Unification Grammars and Off-Line Parsability

Efrat Jaeger and Nissim Francez
Department of Computer Science
Technion, Israel Institute of Technology
32000 Haifa, Israel

Shuly Wintner
Department of Computer Science
University of Haifa
31905 Haifa, Israel

Abstract

Unification grammars are known to be Turing-equivalent; given a grammar G and a word w, it is undecidable whether $w \in L(G)$. In order to ensure decidability, several constraints on grammars, commonly known as *off-line parsability (OLP)*, were suggested, such that the recognition problem is decidable for grammars which satisfy OLP. An open question is whether it is decidable if a given grammar satisfies OLP. In this paper we investigate various definitions of OLP and discuss their inter-relations, proving that some of the OLP variants are indeed undecidable. We then present a novel, decidable OLP constraint which is more liberal than the existing decidable ones.

1 Introduction

Context-free grammars are considered to lack the expressive power needed for modeling the syntax of natural languages. Unification grammars have originated as an extension of context-free grammars, the basic idea being to augment the context-free rules with non context-free annotations (feature structures) in order to express some additional information. Unification grammars have the ability to describe phonological, morphological, syntactic and semantic properties of languages and thus they are linguistically plausible for modeling natural languages. Today, several formalisms of unification grammars exist, some of which do not assume an explicit context-free backbone.

The *recognition problem* (also known as the membership problem), for a grammar G and a string w, is whether $w \in L(G)$. The *parsing problem*, for a grammar G and a string w, is to determine all the structural descriptions (usually, trees) that are assigned by G to w. Unification grammars are Turing equivalent in their generative capacity: determining whether a given string is generated by a given grammar is as hard as deciding whether a Turing machine halts on the empty input (Johnson, 1988). Therefore, the recognition problem for unification grammars is undecidable in the general case. In order to ensure decidability of the recognition problem, several constraints on grammars, commonly known as the *off-line parsability constraints (OLP)*, were suggested, such that the recognition problem is decidable for OLP unification grammars. Several variants of OLP are known (Pereira and Warren, 1983; Johnson, 1988; Haas, 1989; Torenvliet and Trautwein, 1995; Shieber, 1992; Wintner and Francez, 1999; Kuhn, 1999); in section 3 we present many of the OLP variants and show how they ensure the decidability of the membership problem. These variants are analyzed, and their inter-relations are proven, in section 4.

Our main concern in the work is a computational investigation of the properties of OLP constraints. Some of the OLP variants were conjectured (Haas, 1989; Torenvliet and Trautwein, 1995) to be undecidable: it is undecidable whether a unification grammar satisfies the constraint. However, there exists no proof of this conjecture in the literature. Some OLP variants *are* decidable, but these conditions are too restrictive;

there is a large class of non-OLP grammars for which the membership problem is decidable. In particular, the decidable OLP definitions are limited only to unification grammars which assume an explicit context-free backbone. One of our contributions is providing undecidability proofs for four of the undecidable OLP variants (section 5). The major contribution of this paper is providing a novel, decidable OLP constraint, presented in section 6. Our constraint is more liberal than the existing decidable constraints as it applies to all unification grammar formalisms, and it can be tested effectively.

2 Unification grammars

We assume familiarity with theories of feature structures as formulated, e.g., by Shieber (1992) or Carpenter (1992). For our purposes here feature structures can be typed or untyped, as our results are valid in both cases. We summarize below the few concepts that are needed for the rest of this paper in order to set up notation.

A *multi-rooted structure* (MRS) of length n is a sequence of n feature structures, with possible reentrancies among elements of the sequence. We use the common AVM notation for depicting feature structures and MRSs, where reentrancies are indicated by boxed numbers, as in the following example of an MRS of length 3. Features are elements of a finite, non-empty set FEATS and are printed in SMALLCAPS; atoms are elements of a finite, non-empty set ATOMS and are printed in *italics*:

$$\left[\begin{array}{l}\textsc{list} : \left[\begin{array}{l}\textsc{hd} : \boxed{1}\left[\begin{array}{l}\textsc{f} : a\\ \textsc{g} : b\end{array}\right]\\ \textsc{tl} : \boxed{2}\end{array}\right]\end{array}\right] \qquad \left[\textsc{list} : \boxed{2}\right] \qquad \left[\textsc{list} : \left[\begin{array}{l}\textsc{hd} : \boxed{1}\\ \textsc{tl} : \mathit{elist}\end{array}\right]\right]$$

Meta-variables A, B range over feature structures and σ, ρ over MRSs. An MRS σ can be viewed as an ordered sequence $\langle A_1, \ldots, A_n\rangle$ of (not necessarily disjoint) feature structures. We identify MRSs of length 1 with feature structures.

Feature structures and MRSs are partially ordered by *subsumption*, denoted '$\sqsubseteq$'. The least upper bound with respect to subsumption is the *unification* operator, denoted '$\sqcup$' (we use the term 'unification' both for the operator and for the result of its application). Unification is a partial operator; when $A \sqcup B$ is undefined we say that the unification *fails*. Unification is lifted to MRSs: given two MRSs σ and ρ, it is possible to unify the i-th element of σ with the j-th element of ρ. This operation, called *unification in context* and denoted $(\sigma, i) \sqcup (\rho, j)$, yields two modified variants of σ and ρ: as the unification is done *in the context* of the entire MRSs, other elements might be affected. Hence, the result of unification in context (when it is defined) is a pair (σ', ρ').

In the sequel we distinguish between two kinds of unification grammars: those with an explicit context-free skeleton are referred to as *skeletal* unification grammars, whereas grammars with no explicit backbone are referred to as *general*. We begin with a definition of general unification grammars; these are defined over a signature consisting of FEATS, ATOMS and a finite set Σ of terminal symbols.

2.1 General unification grammars

Definition 1. *A **general unification grammar** is a tuple $G = \langle \mathcal{R}, \mathcal{L}, A^s\rangle$ where:*

- *$\mathcal{R}$ is a finite set of rules, each of which is an MRS of length $n \geq 1$, with a designated first element, the **head** of the rule, followed by its **body**. The head and body are separated by an arrow ($\rightarrow$).*
- *$\mathcal{L}$ is a **lexicon**, which associates with every terminal symbol a_i a finite set of feature structures, $\mathcal{L}(a)$.*

- A^s *is a feature structure, the* ***start symbol****.*

Figure 1 depicts an example unification grammar, G_{ww}, over the sets FEATS $= \{$LIST, HD, TL$\}$, ATOMS $= \{$*s, elist, ta, tb*$\}$ and $\Sigma = \{a, b\}$. The grammar has two rules, each an MRS of length 3, and two lexical entries, one for each element of Σ.

$$A^s = \left[\text{LIST} : \begin{bmatrix}\text{HD} : \textit{s} \\ \text{TL} : \textit{elist}\end{bmatrix}\right]$$

$$\mathcal{R} = \left\{ \begin{array}{l} \left[\text{LIST} : \begin{bmatrix}\text{HD} : \textit{s} \\ \text{TL} : \textit{elist}\end{bmatrix}\right] \longrightarrow \left[\text{LIST} : \boxed{3}\right] \quad \left[\text{LIST} : \boxed{3}\right] \\ \left[\text{LIST} : \begin{bmatrix}\text{HD} : \boxed{1} \\ \text{TL} : \boxed{2}\end{bmatrix}\right] \longrightarrow \left[\text{LIST} : \boxed{2}\right] \quad \left[\text{LIST} : \begin{bmatrix}\text{HD} : \boxed{1} \\ \text{TL} : \textit{elist}\end{bmatrix}\right] \end{array} \right\}$$

$$\mathcal{L}(a) = \left\{\left[\text{LIST} : \begin{bmatrix}\text{HD} : \textit{ta} \\ \text{TL} : \textit{elist}\end{bmatrix}\right]\right\} \qquad \mathcal{L}(b) = \left\{\left[\text{LIST} : \begin{bmatrix}\text{HD} : \textit{tb} \\ \text{TL} : \textit{elist}\end{bmatrix}\right]\right\}$$

Figure 1: An example unification grammar, G_{ww}

To define the *language* generated by a unification grammar G, we extend the notion of *forms*: a form is simply an MRS. A form $\sigma_A = \langle A_1, \ldots, A_k\rangle$ *immediately derives* another form $\sigma_B = \langle B_1, \ldots, B_m\rangle$ (denoted by $\sigma_A \Rightarrow \sigma_B$) iff there exists a rule $\rho \in \mathcal{R}$ of length n that licenses the derivation. The head of the rule is matched against some element A_i in σ_A using unification in context: $(\rho, 1) \sqcup (\sigma_A, i) = (\rho', \sigma'_A)$. If the unification does not fail, σ_B is obtained by replacing the i-th element of σ'_A with the body of ρ'. The reflexive transitive closure of '$\Rightarrow$' is denoted by '$\overset{*}{\Rightarrow}$'.

Definition 2. *The* ***language*** *of a (general) unification grammar G is $L(G) = \{w \in \Sigma^* \mid w = a_1 \cdots a_n$ and $\langle A^s\rangle \overset{*}{\Rightarrow} \sigma_l$ such that σ_l is unifiable with $\langle A_1, \ldots, A_n\rangle\}$, where $A_i \in \mathcal{L}(a_i)$ for $1 \leq i \leq n$.*

As an example, consider again the grammar G_{ww} of figure 1. The following is a derivation sequence for the string $baba$ with this grammar. Note that the scope of variables is limited to a single MRS; multiple occurrences of the same tag in a single form denote reentrancy, whereas across forms they are unrelated.

$$A^s = \left[\text{LIST} : \begin{bmatrix}\text{HD} : \textit{s} \\ \text{TL} : \textit{elist}\end{bmatrix}\right] \qquad \text{apply rule 1 to the single element of the form}$$

$$\sigma_1 = \left[\text{LIST} : \boxed{3}\right] \quad \left[\text{LIST} : \boxed{3}\right] \qquad \text{apply rule 2 to the second element}$$

$$\sigma_2 = \left[\text{LIST} : \begin{bmatrix}\text{HD} : \boxed{1} \\ \text{TL} : \boxed{2}\end{bmatrix}\right] \quad \left[\text{LIST} : \boxed{2}\right] \quad \left[\text{LIST} : \begin{bmatrix}\text{HD} : \boxed{1} \\ \text{TL} : \textit{elist}\end{bmatrix}\right] \text{ apply rule 2 to first element}$$

$$\sigma_3 = \left[\text{LIST} : \boxed{2}\right] \quad \left[\text{LIST} : \begin{bmatrix}\text{HD} : \boxed{1} \\ \text{TL} : \textit{elist}\end{bmatrix}\right] \quad \left[\text{LIST} : \boxed{2}\right] \quad \left[\text{LIST} : \begin{bmatrix}\text{HD} : \boxed{1} \\ \text{TL} : \textit{elist}\end{bmatrix}\right]$$

Now consider the MRS obtained by concatenating (the single elements of) $\langle \mathcal{L}(b), \mathcal{L}(a), \mathcal{L}(b), \mathcal{L}(a)\rangle$:

$$\sigma_l = \left[\text{LIST} : \begin{bmatrix}\text{HD} : \textit{tb} \\ \text{TL} : \textit{elist}\end{bmatrix}\right] \quad \left[\text{LIST} : \begin{bmatrix}\text{HD} : \textit{ta} \\ \text{TL} : \textit{elist}\end{bmatrix}\right] \quad \left[\text{LIST} : \begin{bmatrix}\text{HD} : \textit{tb} \\ \text{TL} : \textit{elist}\end{bmatrix}\right] \quad \left[\text{LIST} : \begin{bmatrix}\text{HD} : \textit{ta} \\ \text{TL} : \textit{elist}\end{bmatrix}\right]$$

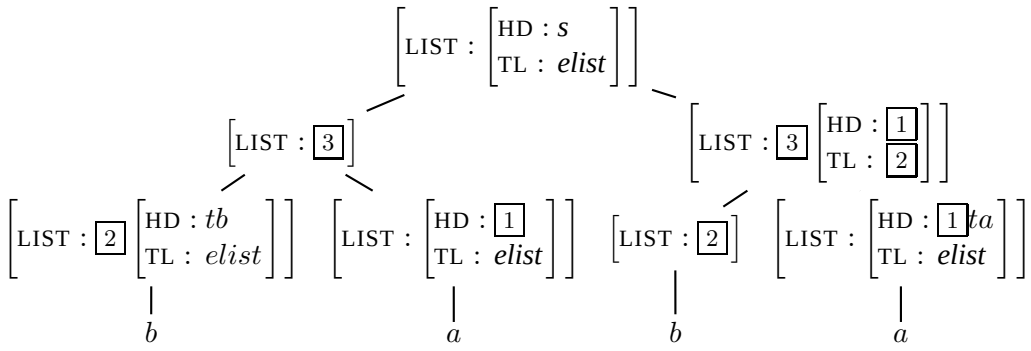


Figure 2: A derivation tree for $baba$

Since σ_l and σ_3 are unifiable, the string $baba$ is in $L(G_{ww})$. In fact, $L(G_{ww}) = \{ww \mid w \in \{a, b\}^*\}$.

The notion of derivation *trees* can be naturally extended from context-free grammars to unification grammars. The nodes of a tree are feature structures, rather than atomic non-terminal symbols. However, care must be taken when reentrancies are concerned: the scope of variables must be the *entire* tree. The leaves of the tree constitute an MRS, the i-th element of which is unifiable with a lexical entry of the i-th word of the input. For example, figure 2 depicts the derivation tree associated by G_{ww} to the string $baba$. Note that an alternative definition for derivation trees (employed later in definition 8 and referred to as *partial* derivation trees) admits trees whose frontier is *not* necessarily unifiable with lexical items, corresponding to *partial* derivations.

2.2 Skeletal grammars

Skeletal grammars assume an explicit *context-free backbone* (or *skeleton*), and can be viewed as an extension of context-free grammars, where every category is associated with an informative feature structure. They are defined over a signature which includes, in addition to Σ, FEATS and ATOMS, a finite, non-empty set CATS of categories[1] with a distinguished element, S. An *extended category* is a pair $\langle A, C\rangle$ where A is a feature structure and $C \in$ CATS is a category. The *context-free backbone* of a skeletal grammar is obtained by ignoring all feature structures of the grammar rules and considering only the categories.

Definition 3. *A **skeletal grammar** is a tuple $G = \langle \mathcal{R}, \mathcal{L}, A^s\rangle$ where:*

- *$\mathcal{R}$ is a finite set of rules, each of which is an MRS of length n $\geq$ 1 (with a designated first element, the head of the rule), and a sequence of length n of categories over the parameter* CATS *(The first category represents the head's category).*
- *$\mathcal{L}$ is a lexicon, which associates with every terminal a_i a finite set $\mathcal{L}(a)$ of extended categories $\langle A, C\rangle$, where A is a feature structure and $C \in$ CATS is a category.*
- *$A^s = \langle A, S\rangle$ is the start symbol (an extended initial category).*

Figure 3 depicts an example skeletal grammar, G_{abc}.

[1] Known also as non-terminal symbols of a context-free grammar.

$$\textsc{Cats} = \{S, A, B, C\}$$

$$A^s = \langle \left[\textsc{len} : s\right], S\rangle$$

$$\mathcal{R} = \left\{ \begin{array}{llll} \left[\textsc{len} : s\right] \longrightarrow \left[\textsc{len} : \boxed{1}\right] & \left[\textsc{len} : \boxed{1}\right] & \left[\textsc{len} : \boxed{1}\right] & S \longrightarrow A\ B\ C \\ \left[\textsc{len} : \left[\textsc{len} : \boxed{1}\right]\right] \longrightarrow \left[\textsc{len} : \boxed{1}\right] & \left[\textsc{len} : elist\right] & & A \longrightarrow A\ A \\ \left[\textsc{len} : \left[\textsc{len} : \boxed{1}\right]\right] \longrightarrow \left[\textsc{len} : \boxed{1}\right] & \left[\textsc{len} : elist\right] & & B \longrightarrow B\ B \\ \left[\textsc{len} : \left[\textsc{len} : \boxed{1}\right]\right] \longrightarrow \left[\textsc{len} : \boxed{1}\right] & \left[\textsc{len} : elist\right] & & C \longrightarrow C\ C \end{array} \right\}$$

$$\mathcal{L}(a) = \left\{\langle \left[\textsc{len} : elist\right], A\rangle\right\} \quad \mathcal{L}(b) = \left\{\langle \left[\textsc{len} : elist\right], B\rangle\right\} \quad \mathcal{L}(c) = \left\{\langle \left[\textsc{len} : elist\right], C\rangle\right\}$$

Figure 3: An example skeletal grammar, G_{abc}

A *skeletal form* is a pair $\langle \sigma, \vec{C}\rangle$, where σ is an MRS of length n and $\vec{C}$ is a sequence of n categories ($C_i \in \textsc{Cats}$ for $1 \leq i \leq n$). A skeletal form $\langle \sigma_A, \vec{C}_A\rangle$ immediately derives $\langle \sigma_B, \vec{C}_B\rangle$ iff there exists a skeletal rule $\langle \rho', \vec{C}_R\rangle \in \mathcal{R}$ that licenses the derivation (i.e., $\sigma_A \Rightarrow \sigma_B$ through the i-th element of σ_A and $\vec{C}_B$ is obtained by replacing the i-th element of $\vec{C}_A$ by the body of $\vec{C}_R$). The *language* of a skeletal grammar is defined similarly to general unification grammars using skeletal forms instead of MRSs. For example, $L(G_{abc}) = \{a^n b^n c^n \mid n > 0\}$. A *skeletal derivation tree* is a pair consisting of a unification grammar tree and a context-free tree; the two trees are isomorphic.

The context-free backbone of a skeletal derivation tree is called a *constituent structure* (c-structure). In a constituent structure, a *non-branching derivation chain* is a branch of the tree all of whose nodes have out-degree 1 (induced by the application of unit rules, i.e., rules with a single element in their bodies). When a constituent structure includes two nodes which are labeled by the same category and span exactly the same substring[2], we say that it contains a *cyclic branch*. Note that if a grammar has no ϵ-rules, a constituent structure contains a cyclic branch if and only if it has non-branching derivation chain in which the same category occurs more than once (annotating more than one node).

3 Off-line parsability constraints

The motivation behind all OLP definitions is to rule out grammars which license trees in which unbounded amount of material is generated without expanding the frontier word. This can happen due to two kinds of rules: ϵ-rules, whose bodies are empty, and unit rules, whose bodies consist of a single element. With context-free grammars the removal of rules which can cause an unbounded growth is always possible. In particular, one can always remove cyclic sequences of unit rules. However, with unification grammars such a procedure turns out to be more problematic. It is not trivial to determine when a sequence of unit rules is, indeed, cyclic; and when a rule is redundant. The definitions we discuss below try to approximate the procedure of determining whether a grammar contains such "problematic" rules.

Of course, it is always possible to ensure the decidability of the membership problem for unification

[2]By "span the same substring" we mean that the frontier dominated by both nodes is identical, not merely two copies of the same string.

grammars by constraining the expressive power of the grammars. For example, it is well known (Gazdar and Pullum, 1985; Berwick, 1987) that when the size of feature structures is guaranteed to be bounded (e.g., when no recursion is allowed in feature structures), unification grammars are equivalent in their generative capacity to context-free grammars; this is basically the motivation behind the linguistic theory GPSG (Gazdar et al., 1985). Another extreme case would be to disallow ϵ-rules and unit rules altogether. Here we are interested in a much less dramatic decrease of expressive power; in other words, OLP definitions attempt to constrain the set of grammars such that the membership problem becomes decidable, yet the expressive power of the grammars is only minimally reduced.

In this section we present several variants of OLP constraints suggested in the literature. Some of the constraints (Pereira and Warren, 1983; Kaplan and Bresnan, 1982; Johnson, 1988; Kuhn, 1999) apply only to skeletal grammars since they explicitly refer to categories, which are undetermined for general unification grammars. Others (Haas, 1989; Shieber, 1992; Torenvliet and Trautwein, 1995; Wintner and Francez, 1999) are applicable to both skeletal and general unification grammars. Some of the OLP constraints refer to OLP grammars, while others impose a restriction on allowable derivation trees rather than grammars. In this work we are concerned with OLP grammars, therefore, in definition 5, we extend these constraints from derivations to grammars.

One way to ensure that parsing with a unification grammar G terminate on every input is by guaranteeing that for every word w and every tree τ which G induces on w, the depth of τ is bounded by some known function of the length of w.

Lemma 1 (The bounding lemma). *For every unification grammar G, if a computable function f_G exists which can be efficiently computed from G, such that for every word w and every tree τ induced by G on w, the depth of τ, $d(\tau)$, is such that $d(\tau) < f_G(|w|)$, then membership for G is decidable.*

Proof. For a given word w, since the number of grammar rules is finite, and the depth of every tree induced by G on w is bounded, the set of all such trees is finite, and its cardinality is bounded by a computable function of $|w|$. Therefore an effective exhaustive search algorithm enumerates the members of this set of trees. If a tree is found by the algorithm, then $w \in L(G)$; otherwise, $w \notin L(G)$ since every tree for w must not be deeper than $f_G(|w|)$. □

As noted above, the OLP definitions discussed below attempt to provide a bounding function f_G which bounds the depth of trees admitted by G (as a function of the tree's width). Notice that the above lemma does not only require the *existence* of a bounding function f_G; in order for the function to be used for terminating the exhaustive search, it must be effectively computed for every grammar G. However, some OLP definitions only require the existence of a bounding function, possibly assuming that the grammar designer will provide it with every grammar. For such definitions, the algorithm alluded to in the bounding lemma is inapplicable.

Furthermore, note that the bounding lemma not only ensures the decidability of the recognition problem, but also the decidability of the parsing problem: if *all* the trees that are induced by a grammar are bounded in depth, then the exhaustive search algorithm is guaranteed to produce all of them in finite time. A more liberal version of the lemma would require only that for every word $w \in L(G)$, *at least one* tree induced by G on w is depth bounded. This would still ensure the decidability of the recognition problem, but not of the parsing problem.

3.1 OLP constraints for skeletal grammars

One of the first definitions of OLP constraints was suggested by Pereira and Warren (1983) for ensuring termination of general proof procedures for definite clause sets. Although the constraint was designed for DCGs, where the predicate names constitute an explicit context-free backbone, it can be rephrased in terms of skeletal grammars as follows:

Definition 4 (Pereira and Warren's OLP constraint for skeletal grammars (OLP_{PW})). *A skeletal grammar is off-line parsable iff its context-free skeleton is not infinitely ambiguous.*

We next prove that the depth of every derivation tree admitted by an OLP_{PW} grammar is bounded by the number of syntactic categories in the grammar times the size of the tree's yield.

Lemma 2. *A finitely-ambiguous context-free grammar cannot admit a derivation tree which contains a cyclic branch.*

Proof. Let G be a finitely-ambiguous context-free grammar. Assume towards a contradiction that a tree τ induced by G on some word w contains a cyclic branch. Then there exist two nodes in τ, u and v, such that u and v are labeled by the same category X and span the same substring w' of w. Without loss of generality, assume that u dominates v. Since u and v are equally labeled, the same sequence of rule applications which was used to derive v from u can be applied again from v, resulting in a subtree with a node v_1, also labeled X, on its frontier. Since u and v span the same substring, so does v_1. Now the same process can be repeated, indefinitely, from v_1, to generate an infinite number of trees with longer cyclic branches over the word w, in contradiction to the assumption of finite ambiguity. □

Lemma 3. *If τ is a constituent structure induced by an OLP_{PW} grammar G on a word w of length l, then the depth of τ is bounded by* $(|\textsc{Cats}| + 1) \times (l + 1)$.

Proof. Let G be an OLP_{PW} grammar. Let G' be the context-free backbone of G. By definition, G' is finitely ambiguous, and hence by lemma 2 it cannot admit a derivation tree which contains a cyclic branch. We first show that the length of any path from the root to a lexical item in τ is bounded by $l \times (|\textsc{Cats}| + 1)$. Let $\pi = u_0 u_1 \cdots u_n$ be a path in τ from the root u_0 to a leaf u_n. Let N be the set of all nodes u_i in π which span exactly the same substring as u_{i+1}. Let $\bar{N}$ be the set of all nodes in π which are not in N. $\bar{N}$ has at most $l-1$ nodes, since any node u_i in $\bar{N}$ spans a frontier which includes the frontier spanned by u_{i+1} and at least one other lexical item not spanned by u_{i+1}. By lemma 2, τ has no cyclic branches. Hence the maximal length of a subpath of π consisting of nodes from N is $|\textsc{Cats}|$. In other words, π consists of subpaths of maximal length $|\textsc{Cats}|$, separated by nodes from $\bar{N}$. Since $|\bar{N}| \leq l - 1$, there are at most l such subpaths in π. Therefore, the maximal length of π is $l \times |\textsc{Cats}|$ (subpaths of nodes from N) plus $(l-1)$ (nodes from $\bar{N}$), i.e., $l \times (|\textsc{Cats}| + 1) - 1$. As for leaves in τ which are labeled by ϵ, these can add one more branch of length at most $|\textsc{Cats}|$ to any path from the root, again by lemma 2. Hence the depth of τ is bounded by $(|\textsc{Cats}| + 1) \times (l + 1)$. □

Corollary 4. *Membership is decidable for OLP_{PW} grammars.*

Proof. From the above lemma and the bounding lemma. □

An additional OLP definition is based on Kaplan and Bresnan (1982). They suggest a linguistically motivated OLP constraint which refers to valid derivations for the Lexical-Functional Grammar formalism (LFG), a skeletal grammar formalism. Unlike the previous OLP definition, for LFG no notion of an OLP

grammar was proposed. Instead, LFG defines OLP *derivations*: for a given string w, a c-structure is not OLP if it has two nodes, u and v, *of the same category*, where u dominates v and u and v span exactly the same substring of w; in other words, a c-structure is not OLP if it contains a cyclic branch. Of course, there can be other derivations of w which are OLP.

Given an OLP definition for derivations, it can be extended to grammars in (at least) two manners:

Definition 5 (OLP_Δ grammar). *Let OLP_Δ be a condition on derivation trees.*

1. *A grammar G is off-line parsable iff for every $w \in L(G)$* ***every*** *derivation tree for w satisfies OLP_Δ.*
2. *A grammar G is off-line parsable iff for every $w \in L(G)$* ***there exists*** *a derivation tree which satisfies OLP_Δ.*

The first definition is very strict, licensing grammars which can only generate OLP derivation trees. The second definition is more liberal, allowing non-OLP derivation trees as long as there exists at least one OLP derivation tree for every word of the grammar's language. Using the algorithm sketched in the bounding lemma, the second definition can ensure the decidability of the recognition problem (determining whether $w \in L(G)$), whereas the first definition can also ensure the decidability of parsing (finding all the trees induced by G on w).

LFG is not a standard rewriting system, and the definition of $L(G)$ is more involved than our basic definition in section 2. That is, $w \in L(G)$ only if there exists an OLP derivation tree for w; the structures induced on w by G are the structures induced by the OLP derivations only. Other possible (non-OLP) derivations are simply ignored. LFG introduces two kinds of ϵ's, controlled and optionality ϵ's, which are used in descriptions of natural languages. General unification grammars are not necessarily designed for natural languages and thus the distinction between the ϵ kinds does not necessarily exist. Hence, we use a variant of Kaplan and Bresnan's constraint, suggested by Johnson (1988, pp. 95-97), eliminating all ϵ's of any kind.

Definition 6 (Johnson's OLP constraint (OLP_{JO})). *A constituent structure satisfies OLP_{JO} iff:*

- *it does not include a cyclic branch; and*
- *no leaf is labeled by ϵ.*

Johnson's constraint, as well as the next OLP variant discussed below, is based on Kaplan and Bresnan's OLP for LFG and hence imposes a restriction on allowable c-structures, rather than on the grammar itself. Johnson provides no explicit definition of an OLP grammar. We use definition 5 and define two conditions on grammars: grammars that satisfy the first and second interpretations of OLP_{JO} according to definition 5 are referred to as $OLP_{JO_\forall}$ and $OLP_{JO_\exists}$, respectively. Note that $OLP_{JO_\forall}$ implies that the grammar cannot include any usable ϵ-rules, whereas $OLP_{JO_\exists}$ allows such rules but requires that for each word in the language there exist at least one derivation which uses no ϵ-rules.

Lemma 5. *The depth of any OLP_{JO} derivation tree for a string of l symbols is bounded by $l \times (|\text{CATS}| + 1)$.*

Proof. Similar to lemma 3. □

Corollary 6. *Membership is decidable for $OLP_{JO_\exists}$ and $OLP_{JO_\forall}$ grammars.*

Proof. From lemma 5 and the bounding lemma. □

Furthermore, the exhaustive search algorithm of the bounding lemma ensures also that the parsing problem is decidable for $OLP_{JO_\forall}$ grammars (but not for $OLP_{JO_\exists}$ grammars).

The next constraint, proposed by Kuhn (1999), is also based on Kaplan and Bresnan's constraint and is also defined in terms of OLP *derivations*; OLP *grammar* definitions are according to definition 5. The constraint uses the notion of 'categories' and thus is applicable only to skeletal grammars. X-bar theory grammars (Chomsky, 1975) are claimed to have a strong linguistic justification in describing natural languages. However, both Kaplan and Bresnan's and Johnson's constraints disallow some valid X-bar derivations. Kuhn (1999) refers to the problem from a linguist's point of view. The purpose of his constraint is to expand the class of derivation trees which satisfy Kaplan and Bresnan's constraint in order to allow X-bar derivations. As Kuhn (1999) does not explicitly refer to ϵ-rules, we assume, similarly to Johnson's constraint, that such rules cannot be used in OLP derivations.

Kuhn (1999) shows some examples of X-bar theory derivation trees of German and Italian sentences which contain the same category twice in a non-branching dominance chain. However, he observes that in these trees, same-category nodes on a non-branching chain occur with different f-annotations. This gives rise to the following definition:

Definition 7 (Kuhn's OLP constraint). *A c-structures derivation is OLP iff no category appears twice in a non-branching dominance chain* with the same f-annotation.

The condition does not require that the feature structures to which same-category nodes on a chain are mapped be different; rather, the annotations (or equations) from which these feature structures are constructed are required to differ. This implies that while cyclic branches are allowed in derivations, they cannot involve multiple applications of the same rule. This extension changes the bounding function of OLP_{JO} from $l \times (|\textsc{Cats}| + 1)$ to $l \times (u + 1)$, where u is the number of unit rules in the grammar. Since u is constant for every grammar, lemma 5 can be used to ensure decidability in this case, too.

The main problem of analyzing Kuhn's definition is that the language of a grammar is not defined the way we defined it in section 2. As noted above, LFG defines the language of a grammar as those words that are derivable by OLP derivations. Comparing this definition with the one we use throughout this work is complicated, and therefore we exclude Kuhn's OLP constraint from this analysis.

3.2 OLP constraints for general unification grammars

Constraints that apply to general unification grammars apply, *a fortiori*, also to skeletal ones. The first such constraint was suggested by Haas (1989). Based on the observation that not every natural unification grammar has an obvious context-free backbone, Haas suggested a constraint for guaranteeing the solvability of the parsing problem which is applicable to all unification grammar formalisms. Haas' definition allows derivation trees with nonterminals at their leaves, thus allowing partial derivations.

Definition 8 (Haas' Depth-boundedness (DB)). *A unification grammar is depth-bounded iff for every $l > 0$ there is a $d > 0$ such that **every** (partial or complete) parse tree for a sentential form of l symbols has depth less than d.*

This definition of a depth-bounded grammar requires the existence of a function d, such that every derivation tree's depth for a sentential form of l symbols is bounded by $d(l)$, but it provides no explicit information about d. This condition ensures the finiteness of the set of all trees for a word $w \in L(G)$, but provides no bound on the size of the set. Hence, if $w \in L(G)$, an exhaustive search will find a tree for w, but there is no criterion for terminating the search with a negative result (the usual situation for *RE*

problems). Note also that depth-boundedness takes as a definition a condition which is a consequence of previous definitions (e.g., OLP_{PW}).

Returning to OLP_{PW}, recall that it applies only to skeletal grammars, as general unification grammars do not necessarily have an explicit context-free skeleton. A natural extension of Pereira and Warren's definition to general unification grammar formalisms is finite ambiguity.

Definition 9 (Finite ambiguity for unification grammars (FA)). *A unification grammar G is OLP_{FA} iff for every string w there exist only a finite number of derivation trees.*

Unlike OLP_{PW}, and similarly to depth-boundedness, finite ambiguity for general unification grammars does not ensure the decidability of the membership problem since it does not provide a specific function for bounding the search space.

An additional OLP definition, defined in terms of logical constraint based grammar formalisms, is suggested by Shieber (1992, pp. 79–82). Rephrased in our terms, this definition is as follows:

Definition 10 (Shieber's OLP (OLP_S)). *A grammar G is off-line parsable iff there exists a finite-ranged function F over feature structures such that $F(A) \sqsubseteq A$ for all A and there are no derivation trees admitted by G in which nodes A and B span the same substring and $F(A) = F(B)$.*

In fact, the requirement that $F(A)$ subsume A is not necessary for our purposes and was left only for compatibility with the original definition. This constraint bounds the depth of every derivation tree by the range of F times the size of the tree's yield. Thus the number of different trees whose yield is a given string is finite.

Lemma 7. *The depth of any OLP_S derivation tree for a string of length l is at most $l \times (|range(F)| + 1)$.*

Proof. Same as lemma 3, the only difference being that here the linear function of l which bounds the depth of trees is $l \times (|range(F)| + 1)$. □

Again, given a bounding function F, decidability of membership via exhaustive search is ensured by the bounding lemma; but when no such function is given, exhaustive search is not guaranteed to terminate.

In order to extend OLP definitions from formalisms that assume a context-free skeleton to general unification grammars, Torenvliet and Trautwein (1995) suggest a more liberal constraint, *honest parsability*, which is applicable to all unification grammar formalisms. This constraint is similar to depth boundedness, with two differences: (1) depth boundedness requires that *all* trees be depth bounded, whereas honest parsability only requires *one* such tree; and (2) honest parsability requires that the depth of the tree be bounded by a polynomial in its width, while depth boundedness allows any function.

Definition 11 (Honest parsability constraint (HP)). *A grammar G satisfies the Honest Parsability Constraint (HPC) iff there exists a polynomial p such that for each $w \in L(G)$* ***there exists*** *a derivation with at most $p(|w|)$ steps.*

The definition ensures that for every string of the grammar's language there exist at least one derivation tree whose depth is polynomial (in the size of the derived string). When p is explicitly given, this condition guarantees the decidability of the membership problem by the bounding lemma: all derivations can be built in parallel in increasing length, until length $p(|w|)$ is reached. When p is not given, this condition cannot guarantee it. The requirement that p be a polynomial, and not just any function, is there to guarantee that the recognition problem for HP is NP, and is not needed for proving decidability.

4 A hierarchy of OLP definitions

In this section we compare the different OLP definitions discussed in the previous section. First, we present some examples of grammars which will be used as reference in the subsequent discussion, and then discuss the OLP properties of each of these grammars. The examples use a straightforward encoding of lists as feature structures: a list is represented by brackets, list items are separated by a comma, an empty list is denoted by $\langle\ \rangle$ and $\langle \mathit{head} \mid \mathit{tail} \rangle$ represents a list whose first item is *head*, followed by *tail*.

Notice that although some of the examples use general unification grammars, they all have a context-free backbone (through the feature CAT), and thus can be viewed as skeletal grammars when definitions that apply only to skeletal grammars are investigated. For example, the general unification rule:

$$\begin{bmatrix} \text{CAT} : p \\ \text{LIST} : \boxed{1} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT} : p \\ \text{LIST} : \langle \mathit{tb} \mid \boxed{1} \rangle \end{bmatrix}$$

can be viewed as the skeletal rule:

$$\left[\text{LIST} : \boxed{1}\right] \longrightarrow \left[\text{LIST} : \langle \mathit{tb} \mid \boxed{1} \rangle\right] \qquad P \longrightarrow Q$$

4.1 Some grammar examples

A unification grammar, G_{FA}, generating the language[3] b^+, is depicted in figure 4. The feature CAT stands for 'category', and LIST is a list of lexical symbols. The string b is the only terminal item in the lexicon, therefore every string generated by the grammar consists of b's only. The second rule adds items to a list at each derivation step. The fourth rule removes items from the list.[4] With each removal a feature structure is added to the form which can be unified with (in fact, is identical to) the lexical entry of b; this process continues until a list of one item is reached. The grammar is unambiguous; a string of n occurrences of b has exactly one parse tree whose depth is $2n + 1$. Therefore, G_{FA} is FA and HP. G_{FA} is neither DB nor OLP_S; it may generate arbitrarily deep partial derivation trees (containing lists of increasing length) whose frontiers consist of only one symbol, and thus there exists no finite-ranged function mapping each feature structure on such a derivation to a finite set of feature structures. G_{FA} is neither OLP_{JO} nor OLP_{PW}; in order to generate the string b^l for $l > 1$, its second rule must be applied at least once, resulting in a non-branching dominance chain in which the category p appears more than once, and thus the context-free backbone is infinitely ambiguous.

Figure 5 presents a unification grammar, G_{inf}, generating the language $\{b\}$. The grammar rules must be applied according to their order as defined by the values of the feature CAT. The second rule adds items to the LIST list, the fourth rule removes items from the list, and a b is generated once the list consists of one item. There exist infinitely many derivation trees, of arbitrary depths, for the string b, for any natural number of applications of the second rule. Therefore, G_{inf} is neither DB nor FA nor OLP_S. G_{inf} is $OLP_{JO_\exists}$ and HP; there exists a derivation tree for the string b of depth 3. G_{inf} is neither OLP_{PW} nor $OLP_{JO_\forall}$; it may generate derivation trees in which the category p appears more than once in a non-branching dominance chain, and thus its context-free backbone is infinitely ambiguous.

A unification grammar, G_{DB}, generating the language b^+, is shown in figure 6. A string of n occurrences of b has exactly one parse tree whose depth is 2^n. The feature DEPTHCOUNT is a list that represents the

[3] We use regular expressions and set notation interchangeably to denote languages.

[4] Removing an item from a list *within* a feature structure, as in this rule, must not be confused with removing a feature structure from a form, which is the result of applying an ϵ-rule.

$$A^s = \left[\text{CAT} : s\right]$$

$$\mathcal{R} = \left\{ \begin{array}{ll} \rho_1: & \left[\text{CAT} : s\right] \longrightarrow \begin{bmatrix}\text{CAT} : p \\ \text{LIST} : \langle tb \rangle\end{bmatrix} \\ \rho_2: & \begin{bmatrix}\text{CAT} : p \\ \text{LIST} : \boxed{1}\end{bmatrix} \longrightarrow \begin{bmatrix}\text{CAT} : p \\ \text{LIST} : \langle tb \mid \boxed{1} \rangle\end{bmatrix} \\ \rho_3: & \begin{bmatrix}\text{CAT} : p \\ \text{LIST} : \boxed{1}\end{bmatrix} \longrightarrow \begin{bmatrix}\text{CAT} : q \\ \text{LIST} : \boxed{1}\end{bmatrix} \\ \rho_4: & \begin{bmatrix}\text{CAT} : q \\ \text{LIST} : \langle tb \mid \boxed{1} \rangle\end{bmatrix} \longrightarrow \begin{bmatrix}\text{CAT} : q \\ \text{LIST} : \boxed{1}\end{bmatrix} \quad \begin{bmatrix}\text{CAT} : q \\ \text{LIST} : \langle tb \rangle\end{bmatrix} \end{array} \right\}$$

$$\mathcal{L}(b) = \left\{ \begin{bmatrix}\text{CAT} : q \\ \text{LIST} : \langle tb \rangle\end{bmatrix} \right\}$$

Figure 4: A unification grammar, G_{FA}

$$A^s = \begin{bmatrix}\text{CAT} : s \\ \text{LIST} : \langle s \rangle\end{bmatrix}$$

$$\mathcal{R} = \left\{ \begin{array}{ll} \rho_1: & \begin{bmatrix}\text{CAT} : s \\ \text{LIST} : \langle s \rangle\end{bmatrix} \longrightarrow \begin{bmatrix}\text{CAT} : p \\ \text{LIST} : \langle tb \rangle\end{bmatrix} \\ \rho_2: & \begin{bmatrix}\text{CAT} : p \\ \text{LIST} : \boxed{1}\end{bmatrix} \longrightarrow \begin{bmatrix}\text{CAT} : p \\ \text{LIST} : \langle tb \mid \boxed{1} \rangle\end{bmatrix} \\ \rho_3: & \begin{bmatrix}\text{CAT} : p \\ \text{LIST} : \boxed{1}\end{bmatrix} \longrightarrow \begin{bmatrix}\text{CAT} : q \\ \text{LIST} : \boxed{1}\end{bmatrix} \\ \rho_4: & \begin{bmatrix}\text{CAT} : q \\ \text{LIST} : \langle tb \mid \boxed{1} \rangle\end{bmatrix} \longrightarrow \begin{bmatrix}\text{CAT} : q \\ \text{LIST} : \boxed{1}\end{bmatrix} \end{array} \right\}$$

$$\mathcal{L}(b) = \left\{ \begin{bmatrix}\text{CAT} : q \\ \text{LIST} : \langle tb \rangle\end{bmatrix} \right\}$$

Figure 5: A unification grammar, G_{inf}

current depth of the derivation tree (i.e, the number of derivation steps from the root). In each derivation step an item is added to the DEPTHCOUNT list, but no items may ever be removed from it. The feature INNERCOUNT is a list that represents the number of derivation steps before generating the next b symbol. Every application of the second rule doubles the depth of the INNERCOUNT list with respect to its length after the previous application of the rule. Thus the number of derivation steps for generating each b is always twice the number of steps for generating its predecessor. Therefore, for every sentential form of length n, the depth of any partial derivation tree is bounded by an exponential function of n (approximately 2^n). Thus G_{DB} is DB and FA. G_{DB} is not HP; no $w \in L(G_{DB})$ has a polynomial (in $|w|$) depth derivation tree. G_{DB} is not OLP_S; in order to generate the string b^l exactly 2^l derivation steps must be applied, in order to generate b^{l+1} exactly 2^{l+1} derivation steps must be applied. Between the generation of the l^{th} and $(l+1)^{th}$ symbols there must be 2^l derivation steps of nodes spanning the same yield for every natural

number l. Therefore, there exist no finite-ranged mapping function satisfying the conditions. G_{DB} is neither OLP_{JO} nor OLP_{PW}; in order to generate the string b^l for $l > 1$, its third rule must be applied at least once, resulting in a non-branching dominance chain in which the category p appears more than once, and thus the context-free backbone is infinitely ambiguous.

$$A^s = \begin{bmatrix} \text{CAT} : & s \\ depthCount : & \langle\ \rangle \\ innerCount : & \langle\ \rangle \end{bmatrix}$$

$$\mathcal{R} = \left\{ \begin{array}{lll} \rho_1 : & \begin{bmatrix} \text{CAT} : & s \\ depthCount : & \langle\ \rangle \\ innerCount : & \langle\ \rangle \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : & p \\ depthCount : & \langle tb\ \rangle \\ innerCount : & \langle\ \rangle \end{bmatrix} & \\ \rho_2 : & \begin{bmatrix} \text{CAT} : & p \\ depthCount : & \langle\ \boxed{1}\ \rangle \\ innerCount : & \langle\ \rangle \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : & p \\ depthCount : & \langle tb \mid \boxed{1}\ \rangle \\ innerCount : & \boxed{1} \end{bmatrix} & \begin{bmatrix} \text{CAT} : l \\ lex : \ tb \end{bmatrix} \\ \rho_3 : & \begin{bmatrix} \text{CAT} : & p \\ depthCount : & \boxed{1} \\ innerCount : & \langle tb \mid \boxed{2}\ \rangle \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : & p \\ depthCount : & \langle tb \mid \boxed{1}\ \rangle \\ innerCount : & \boxed{2} \end{bmatrix} & \\ \rho_4 : & \begin{bmatrix} \text{CAT} : & p \\ depthCount : & \langle tb \mid \boxed{1}\ \rangle \\ innerCount : & \langle\ \rangle \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : l \\ lex : \ tb \end{bmatrix} & \end{array} \right\}$$

$$\mathcal{L}(b) = \left\{ \begin{bmatrix} \text{CAT} : l \\ lex : \ tb \end{bmatrix} \right\}$$

Figure 6: A unification grammar, G_{DB}

A skeletal unification grammar, $G_{J\backslash PW}$, generating the language $\{a, b\}$, is presented in Figure 7. An a is generated from the category P and a b from the category Q. The first and third rules are initial rules; after applying the first rule, the only applicable rule is the second rule, generating a b, then no other rule may be applied. After applying the third rule, the only applicable rule is the fourth rule, generating an a, then no other rule may be applied. Therefore, the grammar can only generate the two derivation trees shown in the figure. Since both derivations are OLP_{JO}, the grammar is both $OLP_{JO_\forall}$ and $OLP_{JO_\exists}$. $G_{J\backslash PW}$ is not OLP_{PW} as its context-free backbone can generate a unary branching chain of $P - Q - P$.

Figure 8 depicts a skeletal grammar, G_{S_1}. It is easy to verify that the given derivation is the only possible derivation admitted by G_{S_1}. The derivation is of depth 2, therefore the grammar is both OLP_S and HP (for OLP_S, the finite-ranged function should map $\left[\text{F} : s\right]$ and $\left[\text{F} : \left[\text{F} : s\right]\right]$ to themselves and any other feature structure to the empty feature structure).

To summarize the above discussion, the table of figure 9 lists the off-line parsability properties of the example grammars. Empty entries in the table are not needed for the evaluation.

4.2 The relationships between the OLP variants

This section compares the OLP variants and defines a hierarchy among them, referring to the grammar examples given above. The propositions in this section refer to the properties of the example grammars, as summarized in figure 9.

Proposition 8. *No OLP constraint implies any of the OLP_{JO} constraints.*

$\textsc{Cats} = \{S, P, Q\}$

$A^s = \langle \left[\textsc{list} : \langle\ \rangle\right], S\rangle$

$$\mathcal{R} = \left\{ \begin{array}{lll} \rho_1: & \left[\textsc{list} : \langle\ \rangle\right] \longrightarrow \left[\textsc{list} : \langle\ \rangle\right] & S \longrightarrow P \\ \rho_2: & \left[\textsc{list} : \langle\ \rangle\right] \longrightarrow \left[\textsc{list} : \langle tb\ \rangle\right] & P \longrightarrow Q \\ \rho_3: & \left[\textsc{list} : \langle\ \rangle\right] \longrightarrow \left[\textsc{list} : \langle\ \rangle\right] & S \longrightarrow Q \\ \rho_4: & \left[\textsc{list} : \langle\ \rangle\right] \longrightarrow \left[\textsc{list} : \langle ta\ \rangle\right] & Q \longrightarrow P \end{array} \right\}$$

$\mathcal{L}(a) = \left\{\langle\left[\textsc{list} : \langle ta\ \rangle\right], P\rangle\right\}\ \ \mathcal{L}(b) = \left\{\langle\left[\textsc{list} : \langle tb\ \rangle\right], Q\rangle\right\}$

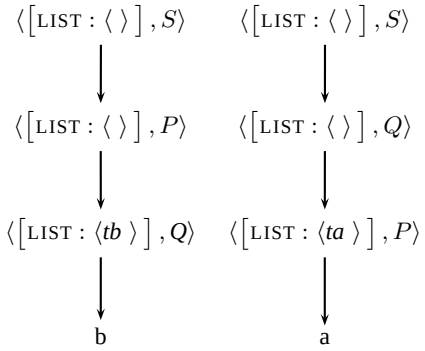


Figure 7: An OLP_{JO} grammar $G_{J\backslash PW}$, $L(G_{J\backslash PW}) = \{a, b\}$

$\textsc{Cats} = \{S\}$
$A^s = \langle\left[\textsc{f} : s\right], S\rangle$
$\mathcal{R} = \{\ \left[\textsc{f} : \boxed{1}\right] \longrightarrow \left[\textsc{f} : \left[\textsc{f} : \boxed{1}\ \right]\ \right] \qquad S \longrightarrow S\ \}$
$\mathcal{L}(b) = \left\{\langle\left[\textsc{f} : \left[\textsc{f} : s\right]\ \right], S\rangle\right\}$

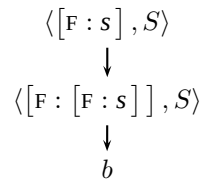


Figure 8: An OLP_S grammar, G_{S_1}

Since Johnson's condition is the only one disallowing ϵ-rules in derivations, none of the other constraints implies either OLP_{JO} constraint.

Proposition 9. *If G is DB then G is FA.*

Proof. Let G be DB. Then there exists a function f such that for every $w \in L(G)$, the depth d of any derivation tree for w satisfies $d \leq f(|w|)$. Hence there are only finitely many trees for w, and G is FA. □

Proposition 10. *An OLP_{JO} grammar G is not necessarily an OLP_{PW} grammar.*

Grammar	Fig.	FA	HP	DB	OLP_S	$OLP_{JO_\forall}$	$OLP_{JO_\exists}$	OLP_{PW}
G_{FA}	4	+	+	–	–	–	–	–
G_{inf}	5	–	+	–	–	–	+	–
G_{DB}	6	+	–	+	–	–	–	–
$G_{J\backslash PW}$	7					+	+	–
G_{S_1}	8		+		+			

Figure 9: Off-line parsability properties of the example grammars

Proof. The grammar $G_{J\setminus PW}$ of figure 7 is OLP_{JO}, but it is not OLP_{PW}. □

Proposition 11. *If G is OLP_{PW} or $OLP_{JO_\forall}$ then G is DB, FA and HP.*

Proof. Let G be either an OLP_{PW} or an $OLP_{JO_\forall}$ grammar. By lemma 3 and lemma 5, the depth of any derivation tree (partial/non-partial) admitted by G is bounded by a linear function of the size of its yield. Therefore, for every sentential form of length l every derivation tree's depth is bounded by a linear function of l and there exist only a finite number of derivation trees up to any finite depth. □

Proposition 12. *If G is OLP_{PW} or $OLP_{JO_\forall}$ then G is OLP_S*

Proof. If G is OLP_{PW} or $OLP_{JO_\forall}$ then it cannot generate a derivation tree in which two nodes have the same category and span the same yield. Therefore, the function F with the finite range $\{\left[\text{CAT} : X\right] \mid X \in \text{CATS}\}$, given by:

$$\forall X \in \text{CATS} : F\left(\begin{bmatrix}\text{CAT} : X \\ \vdots\end{bmatrix}\right) = \left[\text{CAT} : X\right]$$

is such that for every A, $F(A) \sqsubseteq A$ and it is ensured that, in every derivation tree, every two nodes spanning the same yield are mapped by F to a different image. Note that the fact that $F(A) \sqsubseteq A$ is not used here. □

Proposition 13. *A DB or FA grammar G is not necessarily an OLP_{PW} grammar.*

Proof. The grammar G_{DB} of figure 6 is depth-bounded and finitely ambiguous. Its context-free backbone can generate a unary branching chain of P's which immediately leads to infinite ambiguity. □

Proposition 14. *An OLP_S or HP grammar G is not necessarily an OLP_{PW} grammar.*

Proof. The grammar G_{S_1} of figure 8 is both OLP_S and HP. It is not OLP_{PW}, as its context-free backbone can generate a unary branching chain of S's. □

Proposition 15. *An $OLP_{JO_\exists}$ grammar G is not necessarily DB nor FA nor OLP_S.*

Proof. The grammar G_{inf} of figure 5 is $OLP_{JO_\exists}$; there exists an OLP_{JO} derivation for the string b (by applying the first and third rules). But G_{inf} may generate infinitely many non-branching derivation trees of arbitrary depths for the string b and thus it is neither DB nor FA nor OLP_S. □

Proposition 16. *If G is OLP_{JO} then G is HP.*

Proof. If G is OLP_{JO}, then for every $w \in L(G)$ there exists at least one OLP_{JO} derivation. By lemma 5, the derivation's depth is bounded by a linear function of the length of w, therefore, for every $w \in L(G)$ there exists a polynomial function of $|w|$ bounding the derivation's depth. □

Proposition 17. *An FA grammar G is not necessarily a DB grammar.*

Proof. The grammar G_{FA} of figure 4 is finitely ambiguous; the grammar induces a finite number of derivation trees on every string. The grammar is not depth-bounded; the definition of depth boundedness allows partial derivation trees with non-terminals at their leaves, therefore the second rule may be applied repeatedly many times, generating arbitrarily deep parse trees whose frontier is a sequence of non-terminals of length 1. □

Proposition 18. *A DB grammar G is not necessarily an OLP_S grammar.*

Proof. The grammar G_{DB} of figure 6 is depth-bounded, but it is not OLP_S. □

Proposition 19. *An OLP_S grammar G is not necessarily a DB grammar.*

Proof. The grammar G_{S_1} of figure 8 is OLP_S. It is not depth-bounded, as its sole rule may be applied repeatedly many times, resulting in arbitrarily deep parse trees whose frontier has only one symbol. □

Proposition 20. *A DB or FA grammar G is not necessarily an HP grammar.*

Proof. The grammar G_{DB} of figure 6 is depth-bounded; every string of length n has a unique parse tree of depth 2^n and there exist no arbitrarily deep partial trees with non-terminals at their leaves. The grammar is not an HP grammar; for every $w \in L(G_{DB})$ every parse tree is of an exponential depth (in $|w|$). □

Proposition 21. *An HP grammar G is not necessarily DB nor FA.*

Proof. The grammar G_{inf} of figure 5 is an HP grammar; for every string there exists a derivation tree of a polynomial depth in the size of the string. The grammar is neither DB nor FA; it may generate infinitely many derivation trees of arbitrary depths for the string b. □

Proposition 22. *If G is OLP_S then G is FA and HP.*

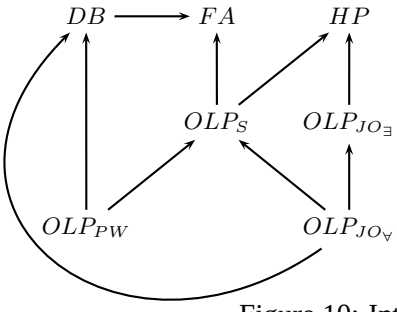
Proof. By lemma 7, if G is OLP_S then the depth of every derivation tree for a string of length l is bounded by $|range(F)| \times l$ (where F is a finite-ranged function satisfying the constraint). Since $|range(F)|$ is independent of l, there exists a polynomial function in the size of the string bounding its derivations' depth. Hence G is HP. Since the depth of *every* tree is thus bounded, only finitely many trees are possible, hence G is FA. □

Proposition 23. *An FA or HP grammar G is not necessarily an OLP_S grammar.*

Proof. The grammar G_{FA} of figure 4 is both FA and HP, but it is not OLP_S. □

Figure 10 depicts the inter-relations hierarchy diagram of the OLP variants, separated for skeletal and general unification grammars. The hierarchy is based on the above analysis where the arrows represent the discussed implications; an arrow from C_1 to C_2 implies that any grammar satisfying C_1 also satisfies C_2.

OLP for **skeletal** grammars:



OLP for **general** unification grammars:

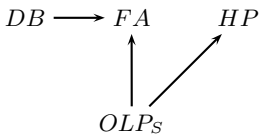


Figure 10: Inter-relations Hierarchy diagram

5 Undecidability proofs

The problem of deciding whether a given grammar is OLP was conjectured to be undecidable (Haas, 1989; Torenvliet and Trautwein, 1995), but no proof has ever been provided. We present undecidability proofs for four OLP definitions: Johnson's OLP (OLP_{JO}), Finite Ambiguity (FA), Depth-Boundedness (DB) and Shieber's OLP (OLP_S).

In order to prove that OLP_{JO}, DB and OLP_S are undecidable we use Johnson's (1988) reduction from the Turing machine halting problem (on the empty input) to the recognition problem for unification grammars. The reduction defines a unification grammar, G_M, for every (deterministic) Turing machine, M, such that the grammar contains only unit rules and can generate at most one complete derivation tree; G_M generates the word halt if and only if the machine accepts the empty input string.

$$L(G_M) = \begin{cases} \{\mathsf{halt}\} & \text{if } M \text{ terminates on the empty input} \\ \emptyset & \text{if } M \text{ does not terminate on the empty input} \end{cases}$$

A characteristic feature of the grammar G_M, for every M, is that its rules are all unit rules, and hence the derivation trees it induces are all non-branching. Furthermore, due to the fact that the Turing machine is deterministic (in particular, since its transition function is uniquely defined for every combination of state and tape symbol), the constructed grammar is such that for every derivation whose frontier is a non-terminal, exactly one grammar rule can be applied to that non-terminal.

We show that any algorithm deciding OLP_{JO}, FA, DB or OLP_S can be used as a subroutine in an algorithm which solves the empty input halting problem.

5.1 Undecidability of OLP_{JO}

Theorem 24. *Given a unification grammar G, it is undecidable whether G is $OLP_{JO_\forall}$ or whether G is $OLP_{JO_\exists}$.*

Proof. Assume towards a contradiction that an algorithm A exists for deciding $OLP_{JO_\forall}$ (or $OLP_{JO_\exists}$). Construct an algorithm B to decide the empty input halting problem, which is known to be undecidable, with B operating as follows: on input $M = (Q, \Sigma, \flat, \delta, s, h)$, a Turing machine:

1. Construct G_M, simulating the operation of M on the empty input, as described above.
2. Run algorithm A on G_M.
3. - If G_M is determined by A *not* to be OLP_{JO}: then there exists some word $w \in L(G_M)$ for which every (in the case of $OLP_{JO_\forall}$) or at least one (in the case of $OLP_{JO_\exists}$) derivation tree is not OLP_{JO}. In any case, there exists some word in $L(G_M)$, and this word must be halt. Hence M terminates on the empty input.
   - If G_M is determined by A to be OLP_{JO}: by corollary 6, membership is decidable for G_M. Decide whether $\mathsf{halt} \in L(G_M)$. If it is, M terminates on the empty input; otherwise, M does not terminate on the empty input.

□

5.2 Undecidability of Finite Ambiguity

Theorem 25. *Given a unification grammar G and a string w, it is undecidable whether the number of derivation trees G induces on w is finite.*

Proof. Assume towards a contradiction that there exists an algorithm, A_{FA}, deciding whether w has a finite number of derivation trees admitted by G. Construct an algorithm, B_{FA}, to decide the membership problem (contradicting Johnson's (1988) theorem), with B_{FA} operating as follows.

On input G, w where $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$ is a unification grammar and w is a string:

1. Construct $G' = \langle \mathcal{R}', \mathcal{L}', A'^s \rangle$ such that $\mathcal{L}' = \mathcal{L}$, $A'^s = A^s$ and $\mathcal{R}' = \mathcal{R} \cup \{A^s \Rightarrow A^s\}$. Observe that $L(G) = L(G')$, independently of whether or not the rule $A^s \Rightarrow A^s$ is in $\mathcal{R}$.
2. Run algorithm A_{FA} on G', w.
3. $w \in L(G)$ iff G' induces an infinite number of derivation trees on w.

If G' can only generate a finite number of derivation trees for a string w, then this number is 0. Suppose that there exists a derivation tree for w admitted by G'. By applying the rule $A^s \Rightarrow A^s$, G' can generate infinitely many derivation trees for w, contradicting the algorithm's outcome. Therefore, $L(G') = \emptyset$ and since $L(G') = L(G)$, also $L(G) = \emptyset$ and hence $w \notin L(G)$.

If G' can generate infinitely many derivation trees for a string w, then there exists at least one derivation tree for w admitted by G' which does not contain any applications of the rule $A^s \Rightarrow A^s$ (since the rule only loops over the start symbol) and therefore there exists a derivation tree for w admitted by G; hence $w \in L(G)$. □

Corollary 26. *Finite Ambiguity is undecidable.*

Proof. By theorem 25, it is undecidable whether G can generate a finite number of derivation trees on a string w. Therefore it is undecidable whether for every w (over Σ, the grammar's terminal symbols) there exist a finite number of derivation trees. □

5.3 Undecidability of Depth-Boundedness

Theorem 27. *depth-boundedness is undecidable.*

Proof. Assume toward a contradiction that there exists an algorithm, A_{DB}, for deciding depth-boundedness. The algorithm decides whether there exists a function f bounding the depth of each derivation tree for a sentential form of length n by $f(n)$. Construct an algorithm, B_{DB}, to decide the empty input halting problem, which is known to be undecidable, with B_{DB} operating as follows.

On input $M = (Q, \Sigma, \flat, \delta, s, h)$, a Turing machine:

1. Construct G_M, simulating the operation of M on the empty input, as described above.
2. Run algorithm A_{DB} on G_M.
3. M terminates on the empty input iff G_M is depth-bounded.

If G_M is depth-bounded then M terminates on the empty input: since G_M is depth bounded it can only generate, for a given word, finitely many (partial) non-branching derivation trees. One of these derivation trees must end in a terminal, because if all the trees had a frontier of a non-terminal, then for each such tree a grammar rule could have been applied (due to the 'total' character of the grammar), resulting in infinitely many trees. Since the grammar has only one terminal, halt, $\mathsf{halt} \in L(G_M)$ and M terminates on the empty input.

If G_M is not depth-bounded, then G_M generates infinitely many non-branching partial derivation trees, none of which ends with a terminal. Assume towards a contradiction that G_M can generate a complete derivation tree. Therefore, since δ is deterministic and undefined on the final state, and G_M simulates it, while reaching a final state, there is no next applicable rule. Hence G_M may only generate a finite set of partial derivation trees (where each is a sub-tree of the complete derivation tree), a contradiction. Therefore, G_M may not generate any complete derivation trees and thus $\mathsf{halt} \notin L(G)$ (M does not terminate on the empty input). □

5.4 Undecidability of OLP_S

Theorem 28. *OLP_S is undecidable.*

Proof. Assume towards a contradiction that there exists an algorithm, A_S, for deciding whether a grammar satisfies OLP_S, that is, deciding whether there exists a finite-ranged function F such that $F(A) \sqsubseteq A$ for all A and there are no derivation trees admitted by G in which a node $\langle u \rangle$ dominates a node $\langle v \rangle$, both are roots of sub-trees with an identical yield and $F(u) = F(v)$. Construct an algorithm, B_S, to decide the empty input halting problem, which is known to be undecidable, with B_S operating as follows.

On input $M = (Q, \Sigma, \flat, \delta, s, h)$, a Turing machine:

1. Construct G_M, simulating the operation of M on the empty input, as described above.
2. Construct G'_M by adding the rule $A^s \Rightarrow A^s$ to G_M's set of rules: $\mathcal{R}'_M = \mathcal{R}_M \cup \{A^s \Rightarrow A^s\}$ (in this case, the rule $A^s \Rightarrow A^s$ cannot be in $\mathcal{R}_M$, by the construction of G_M).
3. Run algorithm A_S on G'_M.
4. M terminates on the empty input iff G'_M is not OLP_S.

If G'_M is OLP_S then there exists no derivation tree for halt admitted by G'_M. Suppose that one exists; then by applying the rule $A^s \Rightarrow A^s$, G'_M can generate infinitely many derivation trees for the string halt of arbitrary depths, and hence no finite-ranged function exists which maps each two nodes on a derivation with the same yield to a different image, contradicting the algorithm's outcome. Therefore, there exists no derivation tree for halt admitted by G_M, $\mathsf{halt} \notin L(G_M)$ (M does not terminate on the empty input).

If G'_M is not OLP_S, then there exist a derivation tree admitted by G'_M for which every finite-ranged function must map at least two nodes spanning the same yield to the same image. Particularly, since halt is the only terminal symbol, G'_M induces a derivation tree on halt. By the construction of G'_M, there exists a derivation tree for halt admitted by G_M, $\mathsf{halt} \in L(G_M)$ (M terminates on the empty input). □

6 A novel OLP constraint - OLP_D

In this section we present our main contribution, a decidable OLP constraint. Our constraint applies to both skeletal and general unification grammars and, unlike all the definitions that apply to general unification

grammars, it can be tested efficiently. We also provide some improvements to our constraint. The optimized constraint is more liberal than all the previously proposed decidable OLP constraints, since it is applicable to general unification grammars as well as to skeletal grammars. Thus more grammars are rendered off-line parsable. As we shall show, every grammar that is off-line parsable by any of the decidable constraints is also off-line parsable by our constraint (but not vice versa).

6.1 A decidable definition of OLP (version 1)

The OLP constraint proposed here, OLP_{D_1}, disallows grammars which can generate derivation trees in which the same rule may be applied more than once from two different nodes dominating the same yield. In this version we assume that grammars include no ϵ-rules, but a more liberal constraint allowing them is given in section 6.4. Since grammars do not include ϵ-rules, the definition only addresses unit rule chains (unlike context-free grammars, these chains cannot be eliminated). Unlike the undecidable constraints discussed above, our constraint is a static property of the grammar, which can be tested off-line, without resorting to possible derivations of the input string.

Definition 12. *A sequence of unit rules $R_1, \ldots, R_k$ ($k \geq 1$) is **cyclicly unifiable** iff there exists a sequence of feature structures*[5] *$\sigma_1, \ldots, \sigma_{k+2}$ such that for $1 \leq i \leq k$, $\sigma_i \Rightarrow \sigma_{i+1}$ by the rule R_i, and $\sigma_{k+1} \Rightarrow \sigma_{k+2}$ by R_1.*

Figure 11 displays two grammar rules, ρ_1, ρ_2. The sequence $\langle \rho_1, \rho_2 \rangle$ is cyclicly unifiable, e.g. by $\langle \sigma_1 = \begin{bmatrix} \text{CAT} : p \\ \text{F} : a \end{bmatrix}, \sigma_2 = \begin{bmatrix} \text{CAT} : q \\ \text{F} : a \end{bmatrix}, \sigma_3 = \begin{bmatrix} \text{F} : b \end{bmatrix}, \sigma_4 = \begin{bmatrix} \text{CAT} : q \\ \text{F} : b \end{bmatrix} \rangle$.
σ_1 is unifiable with ρ_1's head, $\sigma_1 \Rightarrow \sigma_2$ by ρ_1, $\sigma_2 \Rightarrow \sigma_3$ by ρ_2, and then $\sigma_3 \Rightarrow \sigma_4$ by ρ_1.

The sequence $\langle \rho_2, \rho_1 \rangle$ is not cyclicly unifiable; whatever ρ_2 applies to, the resulting feature structure is $\begin{bmatrix} \text{F} : b \end{bmatrix}$; then, applying ρ_1 necessarily yields $\begin{bmatrix} \text{CAT} : q \\ \text{F} : b \end{bmatrix}$, which is incompatible with the head of ρ_2. Hence ρ_2 cannot be applied again.

$$\mathcal{R} = \left\{ \begin{array}{lll} \rho_1 : & \begin{bmatrix} \text{CAT} : p \\ \text{F} : \boxed{1} \end{bmatrix} & \longrightarrow \quad \begin{bmatrix} \text{CAT} : q \\ \text{F} : \boxed{1} \end{bmatrix} \\ \rho_2 : & \begin{bmatrix} \text{F} : a \end{bmatrix} & \longrightarrow \quad \begin{bmatrix} \text{F} : b \end{bmatrix} \end{array} \right\}$$

Figure 11: Cyclicly unifiability example

Definition 13 (A decidable OLP constraint (OLP_{D_1})). *A grammar G is OLP_{D_1} iff it contains no cyclicly unifiable sequences.*

Lemma 29. *If a grammar G contains no cyclicly unifiable sequences, G does not license any derivation tree with a non-branching dominance chain in which the same rule is used more than once.*

Proof. Assume towards a contradiction that a unit rule ρ_1 is used more than once in a non-branching dominance chain. Therefore, there exists a sequence of MRSs $\sigma_1, \ldots, \sigma_{k+2}$, the chain nodes on the derivation tree, and a sequence of unit rules $\rho_1, \ldots, \rho_k$, such that for $1 \leq i \leq k$, $\sigma_i \Rightarrow \sigma_{i+1}$ by ρ_i, and $\sigma_{k+1} \Rightarrow \sigma_{k+2}$ by ρ_1. Thus, the grammar contains a cyclicly unifiable sequence, a contradiction. □

[5] Formally, these should be MRSs of length 1, which are identified with feature structures here.

Lemma 30. *The depth of every derivation tree whose yield is of length n admitted by an OLP_{D_1} grammar G is bounded by $(u+1) \times n$, where u is the number of G's unit rules.*

Proof. Since the grammar contains no cyclicly unifiable sequences, by lemma 29 no rule may be applied more than once in a non-branching dominance chain. Therefore, the depth of any generated non-branching dominance chain is bounded by u. Thus in every derivation tree admitted by G, every u consecutive applications of unit rules (at most) are followed by either a leaf or an application of a non-unit rule expanding the yield (recall that no ϵ-rules are allowed). Therefore, the depth of every derivation tree is at most $(u+1)$ times the size of its yield. □

Corollary 31. *The membership problem is decidable for OLP_{D_1} grammars.*

Proof. From lemma 30 and the bounding lemma. □

Theorem 32. *It is decidable whether a grammar is OLP_{D_1}.*

Proof. An algorithm for the problem is given in the next section. □

6.2 An algorithm for deciding OLP_{D_1}

In order to detect cyclicly unifiable sequences, only unit rules need be considered. The algorithm uses a graph annotation and searches for cycles in the graph.

We first create a *unit rules graph, URG*, which is a directed graph representing unifiability; every vertex is a unit rule, and an edge leads from u to v iff the body of u is unifiable with the head of v (the body is of length 1). The head and the single element in the body of a unit rule ρ_i are represented by H_i, B_i respectively. Obviously, the graph is finite.

Then, we search the *URG* for cycles, which may indicate a cyclicly unifiable sequence. For each cycle, we approximate its operation by consecutively applying all its vertices in order to verify whether they form a cyclicly unifiable sequence. Approximation is done by applying the rules according to the order of the sequence, starting with the empty feature structure.

This process is only an approximation, since the cycle edges represent unifiability between the head and body of each two consecutive cycle vertices, but they are not necessarily indicative of a cyclicly unifiable sequence. It is not guaranteed that after applying several rules, unifiability between the resulting feature structure and the head of the next rule still holds. Approximation of the cycle is done beginning each time with a different cycle vertex. It is possible that by beginning the approximation with some vertex, the cycle's vertices form a cyclicly unifiable sequence, but for others they do not, as exemplified by figure 11.

Note that even if a grammar contains cyclicly unifiable sequences, it does not necessarily imply that any of the cycle vertices may ever be applied. Thus, the constraint may be ruling out grammars for which recognition is decidable, but it is still a decidable constraint and allows (along with the improvements) more grammars than the previously proposed decidable constraints. The algorithm is listed in figure 12.

Since any cyclicly unifiable sequence is represented as a cycle in the URG, only cycles of vertices should be considered. Once a cycle is detected, it does not necessarily imply that its vertices form a cyclicly unifiable sequence. We approximate the cycle's operation using the function *is_cyclicly_unifiable* on each cyclic rotation of its vertices. The function applies each of the rules consecutively using unification in context (as defined in page 2). If one of the rules may not be applied (the resulting feature structure is not unifiable with the rule's head) it returns **false**; if all rules have been applied successfully, the function returns **true**.

An algorithm for deciding OLP_{D_1}

scan_grammar(G): Boolean

Input: A unification grammar G.
Output: **true** iff G is OLP_{D_1}.

Construct a directed unit rules graph, URG, where

Each vertex is a unit rule $\rho \in \mathcal{R}$ where $|\rho| = 2$.

There exists a directed edge from vertex $\langle H_1, B_1 \rangle$ toward vertex $\langle H_2, B_2 \rangle$ iff B_1 is unifiable with H_2.

For each cycle $C = C_1, \ldots, C_k, C_1$ in URG (where the C_i-s are nodes):

for i from 0 to $k-1$

Let $V_1 \ldots V_k$ be the cyclic rotation of C, i positions to the right.

If *is_cyclicly_unifiable*$(V_1, \ldots, V_k)$ return **false**

Return **true**.

is_cyclicly_unifiable($V_1, \ldots, V_k$): Boolean

$FS = [\,]$ /* the most general feature structure */

for i from 1 to k

$V_i = \langle H_i, B_i \rangle$ is the current rule.

if $FS \sqcup H_i$ fails return **false**

else $((FS, 1) \sqcup (V_i, 1) = \langle FS', V_i' \rangle)$,where $V_i' = \langle H_i', B_i' \rangle$

$FS = B_i'$ /* unification in context */

if $FS \sqcup H_1$ fails return **false**

Return **true**

Figure 12: An algorithm for deciding OLP_{D_1}

6.2.1 Correctness of the algorithm

Lemma 33. *If a sequence of unit rules does not appear as a cycle in the URG, then it is not cyclicly unifiable.*

Proof. Let $R_1, \ldots, R_k$ be a cyclicly unifiable sequence, let H_i,B_i be the head and body of each R_i respectively. Therefore, there exists a sequence $\sigma_1, \ldots, \sigma_{k+2}$, such that for each $1 \leq i \leq k-1$, $B_i \sqsubseteq \sigma_{i+1}$ and σ_{i+1} is unifiable with H_{i+1}, therefore B_i is unifiable with H_{i+1}. Furthermore, $B_k \sqsubseteq \sigma_{k+1}$, σ_{k+1} is unifiable with H_1, and therefore B_k is unifiable with H_1. Thus $R_1, \ldots, R_k$ represent a cycle in the URG. Therefore, if a sequence of rules does not form a cycle in the URG, it is not a cyclicly unifiable sequence. □

Lemma 34. *is_cyclicly_unifiable*$(V_1, \ldots, V_k)$ *returns **true** iff $V_1, \ldots, V_k$ is a cyclicly unifiable sequence.*

Proof. If *is_cyclicly_unifiable*$(V_1, \ldots, V_k)$ returns **true**, then all rules $V_1, \ldots, V_k$ have been applied and V_1 may be applied again. The variable FS contains the resulting feature structure after applying each rule. Consider all of FS intermediate values, let FS_i be the value of FS after applying V_i and all its

predecessors. Since FS_k is unifiable with V_1, V_1 may be applied again; let FS_{k+1} be the resulting feature structure. Consider the sequence $\langle \sigma_1, \ldots, \sigma_{k+2} \rangle = \langle \langle [\,] \rangle, \langle FS_1 \rangle, \ldots, \langle FS_{k+1} \rangle \rangle$, for $1 \leq i \leq k$, $\sigma_i \Rightarrow \sigma_{i+1}$ by R_i, and $\sigma_{k+1} \Rightarrow \sigma_{k+2}$ by R_1, therefore, by definition $V_1, \ldots, V_k$ is a cyclicly unifiable sequence.

If *is_cyclicly_unifiable*$(V_1, \ldots, V_k)$ returns **false**, then either there exists some rule V_i, whose head is not unifiable with the resulting feature structure FS, or all rules have been applied and the resulting FS is not unifiable with the head of V_1. Assume that after applying some rules, V_j may not be applied. Since the simulation begins with the most general feature structure, the sequence $\langle \langle [\,] \rangle, \langle FS_1 \rangle, \ldots, \langle FS_{j-1} \rangle \rangle$ is the most general sequence after applying $V_1, \ldots, V_{j-1}$: for any other sequence $\langle \langle FS'_0 \rangle, \langle FS'_1 \rangle, \ldots, \langle FS'_{j-1} \rangle \rangle$ such that each $FS'_{i-1} \Rightarrow FS'_i$ by V_i, each $FS_i \sqsubseteq FS'_i$. Hence, if FS_{j-1} is not unifiable with V_j's head then neither is FS'_{j-1}, and there exists no sequence of MRSs satisfying the constraint. Therefore $V_1, \ldots, V_k$ is not a cyclicly unifiable sequence. □

Theorem 35. *The algorithm returns **true** iff G is OLP_{D_1}.*

Proof. In order to check whether G contains cyclicly unifiable sequences, only unit rules need be considered. By lemma 33, since a cyclicly unifiable sequence is always represented by a cycle in the URG, all cyclicly unifiable sequences are always detected.

On each cycle, *is_cyclicly_unifiable* is applied from each of the cycle's vertices. By lemma 34, the function returns **true** only for cyclicly unifiable sequences. Therefore, if the algorithm returns **true**, then all cycles have been tested and none of their vertices orderings represent a cyclicly unifiable sequence, thus the grammar contains no cyclicly unifiable sequences and is OLP_{D_1}.

If the algorithm returns **false** then *is_cyclicly_unifiable* returned **true** on a set of vertices, by lemma 34, this set represents a cyclicly unifiable sequence, thus the grammar contains at least one cyclicly unifiable sequence and is not OLP_{D_1}. □

6.2.2 Complexity of the algorithm

Assume that a grammar G has n unit rules. Therefore URG contains n vertices and at most n^2 edges. The number of possible cycles of any k vertices (including all cyclic rotations) is bounded by $k! \times \binom{n}{k}$. The number of operations done by *is_cyclicly_unifiable* is linear in k (the number of vertices). Therefore the complexity of the algorithm is in $\Theta(n!)$. As we expect "natural" grammars for natural languages to contain a small number of unit rules, n is expected to be small.

6.3 Evaluation of OLP_{D_1}

In this section we compare OLP_{D_1} with the other constraints discussed above, in order to place it in the hierarchy of figure 10. OLP_{D_1} is applicable to both skeletal and general unification grammars. Ignoring ϵ-rules, it is more liberal than the previously proposed decidable definitions that are limited to skeletal formalisms only, and unlike all definitions that are applicable to general unification grammars, OLP_{D_1} can be tested effectively.

The grammars G_{ww} of figure 1 and G_{abc} of figure 3 are OLP_{D_1}; they include no unit rules, therefore no non-branching dominance chains can be generated. Thus OLP_{D_1} allows non-context-free grammars.

Any $OLP_{JO_\forall}$ grammar is also OLP_{D_1}; since an $OLP_{JO_\forall}$ grammar cannot generate a derivation tree in which the same category appears twice in a non-branching dominance chain, no rule may be applied more than once in a non-branching dominance chain.

An $OLP_{JO_\exists}$ grammar G is not necessarily OLP_{D_1}. In an $OLP_{JO_\exists}$ grammar for every $w \in L(G)$ there exists an OLP derivation tree, but G can still generate non-OLP derivation trees. The grammar G_{inf} of figure 5 is $OLP_{JO_\exists}$, but it is not OLP_{D_1} as its second rule is cyclicly unifiable.

An OLP_{PW} grammar is not necessarily OLP_{D_1} as it may contain ϵ-rules. OLP_{D_1} admits grammars whose c-structure may contain a non-branching dominance chain in which the same category appears twice as long as it is generated by a non-cyclicly unifiable sequence of rules. Furthermore, it does not assume an explicit context-free skeleton. Figure 13 presents an example OLP_{D_1} grammar which is neither OLP_{JO} nor OLP_{PW}.

$$\mathcal{R} = \left\{ \begin{array}{ll} \left[\textsc{f} : s\right] \longrightarrow \left[\textsc{f} : a\right] & S \longrightarrow P \\ \left[\textsc{f} : a\right] \longrightarrow \left[\textsc{f} : b\right] & P \longrightarrow P \end{array} \right\}$$

$$\mathcal{L}(b) = \left\{ \langle \left[\textsc{f} : b\right], P \rangle \right\}$$

Figure 13: An OLP_{D_1} grammar, G_D

The following discussion shows that neither HP nor DB nor FA imply OLP_{D_1}. The grammars G_{FA} of figure 4, which is HP and FA, and G_{inf} of figure 5, which is HP, are not OLP_{D_1}; e.g., by their second rule,

$$\begin{bmatrix} \textsc{cat} : & p \\ \textsc{word} : & \boxed{1} \end{bmatrix} \longrightarrow \begin{bmatrix} \textsc{cat} : & p \\ \textsc{word} : & \langle tb \mid \boxed{1} \rangle \end{bmatrix}$$

and the following set of feature structures:

$$\left\{ \begin{bmatrix} \textsc{cat} : & p \\ \textsc{word} : & \langle tb \rangle \end{bmatrix}, \begin{bmatrix} \textsc{cat} : & p \\ \textsc{word} : & \langle tb, tb \rangle \end{bmatrix}, \begin{bmatrix} \textsc{cat} : & p \\ \textsc{word} : & \langle tb, tb, tb \rangle \end{bmatrix} \right\}$$

The grammar G_{DB} of figure 6 is DB and FA, but it is not OLP_{D_1} as its third rule is cyclicly unifiable. By lemma 30, if G is OLP_{D_1}, then the depth of every derivation tree for a string of n symbols is bounded by a linear function of n, therefore G is also HP, DB and FA.

An OLP_S grammar G is not necessarily OLP_{D_1}. Figure 14 depicts an OLP_S grammar generating the language $\{b^+\}$. A string of n occurrences of b has a derivation tree of depth $3 \times n$. The depth of every non-branching chain is 3, therefore there exists a finite-ranged function F (e.g., mapping each feature structure to itself) such that no two nodes on a derivation tree spanning the same yield are mapped to the same feature structure. The grammar is not OLP_{D_1}, since its first rule may be applied twice consecutively, resulting in a cyclicly unifiable sequence. In section 6.4 we present an improvement to OLP_{D_1} which admits G_S.

OLP_{D_1} is a restriction on derivation trees such that no two nodes on a derivation tree spanning the same yield are unifiable with the same rule's head, whereas OLP_S is a restriction on derivation trees such that no two nodes on a derivation tree spanning the same yield are mapped to the same image. We conjecture that an OLP_{D_1} grammar is not necessarily OLP_S, but we have not been able to come up with a proof yet.

Figure 15 depicts the revised OLP inter-relations hierarchy diagram including OLP_{D_1}.

The class of OLP_{D_1} grammars can never be equal to any of the other OLP classes for general unification grammars. Since the constraints for general unification grammars are undecidable, if any of these classes were equal to the class of OLP_{D_1}, then using the algorithm for deciding OLP_{D_1}, we could also decide whether a grammar satisfies the other constraint which is undecidable.

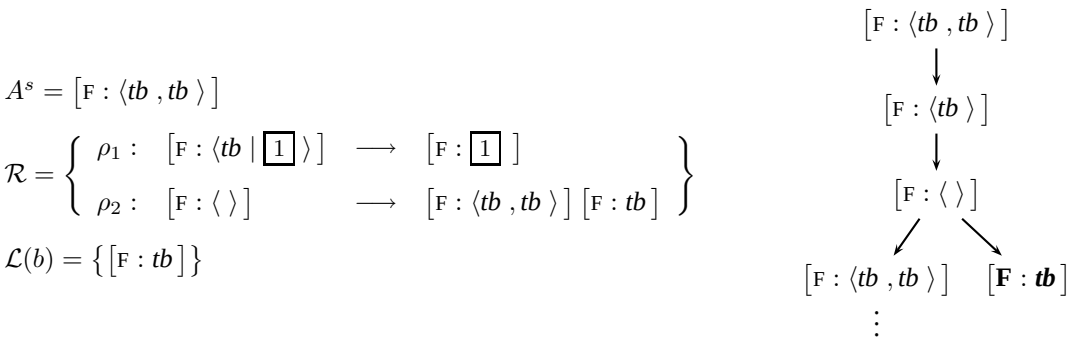


Figure 14: An OLP_S grammar, G_S and its derivation form

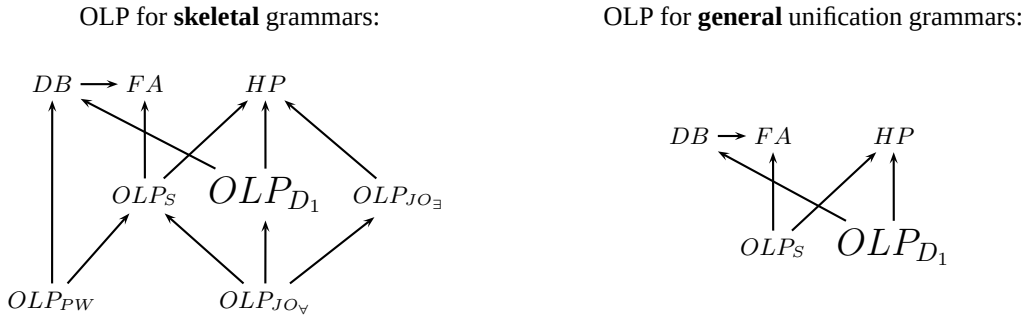


Figure 15: Revised hierarchy diagram with OLP_{D_1}

Assume that a grammar G contains a cyclicly unifiable sequence of rules, $R_1, \ldots, R_k$. Whether any of these rules may ever be applied in any derivation tree admitted by G is not known to the algorithm. Therefore, G is not OLP_{D_1} although recognition may still be decidable for it. OLP_{D_1} does not allow any unit rule sequences in which the same rule may be applied more than once. There might be some unit rule sequences in which at some point, after applying the sequence rules repeatedly several times, unifiability between the resulting feature structure and the head of the next rule may no longer hold, hence the sequence is harmless for decidability. In the next section we propose an improvement of OLP_{D_1} called $OLP_{D\times l}$ which allows such grammars.

6.4 Improvements

OLP_{D_1} does not permit grammars which contain ϵ-rules. As ϵ's play a major role in many natural language descriptions, we present in this section some improvements to OLP_{D_1} allowing ϵ-rules. We show that the improved constraints are more liberal than the existing decidable constraints.

6.4.1 A decidable definition of OLP (version 2)

Let F be the set of all rules heads. Let E' be the set consisting of the heads of all rules that may never derive an ϵ: there exists no sub-derivation tree whose root is an element of E' and whose yield is ϵ. We create a set of *ϵ-derivables*, E, to be the complement of E' consisting of the heads of all the rules that may derive an ϵ: $E = F \setminus E'$.

Definition 14 (ϵ-derivables set, E). *Given a grammar $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$, let E_d be defined as follows:*

- $E_1 = \{A \mid A \Rightarrow \epsilon \in \mathcal{R}\}$
- *For $d > 1$, $E_d = \{A \mid A \Rightarrow A_1 \cdots A_n \in \mathcal{R}$ and there exists a sequence $B_1 \ldots B_n$ such that for each $1 \leq i \leq n$, $B_i \in E_{d_i}$ for some $d_i < d\}$ and $\langle A_1, \ldots, A_n \rangle$ is unifiable with $\langle B_1, \ldots, B_n \rangle$*

Let $E = \bigcup_{1 \leq d \leq |\mathcal{R}|} E_d$.

Lemma 36. *E is finite and can be computed in polynomial time (in $|\mathcal{R}|$).*

Proof. E contains at most all rules' heads, therefore the size of E is at most $|\mathcal{R}|$. Each E_i is computed incrementally beginning with E_1. In order to compute E_i, $i > 1$, all rules' bodies should be checked for unifiability with elements of E_d for $1 \leq d < i$. The union of these sets contains at most $|\mathcal{R}|$ elements. Let l be the maximum rule's length, therefore for each rule, at most $(l-1) \times |\mathcal{R}|$ unifiability checks should be made. Thus each E_i can be computed in at most $(l-1) \times |\mathcal{R}|^2$ steps and therefore E is computed in at most $(l-1) \times |\mathcal{R}|^3$ steps ($(l-1) \times |\mathcal{R}|^2 \times d_{max}$). □

Lemma 37. *If $A \notin E$, where A is some rule's head, then A may never derive the empty string.*

Proof. Assume towards a contradiction that A may derive the empty string. Therefore there exists a derivation tree of some depth n whose root is A and each of its leaves is ϵ. We next prove that each internal vertex on the derivation tree is unifiable with elements of E. We define the internal vertices *level* as follows: the root is on level $l = n$, all internal vertices on depth $n - i$ are on level $l = i$ (a bottom-up view).

The proof is by induction on l, the tree's level.For $l = 1$, since all vertices in level 0 (i.e., the leaves) are ϵ's, all of their mothers are unifiable with ϵ-rule heads. Thus all internal vertices on level 1 are unifiable with elements of E (in fact, of E_1).

Assume that the induction hypothesis holds for all i, $1 \leq i < l$, so that all internal vertices up to level i are unifiable with elements of E. For $i = l$, each node on level l is unifiable with some rule's head, A where $A \Rightarrow A_1, \ldots, A_m \in \mathcal{R}$. By the induction hypothesis, all of $A_1, \ldots, A_m$ are unifiable with elements of E (since they are on level $l - 1$), therefore, by definition, $A \in E$. Thus each internal node on level l is unifiable with elements of E. Therefore all daughters of the root A are unifiable with elements of E, hence $A \in E$, a contradiction. □

In this OLP version, as in the previous one, we want to exclude grammars which generate derivation trees in which the same rule may be applied more than once from two different nodes dominating the same yield. Since a grammar may contain ϵ-rules, it is not enough to search for unit rule chains. Therefore, we reformulate the ϵ-elimination algorithm (Hopcroft and Ullman, 1979, pp. 90–92) for context-free grammars to apply to general unification grammars.

Definition 15. *Given a grammar $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$, $UR(G)$ is the following set of rules:*

- *For each rule $\rho \in \mathcal{R}$, if ρ is a unit rule, then $\rho \in UR(G)$.*
- *For each rule $A \Rightarrow A_1 \ldots A_n \in \mathcal{R}$, if all of the body elements are unifiable with elements of E, then for $1 \leq i \leq n$, $A \Rightarrow A_i \in UR(G)$.*
- *For each rule $A \Rightarrow A_1 \ldots A_n \in \mathcal{R}$, if all of the body elements but one (A_i) are unifiable with elements of E, then $A \Rightarrow A_i \in UR(G)$.*

The purpose of the second clause is to prevent multiple applications of the same rule in a sub-derivation tree whose yield is ϵ (for example, the context-free grammar of figure 16(a)), thus ruling out grammars generating unboundedly deep sub-derivation trees whose yield consists of ϵ's only. The purpose of the third clause is to consider rules which may generate sub-derivation trees whose yield is of length 1 as unit rules, thus preventing multiple applications of the same rule in sub-derivation trees dominating the same yield (For example the context-free grammar of figure 16(b)).

$P \rightarrow PP$	$P \rightarrow PQ$
$P \rightarrow \epsilon$	$P \rightarrow b$
	$Q \rightarrow \epsilon$
(a)	(b)

Figure 16: Motivation for $UR(G)$ rules

Definition 16 (A decidable OLP constraint (OLP_{D_2})). *A grammar G is OLP_{D_2} iff it contains no cyclicly unifiable sequences of $UR(G)$ rules.*

$$A^s = \begin{bmatrix} \text{CAT}: & s \\ \text{WORD}: & \langle s \rangle \end{bmatrix}$$

$$\mathcal{R} = \left\{ \begin{array}{l} \begin{bmatrix} \text{CAT}: & s \\ \text{WORD}: & \langle s \rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \\ \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \boxed{1} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \mid \boxed{1} \rangle \end{bmatrix} \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \\ \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \boxed{1} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \boxed{1} \end{bmatrix} \\ \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \langle tb \mid \boxed{1} \rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \boxed{1} \end{bmatrix} \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \\ \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \longrightarrow \epsilon \end{array} \right\}$$

$$\mathcal{L}(b) = \left\{ \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \right\}$$

Figure 17: Cyclicly unifiability example

Figure 17 shows a grammar example G such that

$$E = \left\{ \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix}, \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & [\,] \end{bmatrix}, \begin{bmatrix} \text{CAT}: & s \\ \text{WORD}: & \langle s \rangle \end{bmatrix} \right\}$$

and $UR(G)$ is listed in figure 18. In this example, ρ_1, ρ_4 and ρ_5 are unit rules and thus belong to $UR(G)$. ρ_2 and ρ_3 are added to $UR(G)$ on account of the grammar's second rule, since both its body elements are ϵ-unifiable (the second bullet of definition 15).

$$UR(G) = \left\{ \begin{array}{ll} \rho_1: & \begin{bmatrix} \text{CAT}: & s \\ \text{WORD}: & \langle s \rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \\ \rho_2: & \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \boxed{1} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \mid \boxed{1} \rangle \end{bmatrix} \\ \rho_3: & \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & [\,] \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \\ \rho_4: & \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \boxed{1} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \boxed{1} \end{bmatrix} \\ \rho_5: & \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \longrightarrow \epsilon \end{array} \right\}$$

Figure 18: $UR(G)$

The sequence $\langle \rho_2 \rangle$ is cyclicly unifiable, for example, by

$$\left\{ \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix}, \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb, tb \rangle \end{bmatrix}, \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb, tb, tb \rangle \end{bmatrix} \right\}$$

The sequence $\langle \rho_3 \rangle$ is also cyclicly unifiable, for example, by

$$\left\{ \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix}, \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix}, \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \right\}$$

Lemma 38. *Let G be an OLP_{D_2} grammar. G does not license any derivation trees in which the same rule is applied more than once from two different nodes dominating the same yield.*

Proof. Assume towards a contradiction that an OLP_{D_2} grammar G can generate a derivation tree which contains a sub-derivation in which the same rule is applied more than once from two nodes dominating the same yield. Let A be the dominating feature structure from which the first application of a rule ρ is applied. Let B be its descendant from which ρ may be applied again dominating the same yield. Such a sub-derivation can result only by consecutive applications of unit rules, ϵ-deriving rules and rules whose body elements (all but one) derive an ϵ (all other rules must expand the derivation's yield).

Let $V_1, \ldots, V_k$ be the applied rules and $FS_1, \ldots, FS_{k-1}$ be the feature structures on the path leading from A to B. We construct the sequence $V'_1, \ldots, V'_k$ as follows, for each $1 \leq i \leq k$:

- If V_i is a unit rule then $V'_i = V_i$.
- If $V_i = A \Rightarrow A_1, \ldots, A_n$, $A \in E$ and FS_i is the j-th daughter on the derivation tree after applying V_i, then $V'_i = A \Rightarrow A_j$.
- If $V_i = A \Rightarrow A_1, \ldots, A_n$, and all of the body elements but A_j are unifiable with elements of E, then $V'_i = A \Rightarrow A_j$ (the derivation step from FS_{i-1} to FS_i must have been by A_j, otherwise since A_j is non ϵ-derivable, A and B would not span the same yield).

Thus each of $V'_1, \ldots, V'_k$ belongs to $UR(G)$ and all of them may be applied consecutively resulting in $FS_1, \ldots, FS_{k-1}, B$. Since V_1 is applied more than once, the head of V'_1 is unifiable with B, hence V'_1 may be applied again resulting in FS. Therefore, the sequence $V'_1, \ldots, V'_k$ is cyclicly unifiable, as evidenced by $A, FS_1, \ldots, FS_{k-1}, B, FS$. Hence G contains a cyclicly unifiable sequence and it is not OLP_{D_2}, a contradiction. □

Lemma 39. *The depth of any OLP_{D_2} derivation tree for a string of n symbols is bounded by $|\mathcal{R}| \times n$.*

Proof. Let G be an OLP_{D_2} grammar. By lemma 38, the maximum depth of a sub-derivation dominating the same yield is bounded by $|\mathcal{R}|$, thus in every derivation tree after at most $|\mathcal{R}|$ derivation steps, in which all nodes dominate the same yield, there must be either a terminating node or an application of a rule expanding the yield. Therefore, in order to generate a string of n symbols, the depth of every derivation tree is at most $|\mathcal{R}| \times n$. □

Corollary 40. *The membership problem is decidable for OLP_{D_2} grammars.*

Proof. From lemma 39 and the bounding lemma. □

Theorem 41. *It is decidable whether a grammar is OLP_{D_2}.*

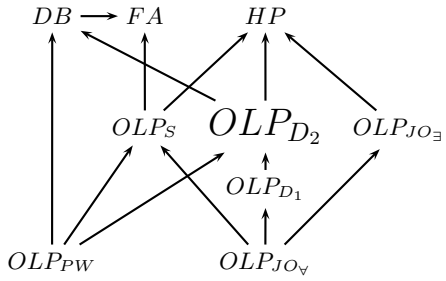
Proof. Since the set $UR(G)$ consists of unit rules only, the algorithm for deciding OLP_{D_1} of section 6.2 can be also used for deciding OLP_{D_2} where each vertex in the URG graph is a $UR(G)$ rule. □

6.4.2 Evaluation of OLP_{D_2}

OLP_{D_2} is more liberal than OLP_{D_1}; since the set of unit rules is a subset of $UR(G)$, any grammar satisfying OLP_{D_1} would also satisfy OLP_{D_2}. Unlike version 1, any OLP_{PW} grammar G is also OLP_{D_2}; assume towards a contradiction that G contains cyclicly unifiable sequences, thus the same rule may be applied more than once from two different nodes dominating the same yield, resulting in an infinitely ambiguous context-free backbone (by lemma 2), a contradiction.

OLP_{D_2} is still not as liberal as the undecidable constraints, but it can be tested efficiently. Figure 19 depicts the revised inter-relations hierarchy diagram of the OLP definitions including OLP_{D_2}.

OLP for **skeletal** grammars:



OLP for **general** unification grammars:

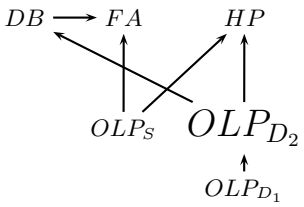


Figure 19: Revised hierarchy diagram, OLP_{D_2}

6.4.3 A decidable definition of OLP (version 3)

We extend the class of OLP_{D_2} grammars by allowing $UR(G)$ rules which may be applied at most a constant number of times. The improved constraint is called $OLP_{D\times l}$, where l is an arbitrary number:

Definition 17. *A sequence of $UR(G)$ rules $R_1, \ldots, R_k$ is* l-cyclicly-unifiable *iff $(R_1, \ldots, R_k)^l$ is cyclicly unifiable.*

Definition 18. *A grammar G is $OLP_{D\times l}$ iff it contains no l-cyclicly-unifiable sequences.*

The grammar of figure 14 is not OLP_{D_2}, but it is $OLP_{D\times 2}$, as its first rule may be applied repeatedly at most twice. Therefore, the sequence $\langle \rho_1, \rho_1 \rangle$ is not cyclicly unifiable.

Decidability of membership is guaranteed for $OLP_{D\times l}$ grammars; since the grammar contains no l-cyclicly-unifiable sequences, the depth of any sub-derivation dominating the same yield is bounded by l times the number of grammar rules (where l is a constant number). Therefore, the depth of every derivation tree whose yield is of length n admitted by an $OLP_{D\times l}$ grammar G is bounded by $(l \times |\mathcal{R}|) \times n$.

$OLP_{D\times l}$ is decidable; the algorithm for deciding OLP_{D_1} can be extended in order to decide $OLP_{D\times l}$, the only difference being (beside using $UR(G)$ instead of the grammar's unit rules) that on each cycle *is_cyclicly_unifiable* is called with some cyclic rotation of $(V_1, \ldots, V_k)^l$. The function *is_cyclicly_unifiable* is unchanged.

7 Conclusions

In this paper (which is a revised, extended version of Jaeger, Francez, and Wintner (2002)) we explore several variants of the OLP constraint, analyze and compare them. We provide proofs of undecidability for four variants. Our main contribution is the definition of a novel OLP constraint which *is* decidable. Our constraint is applicable to both skeletal and general unification grammars. It is more liberal than the existing decidable constraints and, unlike all definitions that are applicable to general unification grammars, it can be tested efficiently. Along with our constraint we also provide an algorithm for deciding whether a grammar satisfies it, as well as an evaluation of our constraint compared with the other OLP variants.

Acknowledgments

The work of Nissim Francez was partially funded by the vice-president's fund for the promotion of research at the Technion. The work of Shuly Wintner was supported by the Israeli Science Foundation (grant no. 136/01).

References

Berwick, Robert C. 1987. Computational complexity, mathematical linguistics, and linguistic theory. In Alexis Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, Amsterdam/Philadelphia, pages 1–17.

Carpenter, Bob. 1992. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Chomsky, Noam. 1975. Remarks on nominalization. In Donald Davidson and Gilbert H. Harman, editors, *The Logic of Grammar*. Dickenson Publishing Co., Encino, California, pages 262–289.

Gazdar, Gerald and Geoffrey K. Pullum. 1985. Computationally relevant properties of natural languages and their grammars. *New Generation Computing*, 3:273–306.

Gazdar, Gerald. E., Ewan Klein, Jeoffrey K. Pullum, and Ivan A. Sag. 1985. *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge, Mass.

Haas, Andrew. 1989. A parsing algorithm for unification grammar. *Computational Linguistics*, 15(4):219–232, December.

Hopcroft, John E. and Jeffrey D. Ullman. 1979. *Introduction to automata theory, languages and computation*. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Company, Reading, Mass.

Jaeger, Efrat, Nissim Francez, and Shuly Wintner. 2002. Guaranteeing parsing termination of unification grammars. In *Proceedgins of COLING'02*, pages 397–403, August.

Johnson, Mark. 1988. *Attribute-Value Logic and the Theory of Grammar*, volume 16 of *CSLI Lecture Notes*. CSLI, Stanford, California.

Kaplan, Ronald and Joan Bresnan. 1982. Lexical functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Mass., pages 173–281.

Kuhn, Jonas. 1999. Towards a simple architecture for the structure-function mapping. In Miriam Butt and Tracy Holloway King, editors, *The Proceedings of the LFG '99 Conference*. CSLI Publications, Stanford.

Pereira, Fernando C. N. and David H. D. Warren. 1983. Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, June.

Shieber, Stuart M. 1992. *Constraint-Based Grammar Formalisms*. MIT Press, Cambridge, Mass.

Torenvliet, Leen and Marten Trautwein. 1995. A note on the complexity of restricted attribute-value grammars. ILLC Research Report and Technical Notes Series CT-95-02, University of Amsterdam, Amsterdam.

Wintner, Shuly and Nissim Francez. 1999. Off-line parsability and the well-foundedness of subsumption. *Journal of Logic, Language and Information*, 8(1):1–16, January.