Parsing Graphs with Regular Graph Grammars

Sorcha Gilroy University of Edinburgh s.gilroy@sms.ed.ac.uk
Adam Lopez University of Edinburgh alopez@inf.ed.ac.uk
Sebastian Maneth Universität Bremen smaneth@uni-bremen.de

Abstract

Recently, several datasets have become available which represent natural language phenomena as graphs. Hyperedge Replacement Languages (HRL) have been the focus of much attention as a formalism to represent the graphs in these datasets. Chiang et al. (2013) prove that HRL graphs can be parsed in polynomial time with respect to the size of the input graph. We believe that HRL are more expressive than is necessary to represent semantic graphs and we propose the use of Regular Graph Languages (RGL; Courcelle 1991), which is a subfamily of HRL, as a possible alternative. We provide a top-down parsing algorithm for RGL that runs in time linear in the size of the input graph.

1 Introduction

NLP systems for machine translation, summarization, paraphrasing, and other tasks often fail to preserve the compositional semantics of sentences and documents because they model language as bags of words, or at best syntactic trees. To preserve semantics, they must model semantics. In pursuit of this goal, several datasets have been produced which pair natural language with compositional semantic representations in the form of directed acyclic graphs (DAGs), including the Abstract Meaning Representation Bank (AMR; Banarescu et al. 2013), the Prague Czech-English Dependency Treebank (Hajič et al., 2012), Deepbank (Flickinger et al., 2012), and the Universal Conceptual Cognitive Annotation (Abend and Rappoport, 2013). To make use of this data, we require models of graphs.

Consider how we might use compositional semantic representations in machine translation

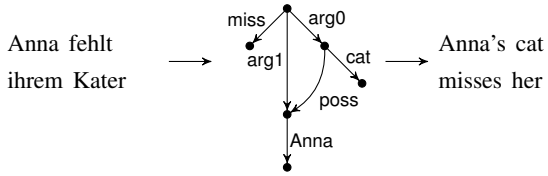

Figure 1: Semantic machine translation using AMR (Jones et al., 2012). The edge labels identify 'cat' as the object of the verb 'miss', 'Anna' as the subject of 'miss' and 'Anna' as the possessor of 'cat'. Edges whose head nodes are not attached to any other edge are interpreted as node labels.

(Figure 1), a two-step process in which semantic analysis is followed by generation. Jones et al. (2012) observe that this decomposition can be modeled with a pair of synchronous grammars, each defining a relation between strings and graphs. Necessarily, one projection of this synchronous grammar produces strings, while the other produces graphs, i.e., is a **graph grammar**. A consequence of this representation is that the complete translation process can be realized by parsing: to analyze a sentence, we parse the input string with the string-generating projection of the synchronous grammar, and read off the synchronous graph from the resulting parse. To generate a sentence, we parse the graph, and read off the synchronous string from the resulting parse. In this paper, we focus on the latter problem: using graph grammars to parse input graphs. We call this **graph recognition** to avoid confusion with other parsing problems.

Recent work in NLP has focused primarily on **hyperedge replacement grammar** (HRG; Drewes et al. 1997), a context-free graph grammar formalism that has been studied in an NLP context by several researchers (Chiang et al., 2013; Peng et al., 2015; Bauer and Rambow, 2016). In particular, Chiang et al. (2013) propose that HRG could be used to represent semantic graphs, and precisely characterize the complexity of a CKY-style

*Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017)*, pages 199–208,
Vancouver, Canada, August 3-4, 2017.

algorithm for graph recognition from Lautemann (1990) to be polynomial in the size of the input graph. HRGs are very expressive—they can generate graphs that simulate non-context-free string languages (Engelfriet and Heyker, 1991; Bauer and Rambow, 2016). This means they are likely more expressive than we need to represent the linguistic phenomena that appear in existing semantic datasets. In this paper, we propose the use of Regular Graph Grammars (RGG; Courcelle 1991) a subfamily of HRG that, like its regular counterparts among string and tree languages, is less expressive than context-free grammars but may admit more practical algorithms. By analogy to Chiang's CKY-style algorithm for HRG. We develop an Earley-style recognition algorithm for RGLs that is linear in the size of the input graph.

2 Regular Graph Languages

We use the following notation. If n is an integer, $[n]$ denotes the set $\{1, \ldots, n\}$. Let Γ be an alphabet, i.e., a finite set. Then $s \in \Gamma^*$ denotes that s is a sequence of arbitrary length, each element of which is in Γ. We denote by $|s|$ the length of s. A **ranked alphabet** is an alphabet Γ paired with an arity mapping (i.e., a total function) rank: $\Gamma \to \mathbb{N}$.

Definition 1. *A **hypergraph** (or simply **graph**) over a ranked alphabet Γ is a tuple $G = (V_G, E_G, att_G, lab_G, ext_G)$ where V_G is a finite set of nodes; E_G is a finite set of edges (distinct from V_G); $att_G : E_G \to V_G^*$ maps each edge to a sequence of nodes; $lab_G : E_G \to \Gamma$ maps each edge to a label such that $|att_G(e)| = rank(lab_G(e))$; and ext_G is an ordered subset of V_G called the **external nodes** of G.*

We assume that the elements of ext_G are pairwise distinct, and the elements of $\text{att}_G(e)$ for each edge e are also pairwise distinct. An edge e is attached to its nodes by **tentacles**, each labeled by an integer indicating the node's position in $\text{att}_G(e) = (v_1, \ldots, v_k)$. The tentacle from e to v_i will have label i, so the tentacle labels lie in the set $[k]$ where $k = \text{rank}(e)$. To express that a node v is attached to the ith tentacle of an edge e, we say $\text{vert}(e, i) = v$. Likewise, the nodes in ext_G are labeled by their position in ext_G. We refer to the ith external node of G by $\text{ext}_G(i)$ and in figures this will be labeled (i). The **rank** of an edge e is k if $\text{att}(e) = (v_1, \ldots, v_k)$ (or equivalently, rank(lab(e)) = k). The **rank** of a hypergraph G, denoted by rank(G) is the size of ext_G.

Example 1. Hypergraph G in Figure 2 has four nodes (shown as black dots) and three hyperedges labeled a, b, and X (shown boxed). The bracketed numbers (1) and (2) denote its external nodes and the numbers between edges and the nodes are tentacle labels. Call the top node v_1 and, proceeding clockwise, call the other nodes v_2, v_3, and v_4. Call its edges e_1, e_2 and e_3. Its definition would state $\text{att}_G(e_1) = (v_1, v_2)$, $\text{att}_G(e_2) = (v_2, v_3)$, $\text{att}_G(e_3) = (v_1, v_4, v_3)$, $\text{lab}_G(e_1) = a$, $\text{lab}_G(e_2) = b$, $\text{lab}_G(e_3) = X$, and $\text{ext}_G = (v_4, v_2)$.

Definition 2. *Let G be a hypergraph containing an edge e with $att_G(e) = (v_1, \ldots, v_k)$ and let H be a hypergraph of rank k with node and edge sets disjoint from those of G. The **replacement** of e by H is the graph $G' = G[e/H]$. Its node set $V_{G'}$ is $V \cup V_H$ where $V = V_G - \{v_1, \ldots, v_k\}$. Its edge set is $E_{G'} = (E_G - \{e\}) \cup E_H$. We define $att_{G'} = att \cup att_H$ where for every $e' \in (E_G - \{e\})$, $att(e)$ is obtained from $att_G(e')$ by replacing v_i by the ith external node of H. Let $lab_{G'} = lab \cup lab_H$ where lab is the restriction of lab_G to edges in $E_G - \{e\}$. Finally, let $ext_{G'} = ext_G$.*

Example 2. A replacement is shown in Figure 2.

2.1 Hyperedge Replacement Grammars

Definition 3. *A **hyperedge replacement grammar** $\mathcal{G} = (N_\mathcal{G}, T_\mathcal{G}, P_\mathcal{G}, S_\mathcal{G})$ consists of ranked (disjoint) alphabets $N_\mathcal{G}$ and $T_\mathcal{G}$ of nonterminal and terminal symbols, respectively, a finite set $P_\mathcal{G}$ of productions, and a start symbol $S_\mathcal{G} \in N_\mathcal{G}$. Every production in $P_\mathcal{G}$ is of the form $X \to G$ where G is a hypergraph over $N_\mathcal{G} \cup T_\mathcal{G}$ and $rank(G) = rank(X)$.*

For each production $p : X \to G$, we use $L(p)$ to refer to X (the left-hand side of p) and $R(p)$ to refer to G (the right-hand side of p). An edge is a **terminal edge** if its label is terminal and a **nonterminal edge** if its label is nonterminal. A graph is a **terminal graph** if all of its edges are terminal. The **terminal subgraph** of a graph is the subgraph consisting of all terminal edges and their incident nodes.

Given a HRG $\mathcal{G}$, we say that graph G **immediately derives** graph G', denoted $G \to G'$, iff there is an edge $e \in E_G$ and a nonterminal $X \in N_\mathcal{G}$ such that $\text{lab}_G(e) = X$ and $G' = G[e/H]$, where $X \to H$ is in $P_\mathcal{G}$. We extend the idea of immediate derivation to its transitive closure $G \to^* G'$, and say here that G **derives** G'. For every $X \in N_\mathcal{G}$ we also use X to de-

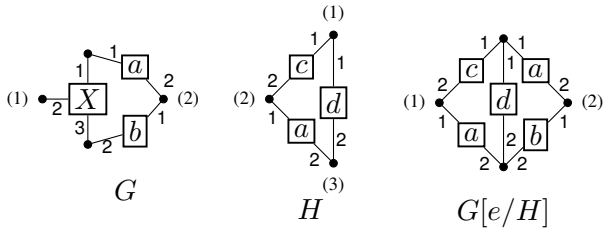

Figure 2: The replacement of the X-labeled edge e in G by the graph H.

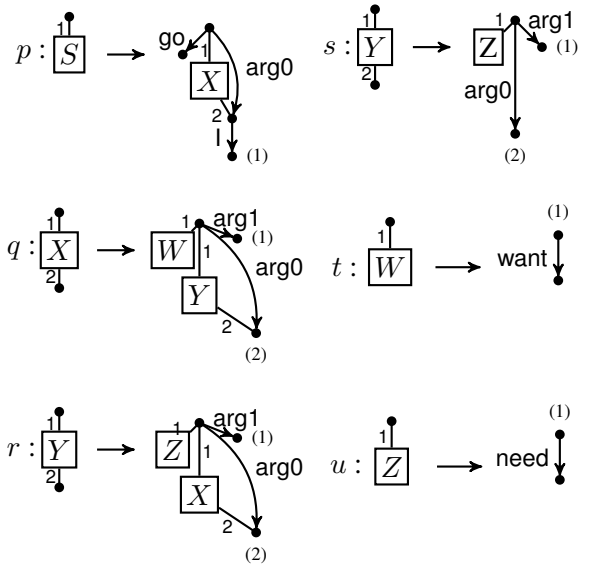

Table 1: Productions of a HRG. The labels p, q, r, s, t, and u label the productions so that we can refer to them in the text. Note that Y can rewrite in two ways, either via production r or s.

note the graph consisting of a single edge e with $\text{lab}(e) = X$ and nodes $(v_1, \ldots, v_{\text{rank}(X)})$ such that $\text{att}_G(e) = (v_1, \ldots, v_{\text{rank}(X)})$, and we define the language $L_X(\mathcal{G})$ as $\{G \mid X \rightarrow^* G \wedge G \text{ is terminal}\}$. The **language of** $\mathcal{G}$ is $L(\mathcal{G}) = L_{S_\mathcal{G}}(\mathcal{G})$. We call the family of languages that can be produced by any HRG the **hyperedge replacement languages (HRL)**.

We assume that terminal edges are always of rank 2, and depict them as directed edges where the direction is determined by the tentacle labels: the tentacle labeled 1 attaches to the source of the edge and the tentacle labeled 2 attaches to the target of the edge.

Example 3. Table 1 shows a HRG deriving AMR graphs for sentences of the form 'I need to want to need to want to ... to want to go'. Figure 3 is a graph derived by the grammar. The grammar is somewhat unnatural, a point we will return to (§4).

We can use HRGs to generate chain graphs

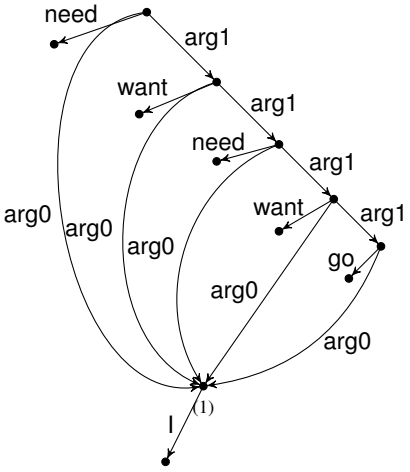

Figure 3: Graph derived by grammar in Table 1.

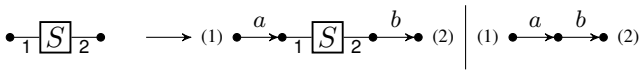

Figure 4: A HRG producing the string language $a^n b^n$.

(strings) by restricting the form of the productions in the grammars. Figure 4 shows a HRG that produces the context-free string language $a^n b^n$. HRGs can simulate the class of mildly context-sensitive languages that is characterized, e.g., by linear context-free rewriting systems (LCFRS;Vijay-Shanker et al. 1987), where the fan-out of the LCFRS will influence the maximum rank of nonterminal required in the HRG, see (Engelfriet and Heyker, 1991).

2.2 Regular Graph Grammars

A regular graph grammar (RGG; Courcelle 1991) is a restricted form of HRG. To explain the restrictions, we first require some definitions.

Definition 4. *Given a graph G, a **path** in G from a node v to a node v' is a sequence*

$$(v_0, i_1, e_1, j_1, v_1)(v_1, i_2, e_2, j_2, v_2) \ldots (v_{k-1}, i_k, e_k, j_k, v_k) \quad (1)$$

such that $v_0 = v, v_k = v'$, and for each $r \in [k]$, $vert(e_r, i_r) = v_{r-1}$ and $vert(e_r, j_r) = v_r$. The length of this path is k.

A path is **terminal** if every edge in the path has a terminal label. A path is **internal** if each v_i is internal for $1 \leq i \leq k-1$. Note that the endpoints v_0 and v_k of an internal path can be external.

Definition 5. *A HRG $\mathcal{G}$ is a **Regular Graph Grammar** (or simply **RGG**) if each nonterminal in $N_\mathcal{G}$ has rank at least one and for each $p \in P_\mathcal{G}$ the following hold:*

(C1) $R(p)$ has at least one edge. Either it is a single terminal edge, all nodes of which are external, or each of its edges has at least one internal node.

(C2) Every pair of nodes in $R(p)$ is connected by a terminal and internal path.

Example 4. The grammar in Table 1 is an RGG. Although HRGs can produce context-free languages (and beyond) as shown in Figure 4, the only string languages RGGs can produce are the regular string languages. See Figure 5 for an example of a string generating RGG. Similarly, RGGs can produce regular tree languages, but not context-free tree languages. Figure 6 shows a tree generating RGG that generates binary trees the internal nodes of which are represented by a-labeled edges, and the leaves of which are represented by b-labeled edges. Note that these two results of regularity of the string- and tree-languages generated by RGG follow from the fact that graph languages produced by RGG are MSO-definable (Courcelle, 1991), and the well-known facts that the regular string and graph languages are MSO-definable.

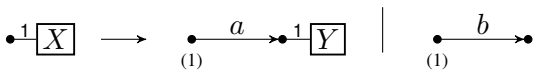

Figure 5: A RGG for a regular string language.

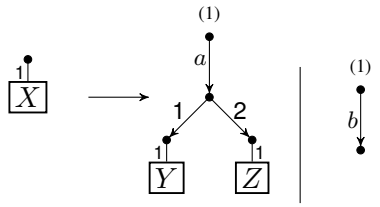

Figure 6: A RGG for a regular tree language.

We call the family of languages generated by RGGs the **regular graph languages** (RGLs).

3 RGL Recognition

To recognize RGG, we exploit the property that every nonterminal including the start symbol has rank at least one (Definition 5), and we assume that the corresponding external node is identified in the input graph. This mild assumption may be reasonable for applications like AMR parsing, where grammars could be designed so that the external node is always the unique root. Later we relax this assumption.

The availability of an identifiable external node suggests a top-down algorithm, and we take inspiration from a top-down recognition algorithm for the predictive top-down parsable grammars, another subclass of HRG (Drewes et al., 2015). These grammars, the graph equivalent of LL(1) string grammars, are incomparable to RGG, but the algorithms are related in their use of top-down prediction and in that they both fix an order of the edges in the right-hand side of each production.

3.1 Top-Down Recognition for RGLs

Just as the algorithm of Chiang et al. (2013) generalizes CKY to HRG, our algorithm generalizes Earley's algorithm (Earley, 1970). Both algorithms operate by recognizing incrementally larger subgraphs of the input graph, using a succinct representation for subgraphs that depends on an arbitrarily chosen **marker node** m of the input graph.

Definition 6. *(Chiang et al. 2013; Definition 6) Let I be a subgraph of a graph G. A **boundary node** of I is a node which is either an endpoint of an edge in $G \backslash I$ or an external node of G. A **boundary edge** of I is an edge in I which has a boundary node as an endpoint. The **boundary representation** of I is the tuple $b(I) = \langle bn(I), be(I), m \in I \rangle$ where*

1. *$bn(I)$ is the set of boundary nodes of I*
2. *$be(I)$ is the set of boundary edges of I*
3. *$(m \in I)$ is a flag indicating whether the marker node is in I.*

Chiang et al. (2013) prove each subgraph has a unique boundary representation, and give algorithms that use only boundary representations to compute the union of two subgraphs, requiring time linear in the number of boundary nodes; and to check disjointness of subgraphs, requiring time linear in the number of boundary edges.

For each production p of the grammar, we impose a fixed order on the edges of $R(p)$, as in Drewes et al. (2015). We discuss this order in detail in §3.2. As in Earley's algorithm, we use dotted rules to represent partial recognition of productions: $X \to \bar{e}_1 \dots \bar{e}_{i-1} \bullet \bar{e}_i \dots \bar{e}_n$ means that we have identified the edges $\bar{e}_1$ to $\bar{e}_{i-1}$ and that we must next recognize edge $\bar{e}_i$. We write $\bar{e}$ and $\bar{v}$ for edges and nodes in productions and e and v for edges and nodes in a derived graph. When the identity of the sequence is immaterial we abbreviate it as α, for example writing $X \to \bullet \alpha$.

We present our recognizer as a deductive proof system (Shieber et al., 1995). The items of the

Name	Rule	Conditions
PREDICT	$\frac{[b(I), p : X \to \bar{e}_1 \dots \bullet\ \bar{e}_i \dots \bar{e}_n, \phi_p][q : Y \to \alpha]}{[\phi_p(\bar{e}_i), q : Y \to \bullet\ \alpha, \phi_q^0[\text{ext}_{R(q)} = \phi_p(\bar{e}_i)]]}$	$\text{lab}(\bar{e}_i) = Y$
SCAN	$\frac{[b(I), X \to \bar{e}_1 \dots \bullet\ \bar{e}_i \dots \bar{e}_n, \phi_p][e = \text{edg}_{\text{lab}(\bar{e}_i)}(v_1, \dots, v_m)]}{[b(I \cup \{e\}), X \to \bar{e}_1 \dots \bullet\ \bar{e}_{i+1} \dots \bar{e}_n, \phi_p[\text{att}(\bar{e}_i) = (v_1, \dots, v_m)]]}$	$\phi_p(\bar{e}_i)(j) \in V_G \Rightarrow$ $\phi_p(\bar{e}_i)(j) = \text{vert}(e, j)$
COMPLETE	$\frac{[b(I), p : X \to \bar{e}_1 \dots \bullet\ \bar{e}_i \dots \bar{e}_n, \phi_p][b(J), q : Y \to \alpha\ \bullet, \phi_q]}{[b(I \cup J), X \to \bar{e}_1 \dots \bullet\ \bar{e}_{i+1} \dots \bar{e}_n, \phi_p[\text{att}(\bar{e}_i) = \phi_p(\text{ext}_{R(q)})]]}$	$\phi_p(\bar{e}_i)(j) \in V_G \Rightarrow$ $\phi_p(\bar{e}_i)(j) =$ $\phi_q(\text{ext}_{R(q)})(j)$, $\text{lab}(\bar{e}_i) = Y$, $E_I \cap E_J = \emptyset$

Table 2: The inference rules for the top-down recognizer.

recognizer are of the form

$$[b(I), p : X \to \bar{e}_1 \dots \bullet\ \bar{e}_i \dots \bar{e}_n, \phi_p]$$

where I is a subgraph that has been recognized as matching $\bar{e}_1, \dots, \bar{e}_{i-1}$; $p : X \to \bar{e}_1, \dots, \bar{e}_n$ is a production in the grammar with the edges in order; and $\phi_p : E_{R(p)} \to V_G^*$ maps the endpoints of edges in $R(p)$ to nodes in G.

For each production p, we number the nodes in some arbitrary but fixed order. Using this, we construct the function $\phi_p^0 : E_{R(p)} \to V_{R(p)}^*$ such that for $\bar{e} \in E_{R(p)}$ if $\text{att}(\bar{e}) = (\bar{v}_1, \bar{v}_2)$ then $\phi_p^0(\bar{e}) = (\bar{v}_1, \bar{v}_2)$. As we match edges in the graph with edges in p, we assign the nodes $\bar{v}$ to nodes in the graph. For example, if we have an edge $\bar{e}$ in a production p such that $\text{att}(\bar{e}) = (\bar{v}_1, \bar{v}_2)$ and we find an edge e which matches $\bar{e}$, then we update ϕ_p to record this fact, written $\phi_p[\text{att}(\bar{e}) = \text{att}(e)]$. We also use ϕ_p to record assignments of external nodes. If we assign the ith external node to v, we write $\phi_p[\text{ext}_p(i) = v]$. We write ϕ_p^0 to represent a mapping with no grounded nodes.

Since our algorithm makes top-down predictions based on known external nodes, our boundary representation must cover the case where a subgraph is empty except for these nodes. If at some point we know that our subgraph has external nodes $\phi(\bar{e})$, then we use the shorthand $\phi(\bar{e})$ rather than the full boundary representation $\langle \phi(\bar{e}), \emptyset, m \in \phi(\bar{e}) \rangle$.

To keep notation uniform, we use dummy nonterminal $S^* \notin N_\mathcal{G}$ that derives $S_\mathcal{G}$ via the production p_0. For graph G, our system includes the **axiom**:

$$[\text{ext}_G, p_0 : S^* \to \bullet\ S_\mathcal{G}, \phi_{p_0}^0[\text{ext}_{R(p_0)} = \text{ext}_G]].$$

Our **goal** is to prove:

$$[b(G), p_S : S^* \to S_\mathcal{G}\ \bullet, \phi_{p_S}]$$

where ϕ_{p_S} has a single edge $\bar{e}$ in its domain which has label $S_\mathcal{G}$ in $R(p_S)$ and $\phi_{p_S}(\bar{e}) = \text{ext}_G$.

As in Earley's algorithm, we have three inference rules: PREDICT, SCAN and COMPLETE (Table 2). PREDICT is applied when the edge after the dot is nonterminal, assigning any external nodes that have been identified. SCAN is applied when the edge after the dot is terminal. Using ϕ_p, we may already know where some of the endpoints of the edge should be, so it requires the endpoints of the scanned edge to match. COMPLETE requires that each of the nodes of $\bar{e}_i$ in $R(p)$ have been identified, these nodes match up with the corresponding external nodes of the subgraph J, and that the subgraphs I and J are edge-disjoint.

We provide a high-level proof that the recognizer is sound and complete.

Proposition 1. *Let $\mathcal{G}$ be a HRG and G a graph. Then the goal $[b(G), p_S : S^* \to S_\mathcal{G}\ \bullet, \phi_{p_S}]$ can be proved from the axiom $[\text{ext}_G, p_S : S^* \to \bullet\ S_\mathcal{G}, \phi_{p_S}[\text{ext}_{R(p_S)} = \text{ext}_G]]$ if and only if $G \in L(\mathcal{G})$.*

Proof. We prove that for each $X \in N_\mathcal{G}$, $[b(G), p_X : X^* \to X\ \bullet, \phi_{p_X}]$ can be proved from $[\text{ext}_G, p_X : X^* \to \bullet\ X, \phi_{p_X}[\text{ext}_{R(p_X)} = \text{ext}_G]]$ if and only if $G \in L_X(\mathcal{G})$ where the dummy nonterminal X^* was added to the set of nonterminals and $p_X : X^* \to X$ was added to the set of productions. We prove this by induction on the number of edges in G.

We assume that each production in the grammar contains at least one terminal edge. If the HRG is not in this form, it can be converted into this form

and in the case of RGGs they are already in this form by definition.

Base Case: Let G consist of a single edge.

If: Assume $G \in L_X(\mathcal{G})$. Since G consists of one edge, there must be a production $q : X \to G$. Apply PREDICT to the axiom and $p_X : X^* \to X$ to obtain the item $[\phi_{p_X}(X), q : X \to \bullet\, G, \phi_q^0[\text{ext}_G = \phi_{p_X}(X)]]$. Apply SCAN to the single terminal edge that makes up G to obtain $[b(G), q : X \to G\, \bullet, \phi_q]$ and finally apply COMPLETE to this and the axiom reach the goal $[b(G), p_X : X^* \to X, \phi_{p_X}]$.

Only if: Assume the goal can be reached from the axiom and $G = e$. Then the item $[b(e), q : X \to e, \phi_q]$ must have been reached at some point for some $q \in P_{\mathcal{G}}$. Therefore $q : X \to e$ is a production and so $e = G \in L_X(\mathcal{G})$.

Assumption: Assume that the proposition holds when G has fewer than k edges.

Inductive Step: Assume G has k edges.

If: Assume $G \in L_X(\mathcal{G})$, then there is a production $q : X \to H$ where H has nonterminals $Y_1, \ldots, Y_n$ and there are graphs $H_1, \ldots, H_n$ such that $G = H[Y_1/H_1] \ldots [Y_n/H_n]$. Each graph H_i for $i \in [n]$ has fewer than k edges and so we apply the inductive hypothesis to show that we can prove the items $[b(H_i), r_i : Y_i \to J_i, \phi_{r_i}]$ for each $i \in [n]$. By applying COMPLETE to each such item and applying SCAN to each terminal edge of H we reach the goal $[b(G), p_X : X^* \to X\, \bullet, \phi_{p_X}]$.

Only If: Assume the goal can be proved from the axiom. Then we must have at some point reached an item of the form $[b(G), q : X \to H, \phi_q]$ and that H has nonterminals $Y_1, \ldots, Y_n$. This means that there are graphs $H_1, \ldots, H_n$ such that $[b(H_i), p_{Y_i} : Y_i^* \to Y_i, \phi_{p_{Y_i}}]$ for each $i \in [n]$ and $G = H[Y_1/H_1] \ldots [Y_n/H_n]$. Since each H_i has fewer than k edges, we apply the inductive hypothesis to get that $H_i \in L_{Y_i}(\mathcal{G})$ for each $i \in [n]$ and therefore $G \in L_X(\mathcal{G})$. □

Example 5. Using the RGG in Table 1, we show how to recognize the graph in Figure 7, which can be derived by applying production s followed by production u, where the external nodes of Y are (v_3, v_2). Assume the ordering of the edges in production s is arg1, arg0, Z; the top node is $\bar{v}_1$; the bottom node is $\bar{v}_2$; and the node on the right is $\bar{v}_3$; and that the marker node is not in this subgraph—we elide reference to it for simplicity. Let $\bar{v}_4$ be the top node of $R(u)$ and $\bar{v}_5$ be the bottom node of $R(u)$. The external nodes of Y are determined top-down, so the recognize of this subgraph is triggered by this item:

$$[\{v_3, v_2\}, Y \to \bullet\ \text{arg1 arg0}\ Z, \quad \phi_s^0[\text{ext}_{R(s)} = (v_3, v_2)]] \quad (2)$$

where $\phi_s(\text{arg1}) = (\bar{v}_1, v_3)$, $\phi_s(\text{arg0}) = (\bar{v}_1, v_2)$, and $\phi_s(Z) = (\bar{v}_1)$.

Table 3 shows how we can prove the item

$$[\langle\{v_3, v_2\}, \{e_3, e_2\}\rangle, Y \to \text{arg1arg0}Z\ \bullet, \phi]$$

The boundary representation $\langle\{v_3, v_2\}, \{e_3, e_2\}\rangle$ in this item represents the whole subgraph shown in Figure 7.

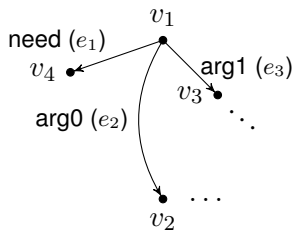

Figure 7: Top left subgraph of Figure 3. To refer to nodes and edges in the text, they are labeled v_1, v_2, v_3, e_1, e_2, and e_3.

3.2 Normal Ordering

Our algorithm requires a fixed ordering of the edges in the right-hand sides of each production. We will constrain this ordering to exploit the structure of RGG productions, allowing us to bound recognition complexity. If $s = \bar{e}_1 \ldots \bar{e}_n$ is an order, define $s_{i:j} = \bar{e}_i \ldots \bar{e}_j$.

Definition 7. *Let $s = \bar{e}_1, \ldots, \bar{e}_n$ be an edge order of a right-hand side of a production. Then s is **normal** if it has the following properties:*

1. *$\bar{e}_1$ is connected to an external node,*
2. *$s_{1:j}$ is a connected graph for all $j \in [n]$*
3. *if $\bar{e}_i$ is nonterminal, each endpoint of $\bar{e}_i$ must be incident with some terminal edge $\bar{e}_j$ for which $j < i$.*

Example 6. The ordering of the edges of production s in Example 5 is normal.

Arbitrary HRGs do not necessarily admit a normal ordering. For example, the graph in Figure 8 cannot satisfy Properties 2 and 3 simultaneously. However, RGGs do admit a normal ordering.

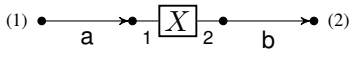

Figure 8: This graph cannot be normally ordered.

Current Item	Reason
1. $[\{v_3, v_2\}, Y \rightarrow \bullet \text{ arg1arg0}Z, \phi_s^0[\text{ext}_{R(s)} = (v_3, v_2)]]$	Equation 2
2. $[\langle\{v_3, v_2, v_1\}, \{e_3\}\rangle, Y \rightarrow \text{arg1} \bullet \text{arg0}Z, \phi_s[\text{att}(\text{arg1}) = (v_1, v_3)]]$	SCAN: 1. and $e_3 = \text{edg}_{\text{arg1}}(v_1, v_3)$
3. $[\langle\{v_3, v_2, v_1\}, \{e_3, e_2\}\rangle, Y \rightarrow \text{arg1arg0} \bullet Z, \phi_s[\text{att}(\text{arg0}) = (v_1, v_2)]]$	SCAN: 2. and $e_2 = \text{edg}_{\text{arg0}}(v_1, v_2)$]
4. $[(v_1), Z \rightarrow \bullet \text{ need}, \phi_u^0[\text{ext}_{R(u)} = (v_1)]]$	PREDICT: 3. and $Z \rightarrow$ need
5. $[\langle\{v_1, v_4\}, \{e_1\}\rangle, Z \rightarrow \text{need} \bullet, \phi_u[\text{att}(\text{need}) = (v_1, v_4)]]$	SCAN: 4. and $e_1 = \text{edg}_{\text{need}}(v_1, v_4)$
6. $[\langle\{v_3, v_2\}, \{e_3, e_2\}\rangle, Y \rightarrow \text{arg1arg0}Z \bullet, \phi_s[\text{att}(Z) = (v_1)]]$	COMPLETE: 3. and 5.

Table 3: The steps of recognizing that the subgraph shown in Figure 7 is derived from productions r_2 and u in the grammar in Table 1.

Proposition 2. *If $\mathcal{G}$ is an RGG, for every $p \in P_{\mathcal{G}}$, there is a normal ordering of the edges in $R(p)$.*

Proof. If $R(p)$ contains a single node then it must be an external node and it must have a terminal edge attached to it since $R(p)$ must contain at least one terminal edge. If $R(p)$ contains multiple nodes then by C2 there must be terminal internal paths between all of them, so there must be a terminal edge attached to the external node, which we use to satisfy Property 1. To produce a normal ordering, we next select terminal edges once one of their endpoints is connected to an ordered edge, and nonterminal edges once all endpoints are connected to ordered edges, possible by C2. Therefore, Properties 2 and 3 are satisfied. □

A normal ordering tightly constrains the recognition of edges. Property 3 ensures that when we apply PREDICT, the external nodes of the predicted edge are all bound to specific nodes in the graph. Properties 1 and 2 ensure that when we apply SCAN, at least one endpoint of the edge is bound (fixed).

3.3 Recognition Complexity

Assume a normally-ordered RGG. Let the maximum number of edges in the right-hand side of any production be m; the maximum number of nodes in any right-hand side of a production k; the maximum degree of any node in the input graph d; and the number of nodes in the input graph n.

As previously mentioned, Drewes et al. (2015) also propose a HRG recognizer which can recognize a subclass of HRG (incomparable to RGG) called the predictive top-down parsable grammars. Their recognizer in this case runs in $\mathcal{O}(n^2)$ time. A well-known bottom-up recognizing algorithm for HRG was first proposed by Lautemann (1990). In this paper, the recognizer is shown to be polynomial in the size of the input graph. Later, Chiang et al. (2013) formulate the same algorithm more precisely and show that the recognizing complexity is $\mathcal{O}((3^d \times n)^{k+1})$ where k in their case is the treewidth of the grammar.

Remark 1. *The maximum number of nodes in any right-hand side of a production (k) is also the maximum number of boundary nodes for any subgraph in the recognizer.*

COMPLETE combines subgraphs I and J only when the entire subgraph derived from Y has been recognized. Boundary nodes of J are also boundary nodes of I because they are nodes in the terminal subgraph of $R(p)$ where Y connects. The boundary nodes of $I \cup J$ are also bounded by k since form a subset of the boundary nodes of I.

Remark 2. *Given a boundary node, there are at most $(d^m)^{k-1}$ ways of identifying the remaining boundary nodes of a subgraph that is isomorphic to the terminal subgraph of the right-hand side of a production.*

The terminal subgraph of each production is connected by C2, with a maximum path length of m. For each edge in the path, there are at most d subsequent edges. Hence for the $k-1$ remaining boundary nodes there are $(d^m)^{k-1}$ ways of choosing them.

We count instantiations of COMPLETE for an upper bound on complexity (McAllester, 2002), using similar logic to (Chiang et al., 2013). The number of boundary nodes of I, J and $I \cup J$ is at most k. Therefore, if we choose an arbitrary node to be some boundary node of $I \cup J$, there are at most $(d^m)^{k-1}$ ways of choosing its remaining boundary nodes. For each of these nodes, there are at most $(3^d)^k$ states of their attached boundary edges: in I, in J, or in neither. The total number

of instantiations is $\mathcal{O}(n(d^m)^{k-1}(3^d)^k)$, linear in the number of input nodes and exponential in the degree of the input graph. Note that in the case of the AMR dataset (Banarescu et al. 2013), the maximum node degree is 17 and the average is 2.12.

We observe that RGGs could be relaxed to produce graphs with no external nodes by adding a dummy nonterminal S' with rank 0 and a single production $S' \rightarrow S$. To adapt the recognition algorithm, we would first need to guess where the graph starts. This would add a factor of n to the complexity as the graph could start at any node.

4 Discussion and Conclusions

We have presented RGG as a formalism that could be useful for semantic representations and we have provided a top-down recognition algorithm for them. The constraints of RGG enable more efficient recognition than general HRG, and this tradeoff is reasonable since HRG is very expressive—when generating strings, it can express non-context-free languages (Engelfriet and Heyker, 1991; Bauer and Rambow, 2016), far more power than needed to express semantic graphs. On the other hand, RGG is so constrained that it may not be expressive enough: it would be more natural to derive the graph in Figure 4 from outermost to innermost predicate; but constraint C2 makes it difficult to express this, and the grammar in Table 1 does not. Perhaps we need less expressivity than HRG but more than RGG.

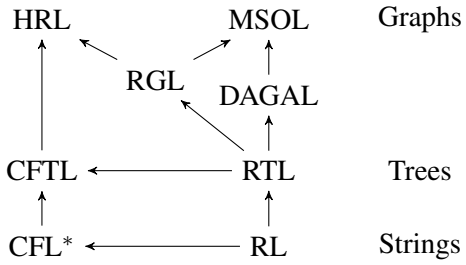

Figure 9: A Hasse diagram of various string, tree and graph language families. An arrow from family A to family B indicates that family A is a subfamily of family B.

A possible alternative would be to consider Restricted DAG Grammars (RDG; Björklund et al. 2016). Parsing for a fixed such grammar can be achieved in quadratic time with respect to the input graph. It is known that for a fixed HRG generating k-connected hypergraphs consisting of hyperedges of rank k only, parsing can be carried out in cubic time (k-HRG; (Drewes, 1993)).

More general than RDGs a is the class of graph languages recognized by DAG automata (DAGAL; Blum and Drewes 2016), for which the deterministic variant provides polynomial time parsing. Note that RGGs can generate graph languages of unbounded node degree. With respect to expressive power, RDGs and k-HRGs are incomparable to RGGs. Figure 9 shows the relationships between the context-free and regular languages for strings, trees and graphs. Monadic-second order logic (MSOL; Courcelle and Engelfriet 2011) is a form of logic which when restricted to strings gives us exactly the regular string languages and when restricted to trees gives us exactly the regular tree languages. RGLs lie in the intersection of HRG and MSOL on graphs but they do not make up this entire intersection. Courcelle (1991) defined (non-constructively) this intersection to be the strongly context-free languages (SCFL). We believe that there may be other formalisms that are subfamilies of SCFL which may be useful for semantic representations. All inclusions shown in Figure 9 are strict. For instance, RGL cannot produce "star graphs" (one node that has edges to n other nodes), while DAGAL and HRL can produce such graphs. It is well-known that HRL and MSOL are incomparable. There is a language in RGL that is not in DAGAL, for instance, "ladders" (two string graphs of n nodes each, with an edge between the ith node of each string).

Another alternative formalism to RGG that is defined as a restriction of HRG are Tree-like Grammars (TLG; Matheja et al. 2015). They define a subclass of SCFL, i.e., they are MSO definable. TLGs have been considered for program verification, where closure under intersection of the formalism is essential. Note that RGGs are also closed under intersection. While TLG and RDG are both incomparable to RGG, they share important characteristics, including the fact that the terminal subgraph of every production is connected. This means that our top-down recognition algorithm is applicable to both. In the future we would like to investigate larger, less restrictive (and more linguistically expressive) subfamilies of SCFL. We plan to implement and evaluate our algorithm experimentally.

Acknowledgments

This work was supported in part by the EPSRC Centre for Doctoral Training in Data Science,

funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh; and in part by a Google faculty research award (to AL). We thank Clara Vania, Sameer Bansal, Ida Szubert, Federico Fancellu, Antonis Anastasopoulos, Marco Damonte, and the anonymous reviews for helpful discussion of this work and comments on previous drafts of the paper.

References

Omri Abend and Ari Rappoport. 2013. Universal conceptual cognitive annotation (ucca). In *ACL (1)*. The Association for Computational Linguistics, pages 228–238. http://dblp.uni-trier.de/db/conf/acl/acl2013-1.html#AbendR13.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*. Association for Computational Linguistics, Sofia, Bulgaria, pages 178–186. http://www.aclweb.org/anthology/W13-2322.

Daniel Bauer and Owen Rambow. 2016. Hyperedge replacement and nonprojective dependency structures. In *Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12), June 29 - July 1, 2016, Heinrich Heine University, Düsseldorf, Germany*. pages 103–111. http://aclweb.org/anthology/W/W16/W16-3311.pdf.

Henrik Björklund, Frank Drewes, and Petter Ericson. 2016. *Between a Rock and a Hard Place – Uniform Parsing for Hyperedge Replacement DAG Grammars*, Springer International Publishing, Cham, pages 521–532. https://doi.org/10.1007/978-3-319-30000-9_40.

Johannes Blum and Frank Drewes. 2016. Properties of regular DAG languages. In *Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*. pages 427–438. https://doi.org/10.1007/978-3-319-30000-9_33.

David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. 2013. Parsing graphs with hyperedge replacement grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Sofia, Bulgaria, pages 924–932. http://www.aclweb.org/anthology/P13-1091.

Bruno Courcelle. 1991. The monadic second-order logic of graphs V: on closing the gap between definability and recognizability. *Theor. Comput. Sci.* 80(2):153–202. https://doi.org/10.1016/0304-3975(91)90387-H.

Bruno Courcelle and Joost Engelfriet. 2011. *Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach*. Cambridge University Press.

Frank Drewes. 1993. Np-completeness of k-connected hyperedge-replacement languages of order k. *Inf. Process. Lett.* 45(2):89–94. https://doi.org/10.1016/0020-0190(93)90221-T.

Frank Drewes, Berthold Hoffmann, and Mark Minas. 2015. *Predictive Top-Down Parsing for Hyperedge Replacement Grammars*, Springer International Publishing, Cham, pages 19–34. https://doi.org/10.1007/978-3-319-21145-9_2.

Frank Drewes, Hans-Jörg Kreowski, and Annegret Habel. 1997. Hyperedge replacement graph grammars. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific, pages 95–162.

Jay Earley. 1970. An efficient context-free parsing algorithm. ACM, New York, NY, USA, volume 13, pages 94–102. https://doi.org/10.1145/362007.362035.

Joost Engelfriet and Linda Heyker. 1991. The string generating power of context-free hypergraph grammars. *Journal of Computer and System Sciences* 43(2):328–360.

Dan Flickinger, Yi Zhang, and Valia Kordoni. 2012. Deepbank : a dynamically annotated treebank of the Wall Street Journal. In *Proceedings of the Eleventh International Workshop on Treebanks and Linguistic Theories (TLT11)*. Lisbon, pages 85–96. HU.

Jan Hajič, Eva Hajičová, Jarmila Panevov, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fučíková, Marie Mikulová, Petr Pajas, Jan Popelka, Jiří Semecký, Jana Šindlerová, Jan Štěpánek, Josef Toman, Zdeňka Urešová, and Zdeněk Žabokrtský. 2012. Announcing prague czech-english dependency treebank 2.0. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uur Doan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*. European Language Resources Association (ELRA), Istanbul, Turkey.

Bevan Jones, Jacob Andreas, Daniel Bauer, Karl Mortiz Hermann, and Kevin Knight. 2012. Semantics-based machine translation with hyperedge replacement grammars. In *Proceedings of COLING*.

Clemens Lautemann. 1990. The complexity of graph languages generated by hyperedge replacement. *Acta Informatica* 27(5):399–421. https://doi.org/10.1007/BF00289017.

Christoph Matheja, Christina Jansen, and Thomas Noll. 2015. *Tree-Like Grammars and Separation Logic*, Springer International Publishing, Cham, pages 90–108. https://doi.org/10.1007/978-3-319-26529-2_6.

David McAllester. 2002. On the complexity analysis of static analyses. *J. ACM* 49(4):512–537. https://doi.org/10.1145/581771.581774.

Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A synchronous hyperedge replacement grammar based approach for AMR parsing. In *Proceedings of the 19th Conference on Computational Natural Language Learning, CoNLL 2015, Beijing, China, July 30-31, 2015*. pages 32–41. http://aclweb.org/anthology/K/K15/K15-1004.pdf.

Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming* 24(1-2).

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, Stroudsburg, PA, USA, ACL '87, pages 104–111. https://doi.org/10.3115/981175.981190.