition 3 (graph)**.** *A* labelling alphabet *(or simply* alphabet*) is a pair* $\mathbb{L} = (\dot{\mathbb{L}}, \bar{\mathbb{L}})$ *of finite sets* $\dot{\mathbb{L}}$ *and* $\bar{\mathbb{L}}$ *of labels. A* graph *(over* $\mathbb{L}$*) is a system* $G = (V, E, lab, port)$ *where*

- V *is a finite set of* nodes*,*
- $E \subseteq V \times \bar{\mathbb{L}} \times V$ *is a (necessarily finite) set of* edges*,*
- $lab \colon V \to \dot{\mathbb{L}}$ *assigns a node label to every node, and*
- $port \in V^{\circledast}$ *is a sequence of nodes called* ports*.*

The type *of* G *is* $|port|$*. The set of all graphs (over an implicitly understood labelling alphabet) of type* τ *is denoted by* $\mathbb{G}_\tau$*. For an edge* $e = (u, \ell, v) \in E$*, we let* $src(e) = u$ *and* $tar(e) = v$*.*

An *isomorphism* (on G) is a bijective function $\mu \colon V \to V'$ for some set V' of nodes. It maps G to the graph $\mu(G) = (V', E', \mu \circ lab, \mu(port))$ where $E' = \{(\mu(u), \ell, \mu(v)) \mid (u, \ell, v) \in E\}$. If such an isomorphism exists, then the graphs G and $\mu(G)$ are said to be *isomorphic*.

The graph operations are of two kinds. The first is essentially disjoint union where the port sequences of the argument graphs are concatenated. Let $\tau, \tau' \in \mathbb{N}$. The operation $\uplus_{\tau\tau'} \colon \mathbb{G}_\tau \times \mathbb{G}_{\tau'} \to \mathbb{G}_{\tau+\tau'}$ is defined as follows: for $G \in \mathbb{G}_\tau$ and $G' \in \mathbb{G}_{\tau'}$, $\uplus_{\tau\tau'}(G, G')$ yields the graph in $\mathbb{G}_{\tau+\tau'}$ obtained by making the two graphs disjoint through a suitable renaming of nodes and taking their union. This is defined in the obvious way for the first three components and concatenates the port sequences of both graphs. Thus, the union operation $\uplus_{\tau\tau'}$ is not commutative and only defined up to isomorphism. We usually write $G \uplus_{\tau\tau'} G'$ instead of $\uplus_{\tau\tau'}(G, G')$. To avoid unnecessary technicalities, we shall generally assume that G and G' are disjoint from the start, and that no renaming of nodes takes place. We extend $\uplus_{\tau\tau'}$ to an operation $\uplus_{\tau\tau'} \colon \wp(\mathbb{G}_\tau) \times \wp(\mathbb{G}_{\tau'}) \to \wp(\mathbb{G}_{\tau+\tau'})$ in the usual way: for $\mathcal{G} \subseteq \mathbb{G}_\tau$ and $\mathcal{G}' \in \mathbb{G}_{\tau'}$, $\mathcal{G} \uplus_{\tau\tau'} \mathcal{G}' = \{G \uplus_{\tau\tau'} G' \mid G \in \mathcal{G}_\tau,\ G' \in \mathcal{G}'\}$.

To introduce the second type of graph operation, we first define an auxiliary nondeterministic *cloning operation*. The purpose of this operation is to make the expansion operations (to be defined afterwards) more flexible by allowing parts of it to be replicated an arbitrary number of times. This increases the flexibility with which it can attach to an existing structure. The operation was originally introduced to formalise the structure of object-oriented programs [17], and later adopted in computational linguistics [5].

To define the cloning operation, consider a graph $G = (V, E, lab, port)$ and let $C \subseteq V \setminus [port]$. Then $clone_C(G)$ is the set of all graphs obtained from G by replacing each of the nodes in C and their incident edges by an arbitrary number of copies (Figure 2 illustrates the construction). Formally, $G' = (V', E', lab', port') \in clone_C(G)$ if there is a family $(C_v)_{v \in V}$ of pairwise disjoint sets C_v of nodes, such that the following hold:

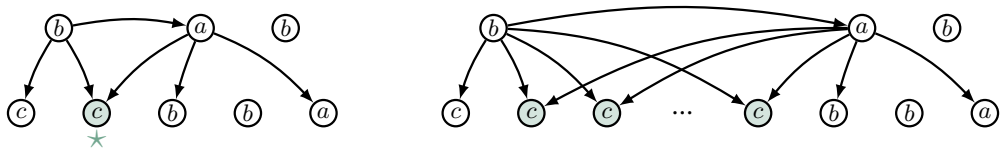


Fig. 2. The cloning operation takes a graph G and a subset C of its nodes, and replaces each of the nodes in C and their incidents edges by an arbitrary number of copies. The above figure shows G on the left-hand side, with C containing the single node marked with a star. On the right-hand side, the node and attaching structure has been replaced by a number of clones.

- $C_v = \{v\}$ for all $v \in V \setminus C$,
- $V' = \bigcup_{v \in V} C_v$,
- $E' = \bigcup_{(u,\ell,v) \in E} C_u \times \{\ell\} \times C_v$, and
- for all $v \in V$ and $v' \in C_v$, $lab'(v') = lab(v)$.

Note that cloning does *not* rename nodes in $V \setminus C$. In the following, we shall continue to denote by C_v ($v \in V$) the set of clones of v in a graph in $clone_C(G)$.

We can now define the second type of operation. It is a unary operation called *graph expansion operation* or simply *expansion operation*. We will actually use a restricted form, called *extension operation*, but define the more general expansion operation first. An expansion operation is described by a graph enriched by two additional components: let

$$\Phi = (V_\Phi, E_\Phi, lab_\Phi, port_\Phi, dock_\Phi, C_\Phi)$$

where both $\underline{\Phi} = (V_\Phi, E_\Phi, lab_\Phi, port_\Phi)$, henceforth called the *underlying graph* of Φ, and $(V_\Phi, E_\Phi, lab_\Phi, dock_\Phi)$ are graphs, and $C_\Phi \subseteq V_\Phi \setminus ([port_\Phi] \cup [dock_\Phi])$. We call $CONT_\Phi = V_\Phi \setminus ([port_\Phi] \cup [dock_\Phi])$ the set of *context nodes* and $NEW_\Phi = [port_\Phi] \setminus [dock_\Phi]$ the *new nodes*. The nodes in C_Φ are the *clonable nodes*. This terminology reflects that, according to the following definition, an application of Φ to a graph adds new (copies of the) nodes in NEW_Φ while those in $CONT_\Phi$ refer to nondeterministically chosen non-ports in the input graph. While every ordinary context node refers to exactly one node in the input graph, clonable context nodes refer to any number of nodes.

Throughout the rest of this paper, we shall continue to denote the components of an expansion operation Φ by V_Φ, E_Φ, lab_Φ, $port_\Phi$, $dock_\Phi$, and C_Φ.

Applying Φ to an argument graph $G = (V, E, lab, port) \in \mathbb{G}_\tau$ is possible if $\tau = |dock|$. It then yields a graph of type $\tau' = |port_\Phi|$ by fusing the nodes in $dock_\Phi$ with those in $port$. Moreover, all context nodes, that is, all nodes in the set $CONT_\Phi$, are fused with arbitrary (pairwise distinct) nodes in $V \setminus [port]$ carrying the same label. Prior to this nodes in C_Φ can be cloned an arbitrary number of times. Thus, the application of the operation to G clones the nodes in C, fuses those in $dock_\Phi$ with those in $port$, and fuses every node in C_v ($v \in CONT_\Phi$) injectively with any node in $V \setminus [port]$ carrying the same label. The port sequence of the resulting graph is $port_\Phi$.

Formally, let $\tau = |dock_\Phi|$ and $\tau' = |port_\Phi|$. Then Φ is interpreted as the nondeterministic operation $\Phi\colon \mathbb{G}_\tau \to \wp(\mathbb{G}_{\tau'})$ defined as follows. Given a graph $G = (V, E, lab, port) \in \mathbb{G}_\tau$, a graph $H \in \mathbb{G}_{\tau'}$ is in $\Phi(G)$ if it can be obtained by the following steps:

1) Choose a graph $G' \in clone_{C_\Phi}(\underline{\Phi})$ and an isomorphism μ on G' such that $\mu(G') = (V', E', lab', port')$ satisfies
   a) $\mu(NEW_\Phi) \cap V = \emptyset$,
   b) $\mu(dock_\Phi) = port$, and
   c) for all nodes $v \in CONT_\Phi$ and $v' \in \mu(C_v)$, it holds that $\mu(C_v) \subseteq V \setminus [port]$ and $lab_\Phi(v) = lab(v')$.
2) Define $H = (V \cup V', E \cup E', lab \sqcup lab', port')$.

Note that, by the definition of $\sqcup$, the labels of nodes in $dock_\Phi$ are disregarded, i.e., the nodes which are ports in G keep their labels as determined by lab.[3] Thus, in specifying an expansion operation Φ as above, the labels of nodes in $dock_\Phi$ can be dropped, essentially regarding these nodes as unlabelled ones.

In the following, we mainly view Φ as an operation $\Phi\colon \wp(\mathbb{G}_\tau) \to \wp(\mathbb{G}_{\tau'})$ defined as $\Phi(\mathcal{G}) = \bigcup_{G\in\mathcal{G}} \Phi(G)$ for all $\mathcal{G} \subseteq \mathbb{G}_\tau$. We call $\underline{\Phi} = (V_\Phi, E_\Phi, lab_\Phi, port_\Phi)$ the *underlying graph* of Φ, and may also denote Φ as $(G, dock, C)$ where $G = \underline{\Phi}$.

We now specialise expansion operations to extension operations by placing conditions on their structure. The intuition is to make sure that graphs are built bottom-up, i.e., Φ always extends the input graph by placing nodes and edges "on top", and in such a way that all nodes of the argument graph are reachable from the ports. For this, we require that only new ports – ports that do not correspond to ports of the input graph – have outgoing edges, and that all ports of the input graph that are not ports of the resulting graph carry incoming edges. Noting that the former set is NEW_Φ and the latter is $[dock_\Phi] \setminus [port_\Phi]$, this results in the following pair of formal requirements:

(R1) $\{src(e) \mid e \in E_\Phi\} \subseteq NEW_\Phi$, and
(R2) $[dock_\Phi] \setminus [port_\Phi] \subseteq \{tar(e) \mid e \in E_\Phi\}$.

Example 4. *Figure 3 depicts an extension operation together with a graph to which it can be applied. In Figure 4, we see three different graphs, all resulting from the application of the extension operation in Figure 3 to the graph in Figure 4.*

A *graph extension algebra* is a Σ-algebra $\mathcal{A} = ((\wp(\mathbb{G}_\tau))_{\tau\in\mathcal{T}}, (f_\mathcal{A})_{f\in\Sigma})$ where $\mathcal{T}$ is a finite subset of $\mathbb{N}$ and every symbol in Σ is interpreted as an extension or union operation or the set $\{\phi\}$, where ϕ is the empty graph $(\emptyset, \emptyset, \emptyset, \varepsilon)$. Note that the operations of the algebra act on sets of graphs rather than on single graphs. This is necessary because of the nondeterministic nature of extension operations.

Definition 5 (Graph Extension Grammar)**.** *A (tree-based)* graph extension grammar *is a pair* $\Gamma = (g, \mathcal{A})$ *where* $\mathcal{A}$ *is a graph extension* Σ*-algebra for some signature* Σ *and* g *is a regular*

[3] This idea has its origins in personal discussions between the second author and Berthold Hoffmann in 2019.

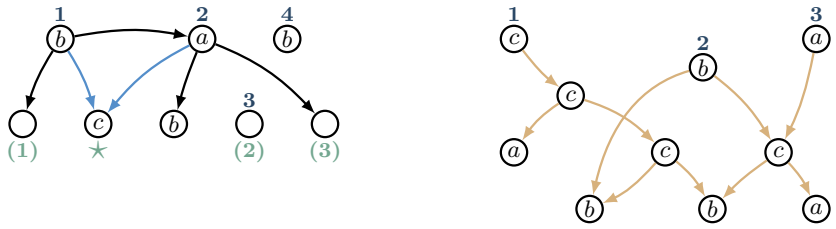


Fig. 3. The figure on the left shows an extension operation Φ with three ports (indicated with numbers above the nodes), three docks (indicated with numbers below the nodes), and one clonable node (indicated with a star). The figure on the right shows a graph G with three ports (again, indicated with numbers above the nodes). The operation Φ is applicable to G because (i) the number of ports of G coincides with the number of docks of Φ, so the types are compatible, and (ii) the pair of nodes in $CONT_\Phi$ can be identified with nodes in G outside of the ports of G (in several ways).

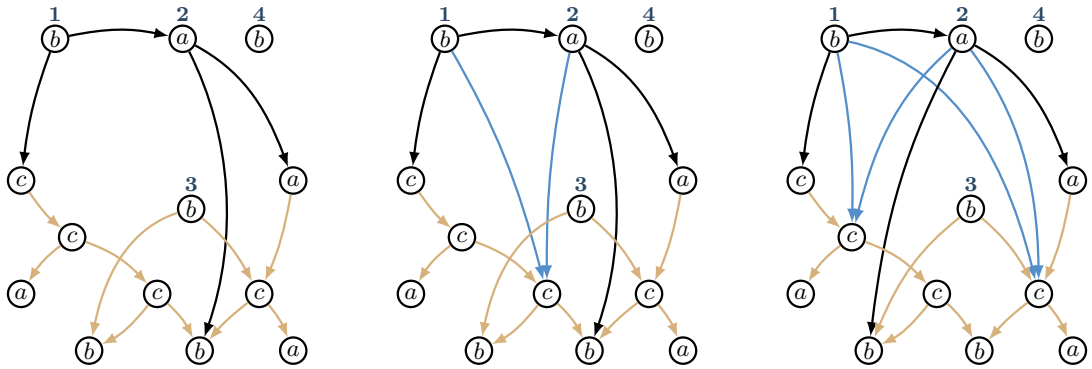


Fig. 4. Three graphs in $\Phi(G)$ where the extension operation Φ and the graph G are as in Figure 3. The differences between the graphs reflect the number of times the nodes in C_Φ have been replicated, and the choice of mapping from the nodes in $CONT_\Phi$ to nodes in G. The single node in C_Φ has been replicated 0 times to obtain the first graph, once to obtain the second, and twice to obtain the third. In the first two graphs, the non-dock node labelled b has been identified with one b-labelled node in G, and in the right graph with another.

tree grammar over Σ. *The graph language generated by* Γ, *denoted by* $L(\Gamma)$, *is defined as*

$$L(\Gamma) = \bigcup_{t \in L(g)} val_{\mathcal{A}}(t).$$

To simplify notation, we shall in the following assume that, in a graph extension grammar as above, $f = f_{\mathcal{A}}$ for all $f \in \Sigma$, i.e., we use the operations themselves as symbols in Σ.

Before continuing on to parsing, let us pause to consider at a concrete example of graph extension grammar in the setting of natural language processing.

Example 6. The graph extension grammar in Figure 5 illustrates how the new formalism can model semantic representations. In this example, the concepts `girl` and `boy` represent entities

that can act as agents (which verbs cannot), `try` and `persuade` represent verbs that require structural control, and `want` and `believe` represent verbs where this is not needed.

First, we take a top-down perspective to understand the tree generation. The initial nonterminal is S. The base case for S is the generation of an extension operation that creates a single node representing a `girl` or a `boy` concept. In all other cases, S generates an extension operation that adds a verb and its outgoing edges, and in which the verb is the single port. The nonterminal C has the same function as S with the following two differences: it has no corresponding base case, and the ports of the extension operations mark both the agent of the verb (designated by an `arg0` edge) and the verb itself. The nonterminal S' can only generate extensions that create `girl` or `boy` nodes. As we shall see, this allows us to restrict the addition of `arg0` to persons. Finally, there is a pair of nonterminals U and U' that both create union operations, the former with two resulting ports, and the latter with three.

Now, we take a bottom-up perspective to see how the extension operations of a tree are evaluated. Unless the tree consists of a single node, we will have several extensions generated by S and S' that create `girl` and `boy` nodes as the leaves of the tree. In this case, we can apply one or more union operations to concatenate their port sequences and make them visible to further operations. The construction ensures that an `arg0` edge only can have a person, i.e., a valid agent, as its target. After the application of a union operation, it is possible to apply any extension that is applicable to U (or U'), meaning that none of the non-port nodes are contextual nodes. The resulting graph can have one or two ports – if the graph has one port, the applied extension operation was generated by S, and if the graph has two ports, the applied operation was generated by C. In other words: C signals that the graph is ready for the addition of a control verb, and S that the graph is a valid semantic graph with one port. When sufficiently many nodes have been generated, it is no longer necessary (but still possible) to use the union operations. Instead, we can add incoming edges to already generated nodes contextually (unless control is explicitly needed, as for control verbs). The generated graphs can thus contain both structural and non-structural dependencies. See Figure 6 for an example of a tree generated by the grammar in Figure 5, and its evaluation into a semantic graph. $\diamond$

Readers familiar with HRGs or Courcelle's hyperedge-replacement algebras [12] will have noticed that, when disregarding context nodes, productions using union and extension operations correspond to hyperedge replacement productions of two types:

(a) $A \to B \uplus_{\tau\tau'} C$ corresponds to a production with two hyperedges in the right-hand side, labelled B and C, where the first one is attached to τ of the nodes to which the nonterminal in the left-hand side (which is labelled A) is attached, and the second is attached to the remaining τ' nodes of the left-hand side. In particular, there are no further nodes in the right-hand side.

(b) $A \to \Phi[B]$ corresponds to a hyperedge-replacement production with a single nonterminal hyperedge labelled B in its right-hand side. Such productions are the ones responsible for actually generating new terminal items (nodes and edges).

Generalised extension operations of arity greater than 1 can be constructed by defining so-called derived operations through composition of suitable extension and union operations. Given a set

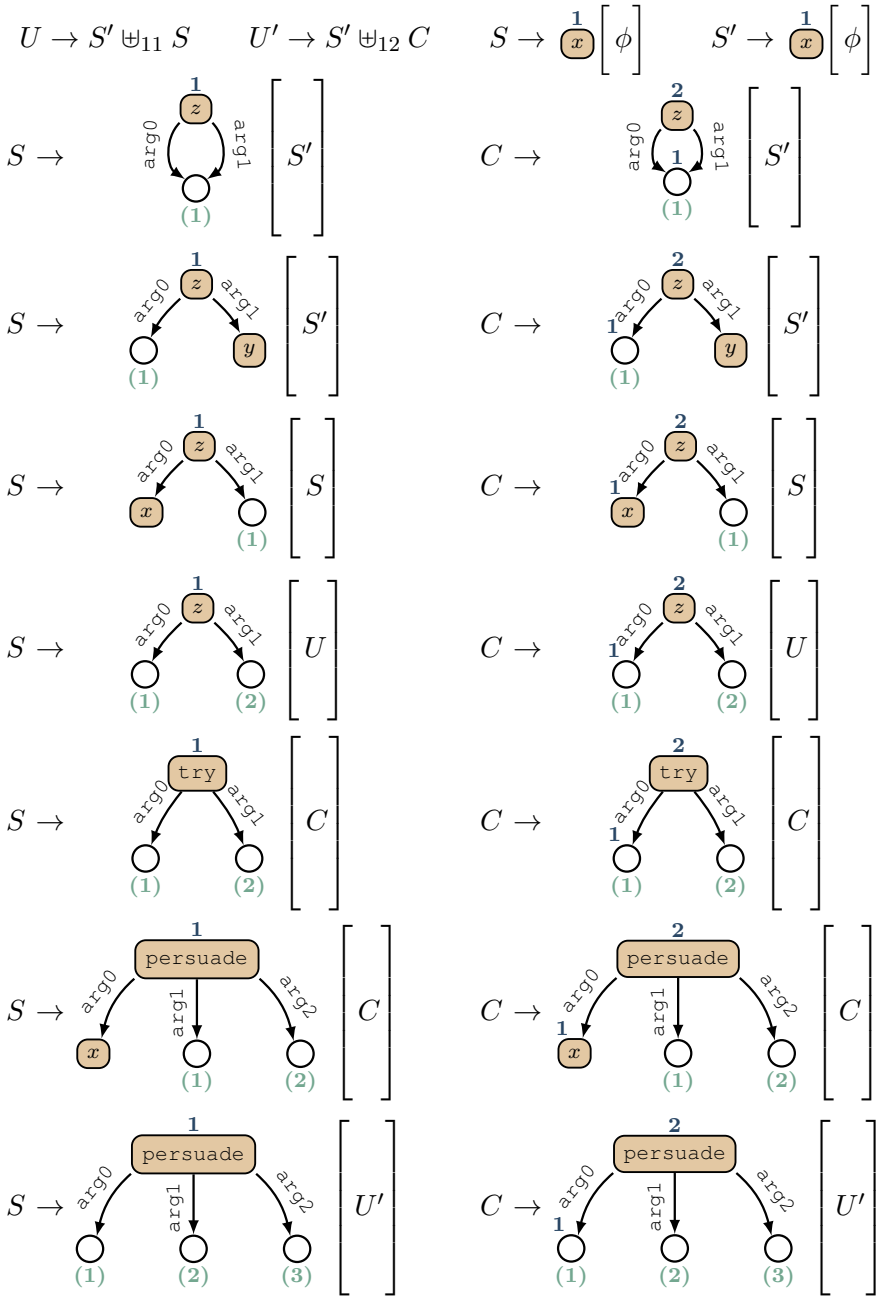


Fig. 5. A (graphical representation of a) graph extension grammar to demonstrate how our formalism can be used to generate semantic graphs akin to AMR. Instead of repeating productions that only differ with respect to node labels, we use variables representing labels to achieve a more compact representation, where $x \in \{\texttt{girl}, \texttt{boy}\}$, $y \in \{\texttt{girl}, \texttt{boy}, \texttt{want}, \texttt{believe}, \texttt{try}, \texttt{persuade}\}$, and $z \in \{\texttt{want}, \texttt{believe}\}$. The nonterminal C generates graphs of type 2 whose ports are to be embedded in a control verb construction.

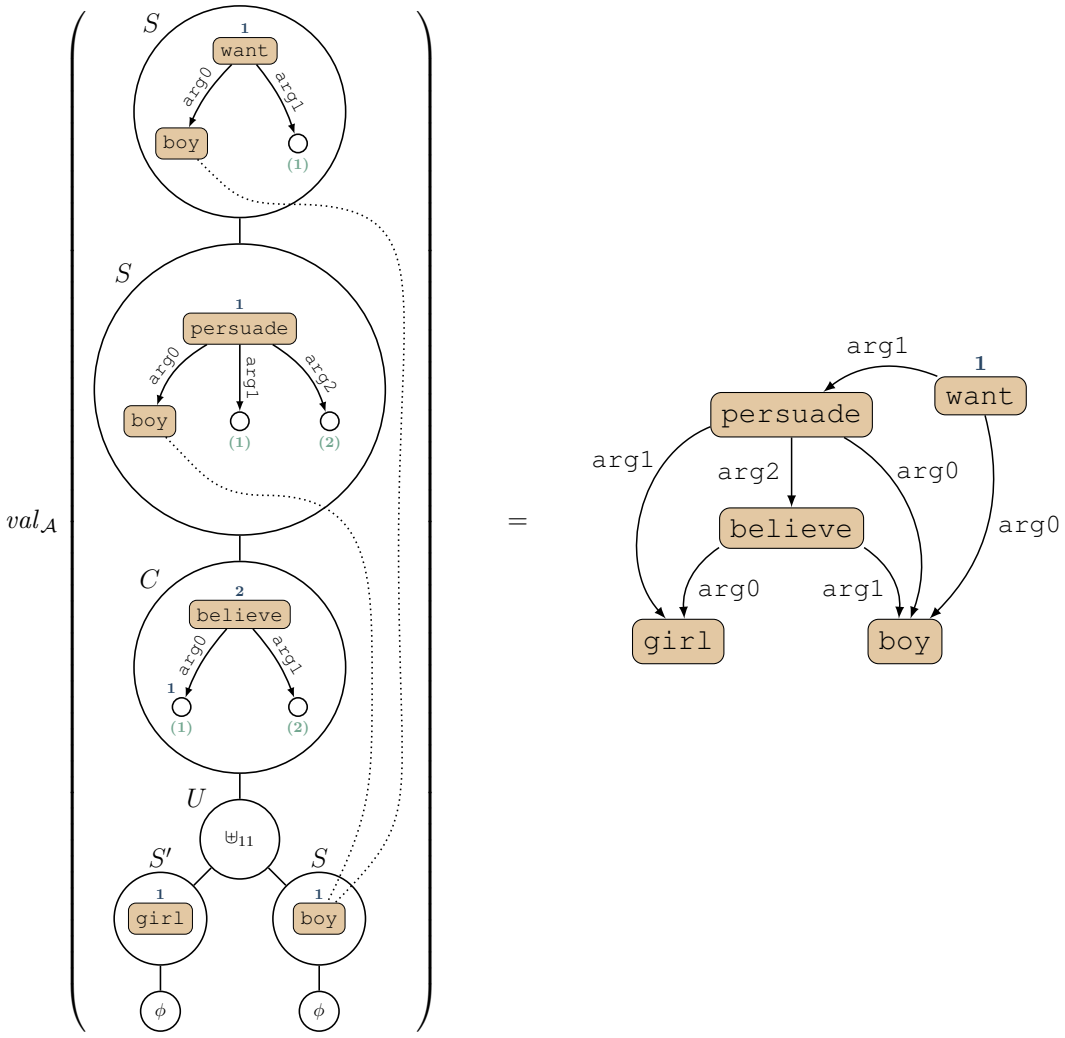


Fig. 6. A (graphical representation of a) tree generated by the regular tree grammar in Figure 5 and the graph resulting from its evaluation. The nonterminal generating each subtree is shown next to the respective circle. The identification of context nodes with previously created nodes is indicated by dotted lines. In this example, there is only one possible candidate for a context node, hence the resulting graph is unique up to isomorphism.

of such derived operations, the tree-to-tree mapping that replaces each occurrence of a derived operation by its definition is a tree homomorphism. As regular tree languages are closed under tree homomorphisms, this shows that the restriction to binary unions and unary extensions is no limitation (in contrast to requirements (R1) and (R2), which ensure polynomial parsability).

III. PARSING

We now provide a basic polynomial parsing algorithm for graph extension grammars. For this, and throughout this section, let $\Gamma = (g, \mathcal{A})$ be a graph extension grammar, where $g = (N, \Sigma, P, S)$ and $\mathcal{A}$ is a graph extension Σ-algebra. The goal is parsing, i.e., to decide the membership problem for Γ, and in the positive case, produce a witness derivation in Γ. The membership problem is stated as follows:

Input: A graph G
Question: Does it hold that $G \in L(\Gamma)$?

By definition, extension operations keep the identities of nodes in the input graph unchanged, whereas new nodes added to the graph may be given arbitrary identities (as long as they are not in the input graph). For the union operation, it holds that renaming of nodes is necessary only if the node sets of the two argument graphs intersect. As a consequence, when evaluating a tree t, we can without loss of generality assume that operations never change node identities of argument graphs. In other words, if $G \in val_{\mathcal{A}}(t)$, then there is a (concrete) graph $G_\alpha \in val_{\mathcal{A}}(t/\alpha)$ such that $G_\varepsilon = G$ and, for every $\alpha \in addr(t)$,

- if $t/\alpha = \Phi[t/\alpha 1]$, then $G_\alpha \in \Phi(G_{\alpha 1})$ and
- if $t/\alpha = t/\alpha 1 \uplus_{\tau\tau'} t/\alpha 2$, then $G_\alpha = (V \cup V', E \cup E', lab \sqcup lab', port\ port')$ where $G_{\alpha i} = (V_i, E_i, lab_i, port_i)$ for $i \in [2]$.

As a consequence, for every $\alpha \in addr(t)$, the graph G_α is a subgraph of G. More precisely, if $G = (V, E, lab, port)$ and $G_\alpha = (V', E', lab', port')$ then $V' \subseteq V$, $E' \subseteq E$, and $lab' = lab|_{V'}$. Note that, while the graphs G_α are usually not uniquely determined by G and t, the important fact is that they do exist if and only if $G \in val_{\mathcal{A}}(t)$. We say in the following that the family $(G_\alpha)_{\alpha \in addr(t)}$ is a *concrete evaluation of t into G*. Hence, the membership problem amounts to decide, for the input graph G, whether there is a tree $t \in L(g)$ that permits a concrete evaluation into G.

The following lemma, which forms the basis of our parsing algorithm, shows that G_α is determined by $port'$ alone, as it is the subgraph of G induced by the nodes that are reachable in G from $port'$ in G. More precisely, given a sequence $p \in V^{\circledast}$ of nodes of G, let $G \triangledown p = (V', E|_{V' \times \mathbb{L} \times V'}, lab|_{V'}, p)$, where V' is the set of nodes reachable in G by directed paths from (any of the nodes in) $port$.

Lemma 7. *Let $(G_\alpha)_{\alpha \in addr(t)}$ be a concrete evaluation of a tree $t \in \mathrm{T}_\Sigma$ into a graph G, and let $G_\alpha = (V_\alpha, E_\alpha, lab_\alpha, port_\alpha)$ for every $\alpha \in addr(t)$. Then $G_\alpha = G \triangledown port_\alpha$ for all $\alpha \in addr(t)$.*

Proof. Using the notations from the statement of the lemma, and furthermore denoting the set of all nodes that can be reached from $p \in V^{\circledast}$ in G by $p^{\triangledown}$, we first show the following claim:

Claim 1. For all $e \in E$ and $v \in V$, $src(e) \in V_\alpha$ implies $e \in E_\alpha$, and $v \in port_\alpha^{\triangledown}$ implies $v \in V_\alpha$.

We prove Claim 1 by induction on the length of α. For $G_\alpha = \varepsilon$, the statement holds trivially. Now, assume that it holds for some $\alpha \in addr(t)$. Since the empty graph ϕ has neither ports nor edges, two relevant cases remain.

Case 1. G_α is of the form $G_{\alpha 1} \uplus_{\tau\tau'} G_{\alpha 2}$.

Let $\{i, j\} = [2]$. We have to show that the statement holds for $G_{\alpha i}$. Thus, assume that $src(e) \in V_{\alpha i}$ and $v \in port^{\triangledown}_{\alpha i}$. Then $src(e) \in V_\alpha$ and thus, by the induction hypothesis, $e \in E_\alpha$. By the definition of $\uplus_{\tau\tau'}$ this means that $e \in E_{\alpha i}$. Furthermore, $v \in port^{\triangledown}_{\alpha i}$ implies $v \in port^{\triangledown}_{\alpha}$ and thus, again by the induction hypothesis, $v \in V_\alpha$. Hence, it remains to argue that no node in $G_{\alpha j}$ can be reached from $port_{\alpha i}$. This follows readily from the just established fact that there is no edge $e' \in V \setminus V_{\alpha i}$ for which $src(e') \in E_{\alpha i}$ (together with the fact that $V_{\alpha i} \cap V_{\alpha j} = \emptyset$).

Case 2. G_α is of the form $\Phi(G_{\alpha 1})$ for an extension operation Φ.

To show that the statement holds for $G_{\alpha 1}$, let $G_\alpha = (V_{\alpha 1} \cup V', E_{\alpha 1} \cup E', lab_{\alpha 1} \sqcup lab', port')$, where $(V', E', lab', port')$ is obtained from $\underline{\Phi}$ as in the definition of $\Phi(G_{\alpha 1})$. By the induction hypothesis, $src(e) \in V_\alpha$ implies $e \in E_\alpha$. Thus, $src(e) \in V_{\alpha 1}$ implies $e \in E_{\alpha 1}$ unless $e \in E'$. However, by requirement (R1), all $e \in E'$ satisfy $src(e) \in [port'] \setminus [port]$, which is equivalent to $src(e) \notin V_{\alpha 1}$ because $V_\alpha \setminus V_{\alpha 1} = [port'] \setminus [port]$. This shows that $e \in E_{\alpha 1}$.

Now, let $v \in port^{\triangledown}_{\alpha 1}$. If $v \notin V_{\alpha 1}$, consider the first edge e on a path from a node in $[port_{\alpha 1}]$ to v such that $tar(e) \notin V_{\alpha 1}$. Then $src(e) \in V_{\alpha 1}$ but $e \notin E_{\alpha 1}$, contradicting the previously established fact that $src(e) \in V_{\alpha 1}$ implies $e \notin E_{\alpha 1}$. This finishes the proof of Claim 1.

The other direction of the second part of Claim 1 remains to be shown:

Claim 2. $V_\alpha \subseteq port^{\triangledown}_{\alpha}$.

This time, we proceed by induction on the size of t/α. The statement is trivially true for $G_\alpha = \phi$. Thus, as before, there are two cases to distinguish.

Case 1. G_α is of the form $G_{\alpha 1} \uplus_{\tau\tau'} G_{\alpha 2}$.

By the induction hypothesis, Claim 2 holds for $G_{\alpha 1}$ and $G_{\alpha 2}$. Consequently, $V_\alpha = V_{\alpha 1} \cup V_{\alpha 2} \subseteq port^{\triangledown}_{\alpha 1} \cup port^{\triangledown}_{\alpha 2} = port^{\triangledown}_{\alpha}$.

Case 2. G_α is of the form $\Phi(G_{\alpha 1})$ for an extension operation Φ.

Again, let $G_\alpha = (V_{\alpha 1} \cup V', E_{\alpha 1} \cup E', lab_{\alpha 1} \sqcup lab', port')$, where $(V', E', lab', port')$ is obtained from $\underline{\Phi}$ as in the definition of $\Phi(G_{\alpha 1})$. (In particular, $port' = port_\alpha$.) By the induction hypothesis, $V_{\alpha 1} \subseteq port^{\triangledown}_{\alpha 1}$. Moreover, by requirement (R2), for every node $v \in [port_{\alpha 1}]$ it either holds that $v \in [port_\alpha]$, or there are $u \in [port_\alpha]$ and $e \in E'$ with $src(e) = u$ and $tar(e) = v$. Hence, $[port_{\alpha 1}] \subseteq port^{\triangledown}_{\alpha}$ and thus $V_{\alpha 1} \subseteq port^{\triangledown}_{\alpha 1} \subseteq port^{\triangledown}_{\alpha}$. Since, furthermore, $V' \setminus V_{\alpha 1} \subseteq [port_\alpha]$, this shows that $V_\alpha \subseteq [port_\alpha]$, and thus $V_\alpha \subseteq port^{\triangledown}_{\alpha}$, as claimed. □

We note the following immediate consequence of Lemma 7:

Corollary 8. *Let $(G_\alpha)_{\alpha \in addr(t)}$ be a concrete evaluation of a tree $t \in \mathrm{T}_\Sigma$ into a graph G. If $G_\alpha = G \triangledown \varepsilon$ for some $\alpha \in addr(t)$, then $G_\alpha = \phi$.*

It follows from Corollary 8 that a nonterminal A of type ε can only derive trees that evaluate to the empty graph ϕ. We can therefore replace all rules where A appears on the left-hand side

by a single rule that takes A to the empty graph, without changing the generated graph language. The construction suggests the following normal form for graph extension grammars.

Lemma 9 (normal form)**.** Γ *can be transformed into a graph extension grammar* $\Gamma' = (g', \mathcal{A}')$ *with* $L(\Gamma') = L(\Gamma)$ *such that the co-domain* s *of every extension and union operation in* $\mathcal{A}'$ *satisfies* $|s| \geq 1$.

Proof. We may assume that $L_A(g) \neq \emptyset$ for all $A \in N$. By Corollary 8, $\sigma(A) = \varepsilon$ then implies that $val_{\mathcal{A}}(L_A(g)) = \{\phi\}$. It follows that all rules with the left-hand side A can be replaced by a single rule $A \to \phi$. □

From here on, let us assume that Γ is in the normal form given by Lemma 9. Turning our attention back on Lemma 7, we find the beginnings of a recursive parsing algorithm: In the following, consider an input graph $G = (V, E, lab, port)$, a nonterminal $A \in N$, and a sequence $p \in V^{\circledast}$ with $\dot{\mathbb{L}}(p) = \sigma(A)$. To decide whether $G \triangledown p \in val_{\mathcal{A}}(L_A(g))$ (and thus whether $G \in L(\Gamma)$ if $A = S$ and $p = port_G$), there are two cases. If $p = \varepsilon$, then $G \triangledown p = \phi \in val_{\mathcal{A}}(L_A(g))$ if and only if the production $A \to \phi$ is in P. Otherwise, we need to check each production with the left-hand side A, as follows:

Case 1. If the production is of the form $A \to B_1 \uplus_{\tau_1 \tau_2} B_2$, then we recursively need to determine whether $G \triangledown p_i \in val_{\mathcal{A}}(L_{B_i}(g))$ for $i = 1, 2$, where $p = p_1 p_2$ and $|p_1| = \tau_1$.

Case 2. If the production is of the form $A \to \Phi(B)$, the basic intuition is that we need to check all possible mappings of the edges in E_Φ to edges in E that have their sources in p. However, since nodes in C_Φ can be cloned, and since some context nodes of Φ may be isolated, the technicalities are slightly more involved.

To simplify the reasoning, let us first assume that $\underline{\Phi}$ does not contain isolated context nodes, i.e., that it satisfies the following stronger variant of (R2):

(R3) $V_\Phi \setminus [port_\Phi] \subseteq \{tar(e) \mid e \in E_\Phi\}$.

Let $port_\Phi = u_1 \cdots u_k$, $p = v_1 \cdots v_k$, and define $NEW = \{v_i \mid i \in [k] \text{ and } u_i \notin [dock_\Phi]\}$. In other words, NEW contains the nodes in p which correspond to those in NEW_Φ, which are the only nodes that may have edges to other nodes in Φ. Now, let $E_P = \{e \in E \mid src(e) \in NEW\}$ and $V_p = [p] \cup \{tar(e) \mid e \in E_p\}$. Note that, by (R3), V_p contains all the nodes of V that must be put into correspondence with nodes in V_Φ, while the nodes in in $V \setminus V_p$ are to be left for future, recursive, invocations of the parsing algorithm. We say that a surjective mapping $m \colon V_p \to V_\Phi$ is a *matching of* Φ *to* $G \triangledown p$ if it:

(M1) preserves node labels, that is, $lab_\Phi(m(v)) = lab(v)$ for all $v \in V$ with $m(v) \notin [dock_\Phi]$,
(M2) clones only clonable nodes, meaning that $|m^{-1}(u)| = 1$ for all $u \in V_\Phi \setminus C_\Phi$, and
(M3) bidirectionally preserves edges, that is, $(v, \ell, v') \in E_p \iff (m(v), \ell, m(v')) \in E_\Phi$ for all $v, v' \in V_p$ and all $\ell \in \dot{\mathbb{L}}$.

Now, by Lemma 7, the parsing algorithm needs to check recursively whether there is a matching of Φ to $G \triangledown p$ such that $G \triangledown m^{-1}(dock_\Phi) \in val_{\mathcal{A}}(L_B(g))$.[4]

[4]Recall that $[dock_\Phi] \cap C_\Phi = \emptyset$, and thus $m^{-1}(dock_\Phi) \in V^{|dock_\Phi|}$.

Algorithm 1 Parse a graph G with respect to a graph extension grammar $\Gamma = (g, \mathcal{A})$

procedure PARSE($G = (V, E, lab, port)$)
  return PARSE_REC($S, port$) ▷ Invoke recursive procedure with initial nonterminal S
end procedure
procedure PARSE_REC($A \in N$, $p \in V^{\circledast}$)
  if $\sigma(A) = \varepsilon$ **then** ▷ Leverage normal form
    return $(A \to \phi) \in P$
  else if $result[A, p]$ defined **then** ▷ Use memoïsed result
    return $result[A, p]$
  end if
  memoïse $result[A, p] \leftarrow false$ ▷ No success so far
  for all $(A \to B_1 \uplus_{\tau_1 \tau_2} B_2) \in P$ **do**
    let $p = p_1 p_2$ where $|p_1| = \tau_1$
    if PARSE_REC(B_1, p_1) and PARSE_REC(B_2, p_2) **then**
      memoïse $result[A, p] \leftarrow true$ ▷ Memoïse success
      return $true$
    end if
  end for
  for all $(A \to \Phi[B]) \in P$ **do**
    for all matchings m of Φ to $G \triangledown p$ **do**
      if PARSE_REC($B, m^{-1}(dock_\Phi)$) **then**
        memoïse $result[A, p] \leftarrow true$ ▷ Memoïse success
        return $true$
      end if
    end for
  end for
  return $false$
end procedure

Finally, to avoid the stronger assumption (R3), we have to handle any isolated context nodes $u \in V_\Phi$. Since, by injectivity, such a node u must be mapped to any node v in $G \triangledown p$ with the same label, except those in V_p. Thus, we only need to check the existence of such nodes outside V_p. Note that clonable isolated context nodes u can (and must) be disregarded because they can be cloned zero times, thus having no effect. In summary, let

$$\#_\Phi(\ell) = |\{u \in CONT_\Phi \setminus C_\Phi \mid lab_\Phi(u) = \ell\}|$$

for all $\ell \in \dot{\mathbb{L}}$. Then a matching m must, in addition, satisfy the following condition:

(M4) For all $\ell \in \dot{\mathbb{L}}$, $G \triangledown p$ contains at least $\#_\Phi(\ell)$ nodes v with $lab(v) = \ell$ that are not in V_p.

Note that this condition is independent of m.

The pseudocode of the algorithm is shown in Algorithm 1. As we shall see, the algorithm runs in polynomial time in the size of the input graph. However, the maximum length of the type sequences used appears in the exponent, so the degree of the polynomial depends on a property of the grammar. In the following theorem, we let $|G| = |V| + |E|$ for a graph $G = (V, E, lab, port)$.

Theorem 10. *Let Σ be $\mathcal{T}$-typed and $c = \max\{\tau \mid \tau \in \mathcal{T}\}$. If applied to an input graph G, Algorithm 1 terminates in $O(|G|^{2c+1})$ steps and returns $true$ if and only $G \in L(\Gamma)$.*

Proof. Let $n = |G|$. Correctness should be clear from Lemma 7, the assumption that Γ is in normal form (cf. Lemma 9), and the preceding discussion. Further, by the choice of c, there are at most $O(n^c)$ distinct parameters PARSE_REC can be called with. Since we consider a fixed graph extension grammar Γ, this is a polynomial. For the same reason, the two outer **for** loops of the algorithm are executed a constant number of times during each call. It remains to consider the loop starting on line 19. Undoubtedly, the naïve implementation that enumerates all matchings is exponential. However, it is clear from the loop body that we only really need to determine all sequences $d_1 \cdots d_k = m^{-1}(\delta_1 \cdots \delta_k)$, where $\delta_1 \cdots \delta_k = dock_\Phi$ and m is a matching of Φ to $G \triangledown p$. (Recall that m^{-1}, restricted to $[dock_\Phi]$, is a function by (M2).) Since there are no more than $O(n^c)$ such sequences it remains to be shown that, for each of them, it is possible to check in linear time whether it can be extended to a matching. It bears repeating that we, in the paragraphs that follow, are only looking to establish the existence of such an extension, not compute a full set of complete matchings, as that would be too costly.

Clearly, condition (M4), which is independent of the choice of the sequence $d_1 \dots d_k$, can be checked in linear time. Thus, it remains to be shown how to verify that the mapping $d_i \mapsto \delta_i$ can be extended to a matching under the assumption that (M4) is satisfied.

For this, let us use the notation of Case 2 in the discussion preceding the theorem (i.e., the definition of matchings following (R3)). In particular, $E_P = \{e \in E \mid src(e) \in NEW\}$ and $V_p = [p] \cup \{tar(e) \mid e \in E_p\}$. Consider a candidate sequence $d_1 \cdots d_k \in V_p^k$. We have to check whether there is a matching m such that $m(d_1 \cdots d_k) = \delta_1 \cdots \delta_k$. For this, let

$$lab'(u) = \begin{cases} lab_\Phi(u) & \text{if } u \in V_\Phi \setminus [dock_\Phi] \\ lab(d_i) & \text{if } u = \delta_i \text{ for some } i \in [k]. \end{cases}$$

The idea is now to compute a characterisation, or *profile*, of the nodes in V_Φ and V_p that allows us to quickly decide whether such a matching m exists. In short, we record for each node u a set of triples, where each triple describes an incoming edge to u that must be covered by the matching. For every node $u \in V_\Phi$, let $profile_\Phi(u) = \{(i, \ell, lab'(u)) \mid (u_i, \ell, u) \in E_\Phi\}$. Similarly, for $v \in V_p$, let $profile_G(v) = \{(i, \ell, lab(v)) \mid (v_i, \ell, v) \in E\}$. Figure 7 illustrates the construction with an example.

We now combine the profiles for the nodes in V_Φ into a single, composite, profile for all of Φ, giving us a set of sets of edge descriptions. For each such edge description, we count how many of its type we need to find. This can be an exact number, or a lower bound, depending on whether some of the target nodes are clonable and can be left out or replicated as desired. Thus, define $\Pi_\Phi = \{profile_\Phi(u) \mid u \in V_\Phi \setminus [port_\Phi]\}$ and let, for all $\pi \in \Pi_\Phi$,

$$mult(\pi) = \begin{cases} eq_m & \nexists\, u \in C_\Phi \colon profile_\Phi(u) = \pi \\ gt_m & \text{otherwise} \end{cases}$$

where $m = |\{u \in V_\Phi \setminus ([port_\Phi] \cup C_\Phi) \mid profile_\Phi(u) = \pi\}|$. Then, by the definition of matchings, there exists a matching $m \colon V_p \to V_\Phi$ such that $m(d_1 \cdots d_k) = \delta_1 \cdots \delta_k$ if and only if

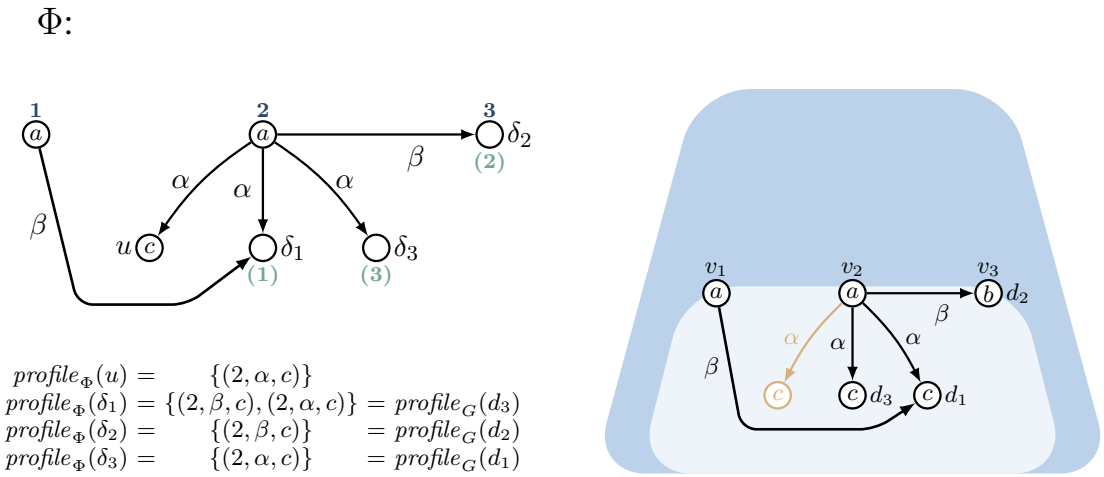


Fig. 7. On the top-left is a schematic view of an extension operation Φ with three ports and equally many docks. Deciding whether there is a matching m such that $m(d_i) = \delta_i$ for $i \in [3]$ boils down to computing the functions $profile_\Phi$ and $profile_G$ for all nodes in V_Φ and V_p, respectively. (Note that $profile_\Phi(\delta_i)$ depends on d_i.) Based on the result, we find that $G \triangledown p$ must contain the part drawn in brown for $d_1 d_2 d_3$ to extend to a matching, but only the existence of such an edge and node is relevant.

- the subgraph of G induced by $[p]$ is isomorphic to the subgraph of $\underline{\Phi}$ induced by $[port_\Phi]$,
- for all $i \in [k]$, $profile_\Phi(\delta_i) = profile_G(d_i)$,[5] and
- for all $\pi \in \Pi_\Phi$,

$$|\{v \in V_p \mid profile(v) = \pi\}| = k \quad \text{if } mult(\pi) = eq_k \text{ for some } k \in \mathbb{N}$$
$$|\{v \in V_p \mid profile(v) = \pi\}| \geq k \quad \text{if } mult(\pi) = gt_k \text{ for some } k \in \mathbb{N} \ .$$

It should be clear that these conditions can be checked in linear time. □

While Algorithm 1 only returns *true* or *false*, i.e., solves the pure membership problem, we can instead solve the parsing problem by changing the memoïsation to store for every nonterminal A and every p a tree that gives rise to the generated subgraph $G \triangledown p$ (or all of those trees).

We note here that the theoretical upper bound $O(|G|^{2c+1})$ proved above is unlikely to be reached in the average practical case, because an efficient implementation would only test candidate sequences $d_1 \cdots d_k$ such that $profile_\Phi(\delta_i) = profile_G(d_i)$ for all $i \in [k]$. In fact, there are many useful special cases for which better bounds can be proved. Two examples are mentioned in the next theorem, in which we use the following notation similar to the profiles used in the proof above, but restricted to the first and second components: For every node $u \in V_\Phi$, we let $profile'_\Phi(u) = \{(i, \ell) \mid (u_i, \ell, u) \in E_\Phi\}$ and $\Pi'_\Phi = \{profile'_\Phi(u) \mid u \in V_\Phi \setminus [port_\Phi]\}$.

Theorem 11. *Let Σ be $\mathcal{T}$-typed and $c = \max\{\tau \mid \tau \in \mathcal{T}\}$.*

[5] In fact, because of the previous item this only needs to be checked for $\delta_i \notin [port_\Phi]$.

1) Assume that, for every extension operation Φ of $\mathcal{A}$ and all $u \in [dock_\Phi] \setminus [port_\Phi]$ and $u' \in C_\Phi$, it holds that $profile'_\Phi(u) \neq profile'_\Phi(u')$. Then Algorithm 1 runs in time $O(|G|^{c+1})$.

2) Assume that the following hold:

a) For every extension operation Φ of $\mathcal{A}$ and all distinct $u \in [dock_\Phi] \setminus [port_\Phi]$ and $u' \in V_\Phi \setminus [port_\Phi]$, we have $profile'_\Phi(u) \neq profile'_\Phi(u')$.

b) For all distinct productions of the form $A \to \Phi[B]$ and $A \to \Phi'[C]$, where Φ and $\Phi' = (V'_\Phi, E'_\Phi, lab'_\Phi, dock'_\Phi, C'_\Phi)$, it holds that

$$\{profile'_\Phi(u) \mid u \in V_\Phi \setminus C_\Phi\} \not\subseteq \Pi'_{\Phi'} \quad or \quad \{profile'_{\Phi'}(u) \mid u \in V_{\Phi'} \setminus C_{\Phi'}\} \not\subseteq \Pi'_\Phi \ .$$

c) For all $A \in N$, if there is a production of the form $A \to B \uplus_{\tau\tau'} C$ in P, then there is no other production with the left-hand side A.

Then Algorithm 1 runs uniformly in linear time.

Proof. The assumption of the first statement implies that only a constant number of sequences $d_1 \cdots d_k$ in the proof of Theorem 10 can fulfil the requirement $profile_\Phi(\delta_i) = profile_G(d_i)$ for all $i \in [k]$. Hence, the **for** loop starting on line 19 can be implemented to run in linear time, which proves the statement.

For the second statement, note that the first assumption, which strengthens the assumption of the first part of the theorem, implies that at most one candidate sequence $d_1 \cdots d_k$ can fulfil the requirement that $profile_\Phi(\delta_i) = profile_G(d_i)$ for all $i \in [k]$. Moreover, by the second assumption, if there is such a sequence for a production $A \to \Phi[B]$ considered in the loop starting on line 18, then all other productions to be considered in that loop lack such a sequence. Finally, the third assumption makes sure that at most one production is to be considered on line 11, and if there is one, then there is no production of the form required on line 18.

It follows that the algorithm will decompose G in a deterministic top-down process, each step taking constant time. This proves the assertion. □

IV. Conclusion

We have introduced a graph grammar formalism that is both a restriction and an extension of hyperedge-replacement graph grammars, namely graph extension grammars. These grammars are formalised in a tree-based fashion, consisting of an algebra over graphs and a regular tree grammar that generates expressions over the operations of the algebra. The graphs are directed graphs with ports, constructed using operations of two kinds: The first is disjoint union, and the second is a family of unary operations that add new nodes and edges to an existing graph G. The augmentation is done in such a way that all new edges lead from a new node to either

(i) a port in G,
(ii) an arbitrary node in G, chosen only by its node label, or
(iii) any number of such nodes.

Graph extension grammars are essentially a restriction of contextual hyperedge-replacement grammars [18]. By way of example, we have shown that they can model both the structural and non-structural reentrancies that are common in semantic graphs of formalisms such as AMR.

We have provided a parsing algorithm for our formalism and proved it to be correct and have a polynomial-time running time. Finally, we have formulated conditions on the grammar that, if met, make parsing particularly efficient.

There are several promising directions for future work. On the theoretical side, we would like to apply the new ideas presented here to the formalisms by Groschwitz et al. [24] and Björklund et al. [4] to see if also these can be made to accommodate contextual rules without sacrificing parsing efficiency. It is also natural to generalise graph extension grammars to string-to-graph or tree-to-graph transducers to facilitate translation from natural-language sentences or dependency trees to AMR graphs. On the empirical side, we are interested in algorithms for inferring extension and union operations from AMR corpora, and in training neural networks to translate between sentences and AMR graphs using trees over graph extension algebras as an intermediate representation. Such efforts would make the new formalism available to current data-driven approaches in NLP, with the aim of adding structure and interpretability to machine-learning workflows.

REFERENCES

[1] IJsbrand Jan Aalbersberg, Grzegorz Rozenberg, and Andrzej Ehrenfeucht. On the membership problem for regular DNLC grammars. *Discrete Applied Mathematics*, 13(1):79–85, 1986.

[2] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse, Sofia, Bulgaria*, pages 178–186. Association for Computational Linguistics, 2013.

[3] Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20(1):83–127, 1987.

[4] Henrik Björklund, Frank Drewes, and Petter Ericson. Between a rock and a hard place - uniform parsing for hyperedge replacement DAG grammars. In Adrian-Horia Dediu, Jan Janousek, Carlos Martín-Vide, and Bianca Truthe, editors, *the 10th International Conference on Language and Automata Theory and Applications, LATA 2016, Prague, Czech Republic*, volume 9618 of *Lecture Notes in Computer Science*, pages 521–532. Springer, 2016.

[5] Henrik Björklund, Frank Drewes, and Petter Ericson. Parsing weighted order-preserving hyperedge replacement grammars. In *Proceedings of the 16th Meeting on the Mathematics of Language, Toronto, Canada*, pages 1–11. Association for Computational Linguistics, July 2019.

[6] Henrik Björklund, Frank Drewes, Petter Ericson, and Florian Starke. Uniform parsing for hyperedge replacement grammars. *Journal of Computer and System Sciences*, 118:1–27, 2021.

[7] Johannes Blum and Frank Drewes. Language theoretic properties of regular dag languages. *Information and Computation*, 265:57–76, 2019.

[8] Fabienne Braune, Daniel Bauer, and Kevin Knight. Mapping between English strings and reentrant semantic graphs. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014, Reykjavik, Iceland*, pages 4493–4498. European Language Resources Association, 2014.

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[10] David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. Parsing graphs with hyperedge replacement grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Sofia, Bulgaria*, pages 924–932, 2013.

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

[12] Bruno Courcelle. The monadic second-order logic of graphs V: on closing the gap between definability and recognizability. *Theoretical Computer Science*, 80:153–202, 1991.

[13] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*. Cambridge University Press, 2012.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota*, pages 4171–4186. Association for Computational Linguistics, 2019.

[15] Frank Drewes. *Grammatical Picture Generation – A Tree-Based Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[16] Frank Drewes and Berthold Hoffmann. Contextual hyperedge replacement. *Acta Informatica*, 52(6):497–524, 2015.

[17] Frank Drewes, Berthold Hoffmann, Dirk Janssens, and Mark Minas. Adaptive star grammars and their languages. *Theoretical Computer Science*, 411:3090–3109, 2010.

[18] Frank Drewes, Berthold Hoffmann, and Mark Minas. Contextual hyperedge replacement. In A. Schürr, D. Varro, and G. Varro, editors, *Applications of Graph Transformations with Industrial Relevance: 4th International Symposium, Revised Selected and Invited Papers*, number 7233 in Lecture Notes in Computer Science, pages 182–197. Springer, 2012.

[19] Frank Drewes, Berthold Hoffmann, and Mark Minas. Extending predictive shift-reduce parsing to contextual hyperedge replacement grammars. In Esther Guerra and Fernando Orejas, editors, *Proceedings of the 12th International Conference on Graph Transformation, ICGT 2017, Eindhoven, The Netherlands*, volume 11629 of *Lecture Notes in Computer Science*, pages 55–72. Springer International Publishing, 2017.

[20] Frank Drewes, Berthold Hoffmann, and Mark Minas. Predictive shift-reduce parsing for hyperedge replacement grammars. In Juan de Lara and Detlef Plump, editors, *Proceedings of the 10th International Conference on Graph Transformation, ICGT 2017, Marburg, Germany*, volume 10373 of *Lecture Notes in Computer Science*, pages 106–122. Springer, 2017.

[21] Frank Drewes, Berthold Hoffmann, and Mark Minas. Rule-based top-down parsing for acyclic contextual hyperedge replacement grammars. In *Proceedings of the 14th International Conference on Graph Transformation, ICGT 2021*, Lecture Notes in Computer Science, 2021.

[22] Frank Drewes and Anna Jonsson. Contextual hyperedge replacement grammars for abstract meaning representations. In *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms, Umeå, Sweden*, pages 102–111. Association for Computational Linguistics, 2017.

[23] Frank Drewes, Hans-Jörg Kreowski, and Annegret Habel. Hyperedge replacement graph grammars. In *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 95–162. World Scientific Publishing Co., Inc., 1997.

[24] Jonas Groschwitz, Meaghan Fowlie, Mark Johnson, and Alexander Koller. A constrained graph algebra for semantic parsing with AMRs. In *Proceedings of the 12th International Conference on Computational Semantics (Long papers), IWCS 2017, Montpellier, France*. Association for Computational Linguistics, 2017.

[25] Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. Graph parsing with s-graph grammars. In *Proc. 53rd Ann. Meeting of the Association for Computational Linguistics and the 7th Intl. Joint Conf. on Natural Language Processing (Volume 1: Long Papers)*, pages 1481–1490, 2015.

[26] Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. AMR dependency parsing with a typed semantic algebra. In I. Gurevych and Y. Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), ACL 2018*, pages 1831–1841. Association for Computational Linguistics, 2018.

[27] Annegret Habel and Hans-Jörg Kreowski. Some structural aspects of hypergraph languages generated by

hyperedge replacement. In Franz J. Brandenburg, Guy Vidal-Naquet, and Martin Wirsing, editors, *STACS 1987*, pages 207–219, Berlin, Heidelberg, 1987. Springer.

[28] Robert T. Kasper. A flexible interface for linking applications to Penman's sentence generator. In *Proceedings of the workshop on Speech and Natural Language, Philadelphia, Pennsylvania, HLT 1989*, pages 153–158. Association for Computational Linguistics, 1989.

[29] Klaus-Jörn Lange and Emo Welzl. String grammars with disconnecting or a basic root of the difficulty in graph grammar parsing. *Discrete Applied Mathematics*, 16(1):17 – 30, 1987.

[30] Irene Langkilde and Kevin Knight. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (Volume 1), Montreal, Quebec*, pages 704–710, 1998.

[31] Clemens Lautemann. The complexity of graph languages generated by hyperedge replacement. *Acta Informatica*, 27(5):399–421, 1990.

[32] Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. Compositional semantic parsing across graphbanks. In A. Korhonen, D. R. Traum, and L. Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics (Volume 1: Long Papers), Florence, Italy, ACL 2019*, pages 4576–4585. Association for Computational Linguistics, 2019.

[33] Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. Fast semantic parsing with well-typedness guarantees. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 3929–3951. Association for Computational Linguistics, 2020.

[34] Jorge Mezei and Jesse B. Wright. Algebraic automata and context-free sets. *Information and Control*, 11:3–29, 1967.

[35] Daniel Quernheim and Kevin Knight. Towards probabilistic acceptors and transducers for feature structures. In *Proceedings of the Sixth Workshop on Syntax, Semantics and Structure in Statistical Translation, Jeju, Republic of Korea*, pages 76–85. Association for Computational Linguistics, July 2012.