Between a Rock and a Hard Place – Parsing for Hyperedge Replacement DAG Grammars

Henrik Björklund, Frank Drewes, Petter Ericson

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

Abstract. We study the uniform membership problem for hyperedge-replacement grammars that generate directed acyclic graphs. The study of this type of language is motivated by applications in natural language processing. Our major result is a low-degree polynomial-time algorithm that solves the uniform membership problem for a restricted type of such grammars. We motivate the necessity of the restrictions by two different NP-completeness results.

1 Introduction

Hyperedge-replacement grammars (HRGs, see [7, 5]) are one of the most successful formal models for the generative specification of graph languages, thanks to the fact that their language-theoretic and algorithmic properties to a great extent resemble those of context-free grammars. Unfortunately, polynomial parsing is an exception from this general rule: graph languages generated by HRGs may be NP-complete. Thus, not only is the uniform membership problem intractable (unless P $\neq$ NP), but the non-uniform one is as well [1, 8].

Recently, Chiang et al. [4] advocated the use of hyperedge-replacement for describing meaning representations in natural language processing (NLP), and in particular the abstract meaning representations (AMRs) proposed by Banarescu et al. [2]. Chiang et al. described a general recognition algorithm building upon earlier work by Lautemann [9], together with a detailed complexity analysis. Unsurprisingly, the running time of the algorithm is exponential even in the non-uniform case, one of the exponents being the maximum degree of nodes in the input graph. Unfortunately, this is one of the parameters one would ideally not wish to limit, since AMRs may have unbounded node degree. However, AMRs and similar linguistic models to represent meaning are usually directed acyclic graphs (DAGs), a fact that is not exploited in [4]. Another recent approach to HRG parsing is [6], where predictive top-down parsing in the style of *SLL(1)* parsers is proposed. This is a uniform approach yielding parsers of quadratic running time in the size of the input graph, but the generation of the parser from the grammar is not guaranteed to run in polynomial time. (For a list of earlier attempts to HRG parsing, see [6].)

In this paper, we study the complexity of the membership problem for DAG-generating HRGs. Since NLP applications usually involve a machine learning component in which the rules of a grammar are inferred from a corpus, and hence the resulting HRG cannot be assumed to be given beforehand, we are mainly interested in efficient algorithms for the uniform membership problem. We propose restricted DAG-generating HRGs and show, in Section 4, that their uniform membership problem is solvable in polynomial time. More precisely, the upper bound on the running time of the algorithm is $\mathcal{O}(n^2+nm)$, where m and n are the sizes of the grammar and the input graph, resp. In linguistic applications, where grammars are usually much larger than the input structures to be parsed, this is essentially equivalent to $\mathcal{O}(nm)$. To our knowledge, this is the first time a uniform polynomial-time parsing algorithm for a non-trivial subclass of HRGs is proposed. Naturally, the restrictions are rather strong, but we shall briefly argue in Section 5 that they are reasonable in the context of AMRs. We furthermore motivate the restrictions with two NP-completeness results for DAG-generating HRGs, in Section 6. One of these proofs is a reduction of SAT to the uniform membership problem of DAG-generating HRGs whereas the second modifies the construction of [8] to show that there are NP-complete DAG languages of height 1 that can be generated by hyperedge replacement.

2 Preliminaries

The set of non-negative integers is denoted by $\mathbb{N}$. For $n \in \mathbb{N}$, $[n]$ denotes $\{1, \ldots, n\}$. Given a set S, let $S^{\circledast}$ be the set of non-repeating lists of elements of S. If $sw \in S^{\circledast}$ with $s \in S$, we shall also denote sw by (s, w). If $\preceq$ is a (partial) ordering of S, we say that $s_1 \cdots s_k \in S^{\circledast}$ *respects* $\preceq$ if $s_i \preceq s_j$ implies $i \leq j$.

2.1 Hypergraphs and DAGs

A *ranked alphabet* is a pair (Σ, rank) consisting of a finite set Σ of symbols and a *ranking function* $\mathrm{rank} : \Sigma \to \mathbb{N}$ which assigns a *rank* $\mathrm{rank}(a)$ to every symbol $a \in \Sigma$. We usually identify (Σ, rank) with Σ and keep the second component rank implicit.

Let Σ be a ranked alphabet. A (directed hyperedge-labeled) *hypergraph* over Σ is a tuple $G = (V, E, \mathrm{src}, \mathrm{tar}, \mathrm{lab})$ consisting of

- a finite set V of *nodes*,
- a *source* and *target mappings* $\mathrm{src}\colon E \to V$ and $\mathrm{tar}\colon E \to V^{\circledast}$ assigning to each hyperedge e its source $\mathrm{src}(e)$ and its sequence $\mathrm{tar}(e)$ of targets, and
- a *labeling* $\mathrm{lab}\colon E \to \Sigma$ such that $\mathrm{rank}(\mathrm{lab}(e)) = |\mathrm{tar}(e)|$ for every $e \in E$.

To simplify terminology, we shall in the following call hyperedges edges and hypergraphs graphs. Note that edges have only one source but several targets, similarly to the usual notion of term (hyper)graphs. The DAGs we shall consider below are, however, more general than term graphs in that nodes can have out-degree larger than one.

Continuing the formal definitions, a *path* in G is a (possibly empty) sequence $e_1, e_2, \ldots, e_k$ of edges such that for each $i \in [k-1]$ the source of e_{i+1} is a target of e_i. The *length* of a path is the number of edges it contains. A nonempty path is a *cycle* if the source of the first edge is a target of the last edge. If G does not contain any cycle then it is *acyclic* and is called a *DAG*. The *height* of a DAG G is the maximum length of any path in G. A node v is a *descendant* of a node u if $u = v$ or there is a nonempty path $e_1, \ldots, e_k$ in G such that $u = \mathrm{src}(e_1)$ and v occurs in $\mathrm{tar}(e_k)$. An edge e' is a *descendant edge* of an edge e if there is a path $e_1, \ldots, e_k$ in G such that $e_1 = e$ and $e_k = e'$.

The *in-degree* of a node $u \in V$ is the number of edges e such that u is a target of e. The *out-degree* of u is the number of edges e such that u is the source of e. A node with in-degree 0 is a *root* and a node with out-degree 0 is a *leaf*.

For a node u of a DAG $G = (V, E, \mathrm{src}, \mathrm{tar}, \mathrm{lab})$, the *sub-DAG rooted at u* is the DAG $G{\downarrow_u}$ induced by the descendants of u. Thus $G{\downarrow_u} = (U, E', \mathrm{src}', \mathrm{tar}', \mathrm{lab}')$ where U is the set of all descendants of u, $E' = \{e \in E \mid \mathrm{src}(e) \in U\}$, and src', tar', and lab' are the restrictions of src, tar and lab to E'. A leaf v of $G{\downarrow_u}$ is *reentrant* if there exists an edge $e \in E \setminus E'$ such that v occurs in $\mathrm{tar}(e)$.

2.2 DAG Grammars

A *marked* graph is a tuple $G = (V, E, \mathrm{src}, \mathrm{tar}, \mathrm{lab}, X)$ where $(V, E, \mathrm{src}, \mathrm{tar}, \mathrm{lab})$ is a graph and $X \in V^{\circledast}$ is nonempty. The sequence X is called the *marking* of G, and the nodes in X are referred to as *external nodes.* If $X = (v, w)$ for some $v \in V$ and $w \in V^{\circledast}$ then we denote them by $\mathrm{root}(G)$ and $\mathrm{ext}(G)$, resp. The former is motivated by the form or our rules, which is defined next.

Definition 1 (DAG grammar). *A* DAG grammar *is a system $H = (\Sigma, N, S, P)$ where Σ and N are disjoint ranked alphabets of* terminals *and* nonterminals*, respectively, S is the* starting nonterminal *with* $\mathrm{rank}(S) = 0$*, and P is a set of* productions*. Each production is of the form $A \to F$ where $A \in N$ and F is a marked DAG over $\Sigma \cup N$ with* $|\mathrm{ext}(F)| = \mathrm{rank}(A)$ *such that* $\mathrm{root}(F)$ *is the unique root of F and* $\mathrm{ext}(F)$ *contains only leaves of F.*

Naturally, a terminal (nonterminal) edge is an edge labeled by a terminal (nonterminal, resp.). We may sometimes just call them terminals and nonterminals if there is no danger of confusion. By convention, we use capital letters to denote nonterminals, and lowercase letters for terminal symbols.

A derivation step of H is described as follows. Let G be a graph with an edge e such that $\mathrm{lab}(e) = A$ and let $A \to F$ in P be a rule. Applying the rule involves replacing e with an unmarked copy of F in such a way that $\mathrm{src}(e)$ is identified with $\mathrm{root}(F)$ and for each $i \in [|\mathrm{tar}(e)|]$, the ith node in $\mathrm{tar}(e)$ is identified with the ith node in $\mathrm{ext}(F)$. Notice that $|\mathrm{tar}(e)| = |\mathrm{ext}(F)|$ by definition. If the resulting graph is G', we write $G \Rightarrow_H G'$. We write $G \Rightarrow_H^* G'$ if G' can be derived from G in zero or more derivation steps. The *language* $\mathcal{L}(H)$ of H are all graphs G over the terminal alphabet T such that $S^\bullet \Rightarrow_H^* G$ where $S^\bullet$ is the graph consisting of a single node and a single edge labeled by S.

The graphs produced by DAG grammars are connected, single-rooted, and as the name implies, acyclic. This can be proved in a straightforward manner by induction on the length of the derivation.

2.3 Ordering the Leaves of a DAG

Let $G = (V, E, \mathrm{src}, \mathrm{tar}, \mathrm{lab})$ be a DAG and let u and u' be leaves of G. We say that an edge e with $\mathrm{tar}(e) = w$ is a *common ancestor edge* of u and u' if there are t and t' in w such that u is a descendant of t and u' is a descendant of t'. If, in addition, there is no edge with its source in w that is a common ancestor edge of u and u', we say that e is a *closest* common ancestor edge of u and u'. We stress that since a node is a descendant of itself, this definition implies that if u and u' belong to w, then e is a closest common ancestor edge of u and u'. We also note that in a DAG, a pair of nodes can have more than one closest common ancestor edge.

Definition 2. *Let $G = (V, E, \mathrm{src}, \mathrm{tar}, \mathrm{lab})$ be a DAG. Then $\preceq_G$ is the partial order on the leaves of G defined by $u \preceq_G u'$ if, for every closest common ancestor edge e of u and u', $\mathrm{tar}(e)$ can be written as wtw' such that t is an ancestor of u and all ancestors of u' in $\mathrm{tar}(e)$ are in w'.*

3 Restricted DAG Grammars

DAG grammars are a special case of hyperedge-replacement grammars. We now define further restrictions that will allow polynomial time uniform parsing.

Every rule $A \to F$ of a *restricted DAG grammar* is required to satisfy the following conditions (in addition to the conditions formulated in Definition 1):

1. If a node v of F has in-degree larger than one, then v is a leaf
2. If F consists of exactly two edges e_1 and e_2, both labeled by A, such that $\mathrm{src}(e_1) = \mathrm{src}(e_2)$ and $\mathrm{tar}(e_1) = \mathrm{tar}(e_2)$ we call $A \to F$ a *clone rule.* Clone rules are the only rules in which a node can have out-degree larger than 1 and the only rules in which a nonterminal can have the root as its source.
3. For every nonterminal e in F, all nodes in $\mathrm{tar}(e)$ are leaves.
4. If a leaf of F has in-degree exactly one, then it is an external node or its unique incoming edge is terminal.
5. The leaves of F are totally ordered by $\preceq_F$ and $\mathrm{ext}(F)$ respects $\preceq_F$.

As is the case for DAG grammars in general, every graph that can be derived by a restricted DAG grammar is connected, single-rooted, and acyclic. We now demonstrate some additional properties.

Lemma 1. *Let $H = (\Sigma, N, S, P)$ be a restricted DAG grammar, G a DAG such that $S^\bullet \Rightarrow_H^* G$, and U the set of nodes of in-degree larger than 1 in G. Then U contains only leaves of G and $\mathrm{tar}(e) \in U^\circledast$ for every nonterminal e of G.*

Proof. We prove the lemma by induction. The base case, where $G = S^\bullet$ is immediate. Assume that G fulfils the conditions of the lemma and consider G' such that $G \Rightarrow_H G'$. Let $A \to F$ be the rule used in the derivation step.

By assumption, the edge e, labeled by A, that is rewritten has only leaves as targets. As nonterminals in F only appear directly above leaves in F and all the nodes in the marking of F are leaves, nonterminals of G' only appear directly above leaves.

Since only leaves have in-degree larger than 1 in G, all targets of A are leaves, and only leaves have in-degree larger than 1 in F, only leaves have in-degree larger than 1 in G'.

Since the edge that is being rewritten is nonterminal, it is not connected to any leaf with in-degree exactly 1. In F, leaves with in-degree exactly 1 are only connected to terminals. Thus the same holds in G'. □

3.1 Normal form

To simplify the presentation of parsing algorithm, we introduce a normal form for restricted DAG grammars.

Definition 3. *A restricted DAG grammar $H = (\Sigma, N, S, P)$ is on* normal form *if every rule $A \to F$ in P has one of the following three forms.*

(a) The rule is a clone rule.
(b) F has a single edge e, which is terminal.
(c) F has height 2, the unique edge e with $\mathrm{src}(e) = \mathrm{root}(F)$ is terminal, and all other edges are nonterminal.

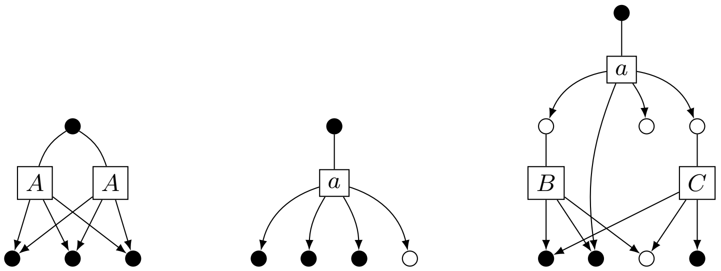

Fig. 1. Examples right-hand sides F of normal form rules of types (a), (b), and (c) for a nonterminal of rank 3. In illustrations such as these, boxes represent hyperedges e, where $\mathrm{src}(e)$ is indicated by a line and the nodes in $\mathrm{tar}(e)$ by arrows. Filled nodes represent the marking of F. Both $\mathrm{tar}(e)$ and $\mathrm{ext}(F)$ are drawn from left to right unless otherwise indicated by numbers.

See Figure 1 for examples of right-hand sides of the three types. In particular, right-hand sides F of the third type consist of nodes $v, v_1, \ldots, v_m, u_1, \ldots, u_n$, a terminal edge e and nonterminal edges $e_1, \ldots, e_k$ such that

- $v = \mathrm{root}(F) = \mathrm{src}(e)$ and $v_1 \cdots v_m$ is a subsequence of $\mathrm{tar}(e)$,
- $\mathrm{src}(e_i) \in \{v_1, \ldots, v_m\}$ for all $i \in [k]$,
- $\mathrm{ext}(F)$ and $\mathrm{tar}(e_i)$, for $i \in [k]$, are subsequences of $u_1 \cdots u_n$.

Lemma 2. *Every restricted DAG grammar H can be transformed in linear time into a restricted DAG grammar H' on normal form such that $\mathcal{L}(H) = \mathcal{L}(H')$.*

Proof. Let $H = (\Sigma, N, S, P)$ and let $r = A \to F$ be a rule in P. We present a recursive procedure for replacing r with a number of rules who together can derive F from A. If F has height 1, then due to restriction 2, r already has form (a) or (b). Thus, nothing needs to be done. Otherwise, we know that F has height at least 2 and, again by restriction 2, a unique edge e such that $\mathrm{src}(e) = \mathrm{root}(F)$. By the height of F, and since only leaves are targets of nonterminals, e is terminal.

Now, assume that F does not have the form (c). Then there exists a node v' in $\mathrm{tar}(e)$ which is not a leaf, such that the unique outgoing edge of v' is terminal. Let $F' = F{\downarrow_{v'}}$ and let s be the sequence of leaves in F, ordered according to $\preceq_F$. Notice that since no node in F has out-degree larger than 1, the leaves are totally ordered by $\preceq_F$, and $\mathrm{ext}(F)$ is a subsequence of s. Now, let s' be the subsequence of s consisting of the leaves in F' that are either in $\mathrm{ext}(F)$ or in $\mathrm{tar}(e')$ for an edge e' in F that does not belong to F'. We create a fresh nonterminal A' with $\mathrm{rank}(A') = |s'|$ and a rule $r' = A' \to (F', v's')$, i.e., the marking of the right-hand side is (v', s'). In F, we replace F' by A'. (More precisely, we remove all edges in F' from F, and likewise all nodes F' except for those in $v's'$, and we add a fresh edge f with $\mathrm{src}(f) = v'$, $\mathrm{tar}(f) = s'$, and $\mathrm{lab}(f) = A'$.)

Clearly, the language generated by the grammar is not affected by this decomposition of r into two rules. Moreover, each of the two new right-hand sides satisfies the conditions 1–5 and has fewer terminal hyperedges than F. Hence, by repeating the process we finally obtain an equivalent restricted DAG grammar in normal form. □

Lemma 3. *Let H be a restricted DAG grammar and $G = (V, E, \mathrm{src}, \mathrm{tar}, \mathrm{lab})$ a DAG generated by H. Then there is a total order $\trianglelefteq$ on the leaves of G such that $\preceq_G \subseteq \trianglelefteq$ and for every $v \in V$ and every pair u, u' of reentrant nodes of $G{\downarrow_v}$ we have $u \trianglelefteq u' \Leftrightarrow u \preceq_{G\downarrow_v} u'$.*

Proof. Note that it suffices to consider nodes v that are not leaves since the statement is trivially true if v is a leaf. Without loss of generality, we may furthermore assume that H is in normal form. We show by induction on the length of derivations that the statement holds for all DAGs G that can be derived from $S^\bullet$, not just the terminal ones. Moreover, we shall additionally prove that $\trianglelefteq$ can be chosen in such a way that $u_1 \trianglelefteq \cdots \trianglelefteq u_k$ for all nonterminals e in G with $\mathrm{tar}(e) = u_1 \cdots u_k$.

The DAG $S^\bullet$ has the claimed property as it does not possess any leaves. Now, consider a derivation $S^\bullet \Rightarrow^n G_0 \Rightarrow G$ and assume that the claim holds for G_0 with the total order $\trianglelefteq_0$. Let G be obtained from G_0 by applying a rule $r = A \to F$ to an edge e in G_0. There are three different cases to consider.

If the rule r is a clone rule, setting $\trianglelefteq = \trianglelefteq_0$ is sufficient because $\preceq_{G\downarrow_v} = \preceq_{G_0\downarrow_v}$ for all nodes v. This follows directly from the fact that the two edges e_1, e_2 that e is replaced with satisfy $\mathrm{tar}(e_1) = \mathrm{tar}(e) = \mathrm{tar}(e_2)$.

If r is of the form (b), let $\trianglelefteq$ be any total extension of $\trianglelefteq_0$ to the set of leaves of G that is consistent with $\preceq_F$. For all v, the reentrant nodes of $G\downarrow_v$ coincide with those of $G_0\downarrow_v$, and by restriction 5, $\preceq_{G\downarrow_v}$ coincides with $\preceq_{G_0\downarrow_v}$ on these nodes.

Finally, suppose r is of the form (c) and let $\mathrm{ext}(F) = v_1 \cdots v_k$. Leaves of F that are not in $\{v_1, \dots, v_k\}$ and have in-degree 1 are not reentrant in any $G\downarrow_v$ and can thus be handled as in the preceding case, i.e., $\trianglelefteq_0$ can be extended to cover these nodes in any way that is consistent with $\preceq_F$. Let U be the remaining set of leaves of F, which thus includes $\{v_1, \dots, v_k\}$. Since the nodes of F have out-degree at most one, U is totally ordered by $\preceq_F$, and by restriction 5 we have $v_1 \preceq_F \cdots \preceq_F v_k$. Moreover, by the induction hypothesis we may assume that $v_1 \trianglelefteq_0 \cdots \trianglelefteq_0 v_k$. We can thus extend $\trianglelefteq_0$ to a total order $\trianglelefteq$ on the leaves of G in such a way that the order coincides with $\preceq_F$ on U. It remains to argue that this definition of $\trianglelefteq$ has the claimed property.

To this end, let v be a non-leaf of G and let u, u' be reentrant nodes of $G\downarrow_v$. If v is a node in G_0 then u, u' are leaves of G_0 and we have

$$u \preceq_{G\downarrow_v} u' \Rightarrow u \preceq_{G_0\downarrow_v} u' \Rightarrow u \trianglelefteq_0 u' \Rightarrow u \trianglelefteq u'. \tag{1}$$

The remaining case is the one in which v is the source of a nonterminal edge f of F and u, u' are targets of f. If not both of u, u' are in $\mathrm{ext}(F)$ then f is the only closest common ancestor edge of u and u', and thus the claim immediately follows. If both u and u' are in $\mathrm{ext}(F)$ and u occurs before u' in $\mathrm{tar}(f)$, then $u \preceq_F u'$ by restriction 5. Consequently, $u \preceq_{G\downarrow_v} u'$ and also $u \preceq_{G_0\downarrow_v} u'$ because the only closest common ancestor of u and u' in G_0 that is not a closest common ancestor of them in G is f. Moreover, both u and u' are targets of f in G_0, so that the induction hypothesis yields $u \trianglelefteq_0 u'$. Altogether, we obtain the same chain of implications as in (1) above. □

3.2 Derivation Transparency

If a DAG G has been derived by a restricted DAG grammar in normal form, it is uniquely determined which subgraphs of G have been produced by a nonterminal, and which leaves were connected to it at that point. In particular, given a non-leaf node v in G, consider the subgraph $G\downarrow_v$. Consider the earliest point in the derivation where there was a nonterminal e having v as its source. We say that e generated $G\downarrow_v$. From the structure of G and $G\downarrow_v$, we know that all reentrant nodes of $G\downarrow_v$ are leaves and, by restriction 4, that e must have had exactly these reentrant leaves of $G\downarrow_v$ as targets. By Lemma 3 and restriction 5, the order of these leaves in $\mathrm{tar}(e)$ coincides with the total order $\preceq_{G\downarrow_v}$.

In other words, during the generation of G by a restricted DAG grammar, $G\downarrow_v$ must be generated from a nonterminal e such that $\mathrm{src}(e) = v$ and $\mathrm{tar}(e)$ is uniquely determined by the condition that it consists of exactly the reentrant nodes of $G{\downarrow_v}$ and respects $\preceq_{G{\downarrow_v}}$. Therefore, we will from now on view $G{\downarrow_v}$ as a *marked* DAG, where the marking is $(v, \mathrm{tar}(e))$.

4 A Polynomial Time Algorithm

We present the parsing algorithm in pseudocode, after which we explain various subfunctions used therein. Intuitively, we work bottom-up on the graph in a manner resembling bottom-up finite-state tree automata, apart from where a node has out-degree greater than one. We assume that a total order $\trianglelefteq$ on the leaves of the input DAG G, as ensured by Lemma 3, is computed in a preprocessing step before the algorithm is executed. At the same time, the sequence w_v of external nodes of each sub-DAG $G{\downarrow_v}$ is computed. (Recall from the paragraph above that these are the reentrant leaves of $G{\downarrow_v}$, ordered according to $\preceq_{G{\downarrow_v}}$.) For a DAG G of size n, this can be done in time $O(n^2)$ by a bottom-up process. To explain how, let us denote the set of all leaves of $G{\downarrow_v}$ by U_v for every node v of G. We proceed as follows. For a leaf v, let $\trianglelefteq_v = \{(v, v)\}$ and $w_v = v$. For every edge e with $\mathrm{tar}(e) = u_1 \dots u_k$ such that u_i has already been processed for all $i \in [k]$, first check if $\trianglelefteq_0 = \bigcup_{i \in [k]} \trianglelefteq_{u_i}$ is a partial order. If so, define $\trianglelefteq_e$ to be the unique extension of $\trianglelefteq_0$ given as follows. Consider two nodes $u, u' \in U_{\mathrm{src}(e)}$ that are not ordered by $\trianglelefteq_0$. If i, j are the smallest indices such that $u \in U_{u_i}$ and $u' \in U_{u_j}$, then $u \trianglelefteq_e u'$ if $i < j$. Note that $\trianglelefteq_e$ is uniquely determined and total. Moreover, let w_e be the unique sequence in $U^{\circledast}_{\mathrm{src}(e)}$ which respects $\trianglelefteq_0$ and contains exactly the nodes in $U_{\mathrm{src}(e)}$ which are targets of edges of which e is not an ancestor edge. Similarly, if v is a node and all edges $e_1, \dots, e_k$ having v as their source have already been processed, check if $\bigcup_{i \in [k]} \trianglelefteq_{e_i}$ is a partial order. If so, define $\trianglelefteq_e$ to be any total extension of this order. Moreover, check that $w_{e_1} = \dots = w_{e_k}$, and let w_v be exactly this sequence.

After this preprocessing, Algorithm 1 can be run. As the sequences w_u of external nodes for each sub-DAG $G{\downarrow_u}$ were computed in the preprocessing step, we consider this information to be readily available in the pseudocode. This, together with the assumption that the DAG grammar H is in normal form allows for much simplification of the algorithm.

Walking through the algorithm step by step, we first extract the root node (line 2) and determine which kind of (sub-)graph we are dealing with (line 4): one with multiple outgoing edges from the root must have been produced by a cloning rule to be valid, meaning we can parse each constituent subgraph (line 5) recursively (line 6) and take the intersection of the resulting nonterminal edges (line 7). Each nonterminal that could have produced all the parsed subgraphs and has a cloning rule is entered into *returns* (line 8). The procedure `subgraphs_below` is used to partition the sub-DAG $G{\downarrow_v}$ into one sub-DAG per edge having v as its source, by taking each such edge and all its descendant edges (and all their source and target nodes) as the subgraph. Note that the order among these subgraphs is undefined, though they are all guaranteed by the preprocessing to have the same sequence of external nodes w_v.

Algorithm 1 Parsing of restricted graph grammars

```
 1: function PARSES_TO(restricted DAG grammar $H$ in normal form, DAG $G$)
 2:     $v \leftarrow \texttt{root}(G)$
 3:     $\mathit{returns} \leftarrow \emptyset$
 4:     if $\texttt{out\_degree}(v) > 1$ then
 5:         for $G_i \leftarrow \texttt{subgraphs\_below}(v)$ do
 6:             $N_i \leftarrow \texttt{parses\_to}(G_i)$
 7:         $N \leftarrow \bigcap_i N_i$
 8:         $\mathit{returns} \leftarrow \{A \in N \mid \texttt{has\_clone\_rule}(A)\}$
 9:     else
10:         $e \leftarrow \texttt{edge\_below}(v)$
11:         $\mathit{children} \leftarrow ()$
12:         for $v' \leftarrow \texttt{targets}(e)$ do
13:             if $\texttt{leaf}(v')$ then
14:                 $\texttt{append}(\mathit{children}, \texttt{external\_node}(v'))$
15:             else
16:                 $\texttt{append}(\mathit{children}, \texttt{parses\_to}(G{\downarrow}_{v'}))$
17:         $\mathit{returns} \leftarrow \{A \mid (A \to F) \in P \text{ and } \texttt{match}(F, e, \mathit{children})\}$
18:     return $\mathit{returns}$
```

If, on the other hand, we have a single outgoing edge from the root node (line 9), we iterate through the subgraphs below the (unique) edge below the root node (line 12). Nodes are marked either with a set of nonterminals (that the subgraph below the nodes can parse to) (line 16), or, if the node is a leaf, with a boolean indicating whether or not the node is reentrant in the currently processed subgraph G (line 14).

The `match` function used in line 17 deserves a closer description, as much of the complexity calculations depend on this function taking no more than time linear in the size of the right-hand side graph on average. It works as follows:

Let $\mathrm{src}(e) = v$ and $\mathrm{tar}(e) = v_1 \cdots v_k$. Each v_i has an entry in *children*. If v_i is a leaf it is a Boolean, otherwise a set of nonterminal labels. From G and *children*, we create a DAG G' as follows. Let T be the union of $\{v, v_1, \ldots, v_k\}$ and the set of leaves ℓ of G such that ℓ is reentrant to G (as indicated by *children*) or there is an $i \in [k]$ with ℓ being external in $G{\downarrow}_{v_i}$. Let $T = \{v, v_1, \ldots, v_k, t_1, \ldots, t_p\}$. Then G' has the set of nodes $U = \{u, u_1, \ldots, u_k, s_1, \ldots, s_p\}$. Let h be the bijective mapping with $h(v) = u$ and $h(v_i) = u_i$ for every $i \in [k]$ and $h(t_i) = (s_i)$ for every $i \in [p]$. We extend h to sequences in the obvious way. The root of G is u and there is a single edge d connected to it such that $\mathrm{lab}(d) = \mathit{lab}(e)$, $\mathrm{src}(d) = u$ and $\mathrm{tar}(d) = u_1 \cdots u_k$. For every $i \in [k]$ such that v_i is not a leaf, G' has an edge d_i with $\mathrm{src}(d_i) = u_i$ and $\mathrm{tar}(d_i) = h(w_i)$, where w_i is the subsequence of leaves of $G{\downarrow}_{v_i}$ that belong to T, ordered by $\trianglelefteq$. The edge is labeled by the *set* of nonterminals $\mathit{children}[i]$.

Once `match` has built G' it tests whether there is a way of selecting exactly one label for each nonterminal edge in G' such that the resulting graph is isomorphic to *rhs*. This can be done in linear time since the leaves of both G' and *rhs* are totally ordered and, furthermore, the ordering on $v_1 \cdots v_k$ and $u_1 \cdots u_k$ makes the matching unambiguous.

Let us now discuss the running time of Algorithm 1.

Entering the `if` branch of `parses_to`, we simply recurse into each subgraph and continue parsing. The actual computation in the `if`-clause is minor: an intersection of the l sets of nonterminals found.

Each time we reach the `else` clause in `parses_to`, we consume one terminal edge of the input graph. We recurse once for each terminal edge below this (no backtracking), so the parsing itself enters the `else`-clause n times, where n is the number of terminal edges in the input graph. For each rule $r = A \rightarrow F$, we build and compare at most $|F|$ nodes or edges in the `match` function. Thus, it takes $\mathcal{O}(nm)$ operations to execute Algorithm 1 in order to parse a graph with n terminal hyperedges according to a restricted DAG grammar H in normal form of size m. If H is not in normal form, Lemma 2 can be used to normalize it in linear time. Since the process does not affect the size of H by more than a (small) linear factor, the time bound is not affected. Finally, a very generous estimation of the running time of the preprocessing stage yields a bound of $\mathcal{O}(n^2)$, because n edges (and at most as many nodes) have to be processed, each one taking no more than n steps. Altogether, we have shown the following theorem, the main result of this paper.

Theorem 1. *The uniform membership problem for restricted DAG grammars is solvable in time $\mathcal{O}(n^2 + mn)$, where n is the size of the input graph and m is the size of the grammar.*

Note that in linguistic applications grammars are usually by orders of magnitude larger than the structures to be parsed (sentences, trees or, in our case, DAGs). Therefore, the bound given in Theorem 1 is essentially $\mathcal{O}(mn)$ in the context of such applications.

5 Representing and Generating AMRs

Let us have a very short glimpse at Abstract Meaning Representations (AMRs) and compare them with the type of DAGs considered in this paper. An AMR is an ordinary directed edge-labeled acyclic graph expressing the meaning of a sentence. An example expressing *"Anna's cat is missing her"* is shown in Figure 2. The root corresponds to the concept "missing", which takes two arguments, the misser and the missed.

In this representation every node has a special "instance edge" that determines the concept represented by its source node (miss, cat, anna). The most important concepts are connected to (specific meanings of) verbs, which have a number of mandatory arguments arg0, arg1, ... whose number depends on the concept in question. While the representation shown is not directly compatible with the restrictions introduced in Section 3 a simple translation helps. Every concept with its k mandatory arguments is turned into a hyperedge of rank $k + 1$, the target nodes of which represent the instance (a leaf) and the roots

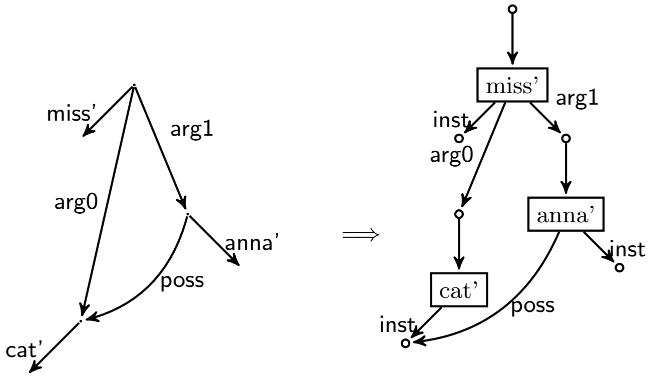

Fig. 2. Example translation of AMR.

of the arguments. The resulting hypergraph is shown in Figure 2 on the right. Note that all shared nodes on the left (corresponding to cross-references) are turned into reentrant leaves. This is important because in a DAG generated by a restricted DAG grammar only leaves can have an in-degree greater than 1.

It might seem that we only need graphs with nodes of out-degree at most 1, and thus no cloning rules for their generation. However, a concept such as miss can typically also have optional so-called modifiers, such as in *"Anna's cat is missing her heavily today"*, not illustrated in the figure. Such modifiers can typically occur in any number. We can add them to the structure by increasing the rank of miss by 1, thus providing the edge with another target v. The out-degree of this node v would be the number of modifiers of miss. Using the notation of Section 4, each sub-DAG $G{\downarrow_e}$ given by one of the outgoing edges e of v would represent one (perhaps complex) modifier. To generate these sub-DAGs $G{\downarrow_e}$ a restricted DAG grammar would use a nonterminal edge that has v as its source and which can be cloned. The latter makes it possible to generate any number of modifiers all of which can refer to the same shared concepts (represented by the leaves having the cloned nonterminals as their common targets).

On the generating side of AMRs, we immediately run into problems if the situation calls for multi-rooted graphs (e.g. two sentences connected via a conjunction or similar). Furthermore, the standard AMR solution for this situation (introducing a "dummy" root node, which connects to all the individual roots) is not necessarily applicable, as there might still be calls for connections among the different parts of the graph, which is a situation that cannot be covered by restricted DAG grammars. However, introducing a dummy *edge* above the different parts lets us decide on an order, and generate all the shared nodes beforehand, so to speak.

In Figure 3 we present a restricted DAG grammar that generates AMR-like graphs for all sentences consisting only of the concepts *boy*, *girl*, *want*, and *believe* in various combinations, an example that was introduced in [3]. Note that there is only *one* boy and girl involved, which requires us to use a "dummy" root creating them (in order not to have several copies), along with the various sub-sentence start symbols.

The first row of rules constructs the basic structure of the graph – one edge each for *boy* and *girl*, and three basic statement edges. Any of these statement edges may be omitted, though we do not show these permutations in Figure 3. The second, third and fourth row are fairly self-explanatory. The rules for V_2

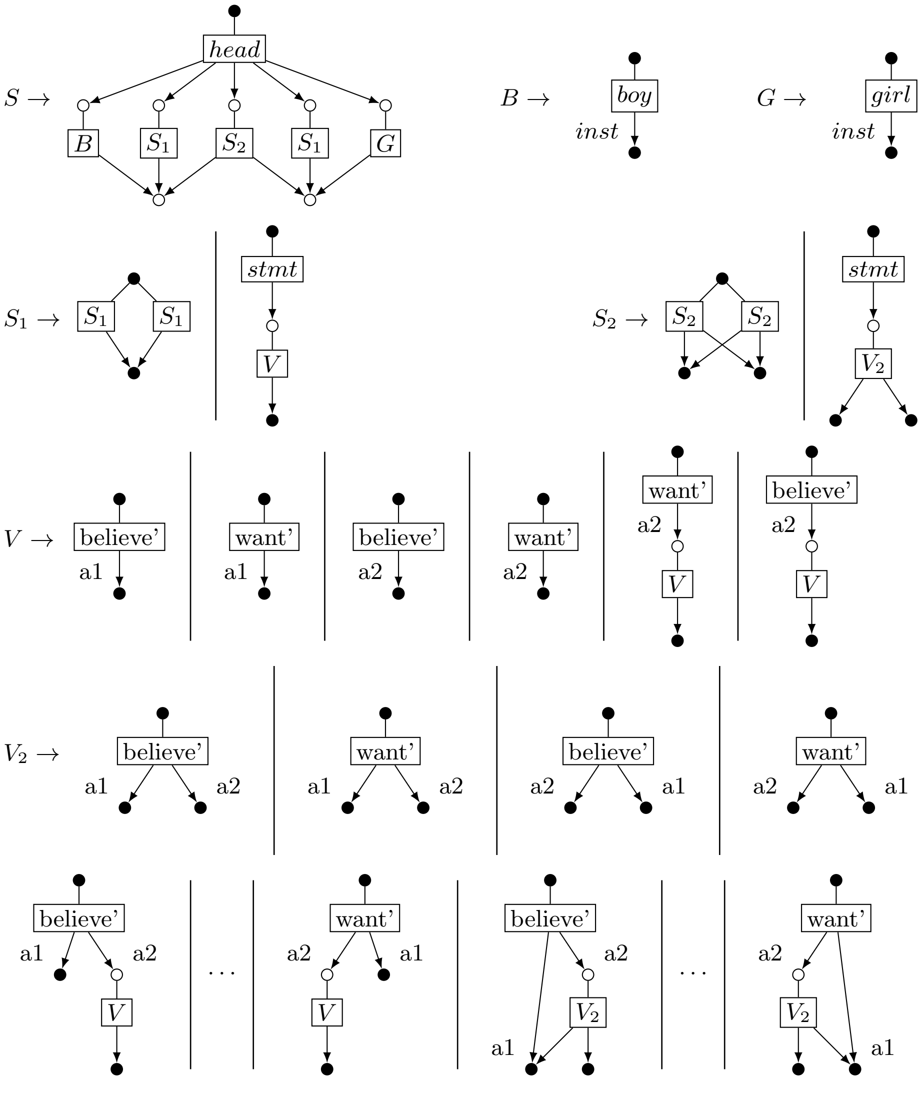

Fig. 3. Rules for a restricted DAG grammar generating AMR-like graphs for all sentences involving *boy*, *girl*, *want* and *believe*

involve quite a bit of (omitted) repetition. In particular, the first ellipsis cover two right-hand side graphs, the second another two.

Though the graph grammar is somewhat cumbersome, it serves as an example of a restricted DAG grammar generating a very general language of AMR-like graphs.

6 NP-hardness Results

In order to motivate the rather harsh restrictions we impose on our grammars, we present NP-hardness results for two different classes of grammars that are obtained by easing the restrictions in different ways.

Theorem 2. *The uniform membership problem for DAG grammars that conform to restrictions 1–4 is NP-complete.*

Proof. Clearly, the problem is in NP since the restrictions guarantee that derivations are of linear length in the size of the input graph. Thus, it remains to prove NP-hardness.

Let us consider an instance φ of the satisfiability problem SAT, i.e., a set $\{C_1, \ldots, C_m\}$ of clauses C_i, each being a set of literals x_j or $\neg x_j$, where $j \in [n]$ for some $m, n \in \mathbb{N}$. Recall that the question asked is whether there is an assignment of truth values to the variables x_j such that each clause contains a true literal. We have to show how to construct a DAG grammar H and an input graph G such that $G \in L(H)$ if and only if φ is satisfiable.

For simplicity, we shall first give a construction in which H violates conditions 4 and 5. The grammar uses nonterminals S, K, K_i, K_{ij} with $i \in [m], j \in [n]$. The terminal labels are c, all $j \in [m]$, and an "invisible" label. The labels K, K_i, K_{ij}, c are of rank $2n$, S is of rank 0 and the remaining ones are of rank 1. Figure 4 depicts the rules of the grammar. In this figure, and in the following, we draw ordinary edges (i.e., whose labels have rank 1) in the usual form as labeled arcs rather than boxes.

The grammar works in the following stages.

First row of rules: (1) Generate $2n$ leaves which, intuitively, represent $x_1, \neg x_1$, $\ldots, x_n, \neg x_n$ and are targets of a K-labeled nonterminal. (2) Clone K any number of times (where the intention is to clone it m times, once for each clause). (3) Let each K "guess" which clause C_i ($i \in \mathbb{N}$) it should check.

Second row of rules: (4) Let every K_i "guess" which literal makes C_i true. If the literal is negative, interchange the corresponding targets, otherwise keep their order.

Third row of rules: (5) For all pairs $(x_\ell, \neg x_\ell)$ that are not used to satisfy C_i, interchange the corresponding targets or keep their order. Finally, (6) replace the nonterminal edge by a terminal one.

Now, consider the input DAG G in Figure 5 (left). Suppose that G is indeed generated by H. Since the jth outgoing tentacles of all c-labeled edges point to the same node (representing either x_j or $\neg x_j$), a consistent assignment is

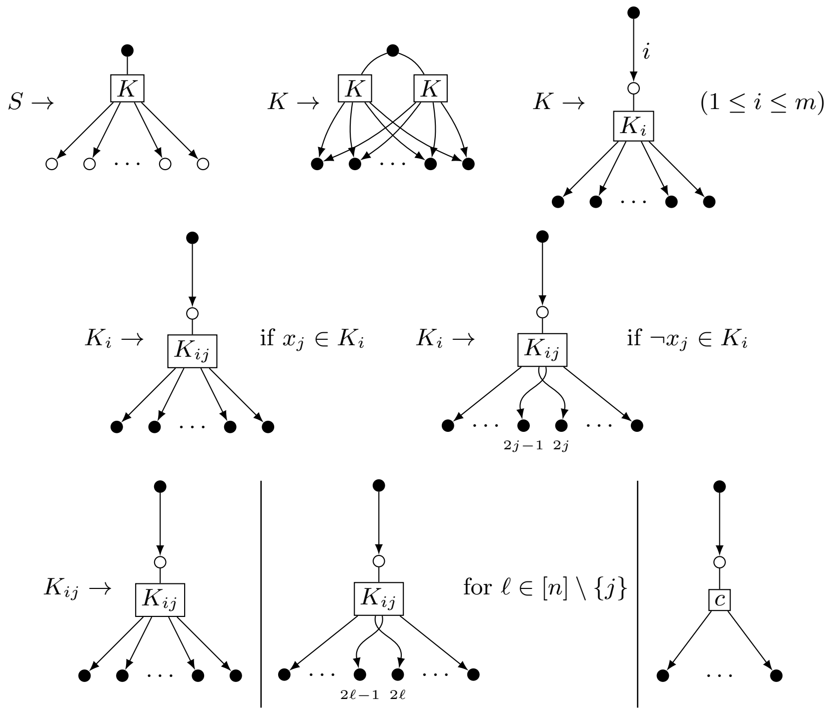

Fig. 4. Reduction of SAT to the uniform membership problem

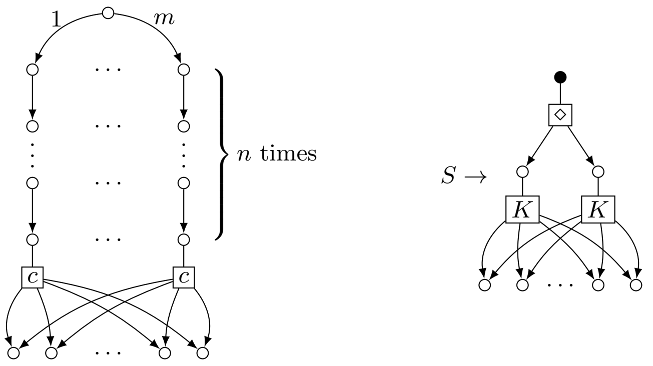

Fig. 5. Input graph in the proof of Theorem 2 (left) and modified starting rule (right)

obtained that satisfies φ. Conversely, a consistent assignment obviously gives rise to a corresponding derivation of G, thus showing that the reduction is correct.

Finally, let us note that changing the initial rule to the one shown in the left part of Figure 5 (using a new terminal $\diamond$ of rank 2) makes H satisfy condition 4 as well. This change being made, the input graph is changed by including two copies of the original input, both sharing their leaves, and adding a new root with an outgoing $\diamond$-hyperedge targeting the roots of the two copies. □

Let us now turn to our second NP-completeness result. It shows that if we, in addition, disregard restriction 2 in the definition of restricted DAG grammars, even the non-uniform membership problem becomes NP-complete. Moreover, this result holds even if all graphs generated by the grammar have height 1.

Theorem 3. *There is a DAG grammar H that conforms to restrictions 1, 3, and 4, such that all graphs in $\mathcal{L}(H)$ have height* 1 *and $\mathcal{L}(H)$ is NP-complete.*

Proof. The proof is by reduction from the (non-uniform) membership problem for *context-free grammars with disconnecting* (CFGD), using a result from [8]. A CFGD is an ordinary context-free grammar G in Chomsky normal form, with additional rules $A \to \diamond$, where $\diamond$ is a special symbol that cuts the string apart. Thus, an element in the generated language is a finite multiset of strings rather than a single string. More precisely, let $w = w_1 \diamond \cdots \diamond w_k \in (\Sigma \cup \{\diamond\})^*$, with $w_1, \ldots, w_k \in \Sigma^*$, be a string generated by G if we view G as an ordinary context-free grammar over $\Sigma \cup \{\diamond\}$. Then the multiset $\{w_1, \ldots, w_k\}$ is in $\mathcal{L}(G)$.

It is shown in [8] that CFGDs can generate NP-complete languages. Now, let us represent a multiset $\{w_1, \ldots, w_k\}$ of strings w_i as a graph consisting of k DAGs of height 1 sharing their roots, as follows. For a single string $w_i = a_1 \cdots a_m$, the graph $\mathrm{dag}(w_i)$ representing it consists of a root v, leaves $u_0, \ldots, u_m$, and a_i-hyperedges e_i with $\mathrm{src}(e_i) = r$ and $\mathrm{tar}(e_i) = u_{i-1}u_i$. Moreover, there are two terminal edges from v to u_0 and u_n, resp. (We draw the latter as unlabeled edges, using a special "invisible" label.) For a finite multiset $W = \{w_1, \ldots, w_k\}$ of strings w_i, $\mathrm{dag}(W)$ is obtained from the disjoint union of the individual DAGs $\mathrm{dag}(w_i)$ by identifying their roots. As an example, $\mathrm{dag}(\{ab, aba, c\})$ is shown in Figure 6.

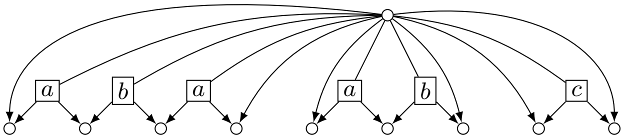

Fig. 6. The DAG $\mathrm{dag}(\{ab, aba, c\})$; note that the DAG does not define an order among the sub-DAGs $\mathrm{dag}(w_i)$ that constitute it

Now, every CFGD G can be turned into a DAG grammar H such that $\mathcal{L}(H) = \{\text{dag}(W) \mid W \in \mathcal{L}(G)\}$ using the schemata in Figure 7. Hence, $\mathcal{L}(H)$ is NP-complete if $\mathcal{L}(G)$ is.

Though the simplicity of the translation should be sufficient to prove its correctness, a few remarks may be in order. On the one hand, the ordering of symbols within the representation of an individual string w_i is faithfully reflected in $\text{dag}(w_i)$, due to the fact that $\text{tar}(e)$ is an ordered sequence for each edge e, which unambiguously determines the start and end of the representation of w_i. On the other hand, there is no order among the represented strings in $\text{dag}(W)$ as they are connected *only* via the root.

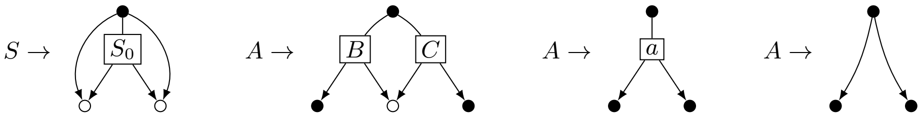

Fig. 7. Rules of a DAG grammar equivalent to a CFGD with initial nonterminal S_0, from left to right: initial rule, $A \rightarrow BC$, $A \rightarrow a$, $A \rightarrow \diamond$.

7 Conclusions

By enforcing rather severe restrictions, we have defined a class of hyperedge replacement graph grammars for which even the uniform parsing problem is solvable in low-degree polynomial time. We also argued that this class, despite its limitations, can still be practically relevant, e.g., in linguistic applications.

A number of interesting questions remain open. We motivate our restrictions by showing how two ways of easing them lead to NP-hardness, but this does not necessarily mean that all of our restrictions are necessary, neither does it mean that they are the only interesting ones. Is it the case that lifting any one of our five restrictions, while keeping the others, leads to NP-hardness? It seems that the algorithm we propose leads to a fixed-parameter tractable algorithm, with the size of right-hand sides in the grammar as the parameter, when we lift restriction 5 (enforcing that the marking respects $\preceq_F$). Is this actually the case and are there other interesting parameterizations that give tractability for some less restricted classes of grammars? Another open question is whether the algorithm for checking the structure of the input graph and computing the ordering on the leaves can be optimized to run in linear or $\mathcal{O}(n \log n)$ time.

From a practical point of view, one should study in detail how well suited restricted DAG grammars are for describing linguistic structures such as AMRs. Which phenomena can be modeled in an appropriate manner and which cannot? Are there important aspects in AMRs that can be modeled by general DAG-generating HRGs but not by restricted DAG grammars? If so, can the restrictions be weakened appropriately without sacrificing polynomial parsability?

References

1. I. J. Aalbersberg, A. Ehrenfeucht, and G. Rozenberg. On the membership problem for regular DNLC grammars. *Discrete Applied Mathematics*, 13:79–85, 1986.
2. L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider. Abstract meaning representation for sembanking. In *Proc. 7th Linguistic Annotation Workshop, ACL 2013 Workshop*, 2013.
3. F. Braune, D. Bauer, , and K. Knight. Mapping between english strings and reentrant semantic graphs. In *Proc. 9th Intl. Conf. on Language Resources and Evaluation (LREC'14)*, 2014.
4. D. Chiang, J. Andreas, D. Bauer, K. M. Hermann, B. Jones, and K. Knight. Parsing graphs with hyperedge replacement grammars. In *Proc. 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013), Volume 1: Long Papers*, pages 924–932. The Association for Computer Linguistics, 2013.
5. F. Drewes, A. Habel, and H.-J. Kreowski. Hyperedge replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations*, chapter 2, pages 95–162. World Scientific, Singapore, 1997.
6. F. Drewes, B. Hoffmann, and M. Minas. Predictive top-down parsing for hyperedge replacement grammars. In *Proc. 8th Intl. Conf. on Graph Transformation (ICGT'15)*, Lecture Notes in Computer Science. Springer, 2015.
7. A. Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer, 1992.
8. K.-J. Lange and E. Welzl. String grammars with disconnecting or a basic root of the difficulty in graph grammar parsing. *Discrete Applied Mathematics*, 16:17–30, 1987.
9. C. Lautemann. The complexity of graph languages generated by hyperedge replacement. *Acta Informatica*, 27:399–421, 1990.