Unification Grammars and Off-Line Parsability

Efrat Jaeger

October 1, 2002

Unification Grammars and Off-Line Parsability

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Efrat Jaeger

Submitted to the Senate of
the Technion — Israel Institute of Technology

Tishrei 5763 Haifa October 2002

Contents

List of Figures

List of Tables

Abstract

Context-free grammars are considered to lack the expressive power needed for modelling natural languages. Unification grammars have originated as an expansion of context-free grammars, the basic idea being to augment the context-free rules with feature structures in order to express additional information. Unification grammars are known to be Turing-equivalent; given a grammar G and a word w, it is undecidable whether there exists a derivation tree for w admitted by G.

In order to ensure decidability of the membership problem, several constraints on grammars, commonly known as the *off-line parsability constraint (OLP)* were suggested. The membership problem is decidable for grammars which satisfy the OLP constraint. An open question is whether it is decidable if a given grammar satisfies OLP.

In this thesis we research the several different definitions of OLP and discuss their inter-relations. Some variants of OLP were suggested without recognizing the existence of all other variants, we make a comparative analysis of the different OLP variants for the first time. Some researchers conjecture that some of the OLP variants are undecidable (it is undecidable whether a grammar satisfies the constraint), although none of them provides any proof of it. There exist some variants of OLP for which decidability holds, but these conditions are too restrictive; there is a large class of non-OLP grammars for which parsing termination is guaranteed, and they are also limited only to a specific unification grammars formalism.

Our main contribution is to show proofs of undecidability for three of the undecidable OLP variants and give a novel OLP constraint as well as an algorithm for deciding whether a grammar satisfies it. Our constraint is applicable to all unification grammar formalisms. It is more liberal than the existing decidable constraints, yet, it can be tested efficiently.

Chapter 1

Introduction

1.1 Background and Literature Survey

Many modern linguistic theories use *feature structures* to describe linguistic objects representing phonological, morphological, syntactic, and semantic properties. These feature structures are specified in terms of constraints which they must satisfy. Unification is the primary operation for determining the satisfiability of conjunctions of constraints. Unification is based on *subsumption* of feature structures, where subsumption is a partial pre-order on feature structures which expresses whether two feature structures are consistent and whether one structure contains more specific information than the other.

Context-free grammars are considered to lack the expressive power for modelling natural languages. Unification grammars have originated as an expansion to context-free grammars, the basic idea being to augment the context-free rules with non context-free notations (feature structures) in order to express some additional information, such that the constraints of the grammar will be sensitive to these features. Today, several formalisms of unification grammars exist, some of which do not necessarily assume an explicit context-free backbone.

Unification grammar formalisms are based on unification of feature structures. The unification operation takes two feature structures and combines the information contained in them to produce a feature structure that includes all the shared information of both. If they contain incompatible information the unification operation fails. Non-failing unification returns the least upper bound *(lub)* of its arguments (in terms of the subsumption

ordering).

A detailed description of unification grammars is given by Shieber (1986; 1992) and Carpenter (1992). The grammar formalisms used here are based on the description of Francez and Wintner (In preperation).

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 deliver all parse trees that G induces on w, determining what structural descriptions are assigned by G to w.

The recognition problem for unification grammars is undecidable. Johnson (1988) shows that the problem is undecidable by using a reduction from the halting problem of Turing machines; a given string is generated by a given grammar if and only if a Turing machine halts on an empty input. Since determining whether an arbitrary Turing machine halts on an empty input is undecidable it follows that the recognition problem is also undecidable.

In order to to ensure decidability of the recognition problem, a constraint called the *off-line parsability constraint (OLP)* was suggested. The recognition problem is decidable for off-line parsable grammars. There exist several variants of OLP in the literature (Pereira and Warren, 1983; Johnson, 1988; Haas, 1989; Torenvliet and Trautwein, 1995; Shieber, 1992; Wintner and Francez, 1999; Kuhn, 1999). Some variants are applicable only to skeletal grammars formalisms others are applicable to all unification grammar formalisms.

The main open question of this research is, whether it can be determined if a grammar satisfies the OLP constraint. Some researchers (Haas, 1989; Torenvliet and Trautwein, 1995) conjecture that some of the OLP variants are undecidable, although none of them gives any proof of it. There exist some variants of OLP for which decidability holds, but these conditions are too restrictive; there is a large class of non-OLP grammars for which parsing termination is guaranteed.

1.2 Motivation

Unification grammars are linguistically plausible for modelling natural languages. The recognition problem is decidable for off-line parsable grammars. Several variants of the OLP constraint were suggested, some of which do not assume the existence of other vari-

ants. No one ever made a comparison of these variants. We explore all of the conditions and discuss their properties in order to search for the relationships between these constraints and find some hierarchy among them for the first time.

It is well known that for OLP grammars decidability of the membership problem is guaranteed, but can it be determined whether a grammar satisfies the OLP constraint? Some researchers (Haas, 1989; Torenvliet and Trautwein, 1995) conjecture that some of the OLP variants are undecidable (it is undecidable whether a grammar satisfies the constraint), but there exists no proof of it. There exist some variants of OLP for which decidability holds, but these conditions are limited only to a certain formalism of unification grammars. Our main contribution is to show proofs of undecidability for three of the undecidable OLP definitions and provide a novel OLP constraint, a decidable constraint, which is more liberal than the existing decidable constraints and applies to all unification grammar formalisms.

Chapter 2

Unification Grammars

2.1 Preliminaries

The following definitions are based on Carpenter (1992) and Wintner and Francez (1999) and will be used later for defining general and skeletal unification grammars. Both grammar formalisms are defined over a finite set FEATS of features and a finite set ATOMS of atom values. Skeletal grammars are defined over an additional parameter, a finite set CATS of categories.

Definition 2.1. *A **feature structure** (FS) $fs = \langle Q, \bar{q}, \delta, \theta \rangle$ is a directed, connected, labelled, rooted graph consisting of a finite, nonempty set of nodes Q, a root $\bar{q} \in Q$, a partial function, $\delta : Q \times \text{FEATS} \rightarrow Q$ specifying the arcs, such that every node $q \in Q$ is accessible from $\bar{q}$, and a partial function, marking some of the sinks, $\theta : Q_s \rightarrow \text{ATOMS}$, where $Q_s = \{q \in Q \mid \delta(q, f)\uparrow \text{ for every } f \in \text{FEATS}\}$ (the nodes for which δ is undefined for all features). Meta-variables A, B (with or without subscripts) range over feature structures.*

Definition 2.2. *A **multi-rooted feature structure** (MRS) is a pair $\langle \bar{Q}, G \rangle$ where:*

- *$G = \langle Q, \delta, \theta \rangle$ is a finite directed, labelled graph consisting of a set, $Q \subseteq \text{NODES}$, of nodes, a partial function, $\delta : Q \times \text{FEATS} \rightarrow Q$, specifying the arcs and a partial function, $\theta : Q \rightarrow Atom$, labelling the sinks.*
- *$\bar{Q}$ is an ordered set of distinguished nodes in Q called **roots**.*

G is not necessarily connected, but the union of all the nodes reachable from all the roots

in $\bar{Q}$ is required to yield exactly Q. The ***length*** *of an MRS is the number of its roots, $|\bar{Q}|$. λ denotes the empty MRS, where $Q = \emptyset$.*

Meta-variables σ, ρ range over MRSs. If $\sigma = \langle \bar{Q}, G \rangle$ is an MRS, and $\bar{q}_i$ is a root in $\bar{Q}$ then $\bar{q}_i$ naturally induces a feature structure $A_i = \langle Q_i, \bar{q}_i, \delta_i, \theta_i \rangle$, where Q_i is the set of nodes reachable from $\bar{q}_i$, $\delta_i = \delta|_{Q_i}$ and $\theta_i = \theta|_{Q_i}$. Thus, σ can be viewed as an ordered sequence $\langle A_1, \ldots, A_n \rangle$ of (not necessarily disjoint) feature structures.

The **sub-structure** of $\sigma = \langle A_1, \ldots, A_n \rangle$, induced by the pair $\langle i, j \rangle$ and denoted $\sigma^{i \ldots j}$, is $\langle A_i, \ldots, A_j \rangle$. If $i > j$, $\sigma^{i \ldots j} = \lambda$. If $i = j$, σ^i is used for $\sigma^{i \ldots i}$.

Definition 2.3. *A* ***path*** *is a finite sequence of features; ϵ is the empty path, and we use π (with or without subscripts) to range over paths. The definition of δ is extended to paths in the natural way: $\delta(q, \epsilon) = q$ and $\delta(q, f\pi) = \delta(\delta(q, f), \pi)$. The value of a path $\pi \in \textsc{Feats}^*$ leaving the i-th root in an MRS σ, denoted by $val(\sigma^i, \pi)$, is $\delta(\bar{q}_i, \pi)$.*

Definition 2.4. *An MRS is* ***reentrant****, denoted by $\boxed{i}$ (where i is some natural number), iff either of the below conditions hold:*

- *there exist two different paths $\pi_1, \pi_2 \in \textsc{Feats}^*$, and a root $\bar{q}_i \in \bar{Q}$ such that $\delta(\bar{q}_i, \pi_1) = \delta(\bar{q}_i, \pi_2)$, implying $val(\sigma^i, \pi_1) = val(\sigma^i, \pi_2)$.*
- *there exist two paths $\pi_1, \pi_2 \in \textsc{Feats}^*$ (not necessarily disjoint) and two different roots $\bar{q}_i, \bar{q}_j \in \bar{Q}$ such that $\delta(\bar{q}_i, \pi_1) = \delta(\bar{q}_j, \pi_2)$, implying $val(\sigma^i, \pi_1) = val(\sigma^j, \pi_2)$.*

Definition 2.5. *An MRS $\sigma = \langle \bar{Q}, G \rangle$* ***subsumes*** *an MRS $\sigma' = \langle \bar{Q}', G' \rangle$ (denoted by $\sigma \sqsubseteq \sigma'$) if $|\bar{Q}| = |\bar{Q}'|$ and there exists a total function $h : Q \to Q'$ such that:*

- *for every root $\bar{q}_i \in \bar{Q}, h(\bar{q}_i) = \bar{q}'_i$*
- *for every $q \in Q$ and $f \in \textsc{Feats}$, if $\delta(q, f)\downarrow$ then $h(\delta(q, f)) = \delta'(h(q), f)$*
- *for every $q \in Q$ if $\theta(q)\downarrow$ then $\theta(q) = \theta'(h(q))$*

2.2 General Unification Grammars

Unification grammars have originated as an expansion to context-free grammars, although not all unification grammars formalisms assume the existence of a context-free backbone.

General unification grammar formalisms do not necessarily assume the existence of a context-free backbone.

Definition 2.6. *A **general unification grammar** (over* FEATS *and* ATOMS*) is a tuple* $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$ *where:*

- $\mathcal{R}$ *is a finite set of rules, where each rule is an MRS of length* $n \geq 1$*. The first element (a feature structure) in each rule is the head of the rule, the rest, is the body of the rule, the head is separated from the body by a right arrow* ($\rightarrow$)*.*
- $\mathcal{L}$ *is a lexicon, which associates with every terminal* a *(over a fixed finite set* Σ *of terminal words) a finite set of feature structures,* $\mathcal{L}(a)$*.*
- A^s *is the start symbol (a feature structure).*

Definition 2.7 (Derivation). *An MRS* $\sigma_A = \langle A_1, \ldots, A_k \rangle$ ***immediately derives*** *an MRS* $\sigma_B = \langle B_1, \ldots, B_m \rangle$ *(denoted* $\sigma_A \rightarrow \sigma_B$*) iff there exist a rule* $\rho' \in \mathcal{R}$ *of length* n *and an MRS* ρ*,* $\rho' \sqsubseteq \rho$*, such that:*

- $m = k + n - 2$*;*
- ρ*'s head is some element* i *of* σ_A*:* $\rho^1 = \sigma_A^i$*;*
- ρ*'s body is a sub-structure of* σ_B*:* $\rho^{2\ldots n} = \sigma_B^{i\ldots i+n-2}$*;*
- *The first* $i-1$ *elements of* σ_A *and* σ_B *are identical:* $\sigma_A^{1\ldots i-1} = \sigma_B^{1\ldots i-1}$*;*
- *The last* $k-i$ *elements of* σ_A *and* σ_B *are identical:* $\sigma_A^{i+1\ldots k} = \sigma_B^{m-(k-i+1)\ldots m}$*.*

The reflexive transitive closure of '$\rightarrow$*' is denoted '*$\stackrel{*}{\rightarrow}$*'.*

An MRS σ **derives** *an MRS* ρ *(denoted* $\sigma \stackrel{*}{\Rightarrow} \rho$*) iff there exist MRSs* σ', ρ' *such that* $\sigma \sqsubseteq \sigma'$*,* $\rho \sqsubseteq \rho'$ *and* $\sigma' \stackrel{*}{\rightarrow} \rho'$*.*

Definition 2.8 (Pre-terminals). *Let* $w = a_1 \cdots a_n \in \Sigma^*$*.* $PT_w(j,k)$ *is defined if* $1 \leq j, k \leq n$*, in which case it is the MRS* $\langle A_j, A_{j+1}, \ldots, A_k \rangle$ *where* $A_i \in \mathcal{L}(a_i)$ *for* $j \leq i \leq k$*. If* $j > k$ *then* $PT_w(j,k) = \lambda$*.*

Definition 2.9 (Language). *The **language** of a unification grammar* G *is* $L(G) = \{w \in \Sigma^* \mid w = a_1 \cdots a_n$ *and* $\langle A^s \rangle \stackrel{*}{\Rightarrow} \langle A_1, \ldots, A_n \rangle \}$*, where* $A_i \in \mathcal{L}(a_i)$ *for* $1 \leq i \leq n$*.*

Figure 2.1 lists an example unification grammar, G_{ww}, where $L(G_{ww}) = \{ww \mid w \in \{a|b\}^*\}$. The reentrancy in the first rule should guarantee that both branching nodes are mapped by δ to the same node using the feature WORD. The second rule removes items from a WORD list. Therefore, in order to apply the second rule, applying the first rule should result in a feature structure containing a non-empty WORD list.

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

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

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

Figure 2.1: An example unification grammar, G_{ww}

Definition 2.10 (Derivation trees). *Let $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$ be a unification grammar. A tree is a derivation (or parse) tree admitted by G iff:*

- *The root of the tree is the start symbol A^s;*
- *The internal vertices are feature structures (over the same features and atoms as the grammar G);*
- *The leaves are pre-terminals of length 1;*
- *If a vertex A has k descendants, $B_1, B_2, \ldots, B_k$, then $\langle A \rangle$ immediately derives $\langle B_1, \ldots, B_k \rangle$ with respect to some rule $\rho \in \mathcal{R}$.*

Figure 2.2 lists an example derivation tree for the string *baabaa* generated by the unification grammar, G_{ww}, of figure 2.1: at the first derivation depth, the initial symbol is unifiable with the first rule's head resulting in,

$$\langle \left[\text{WORD} : \begin{bmatrix}\text{HD} : s \\ \text{TL} : \mathit{elist}\end{bmatrix}\right] \left[\text{WORD} : \boxed{1}\right] \left[\text{WORD} : \boxed{1}\right] \rangle \sqsubseteq$$

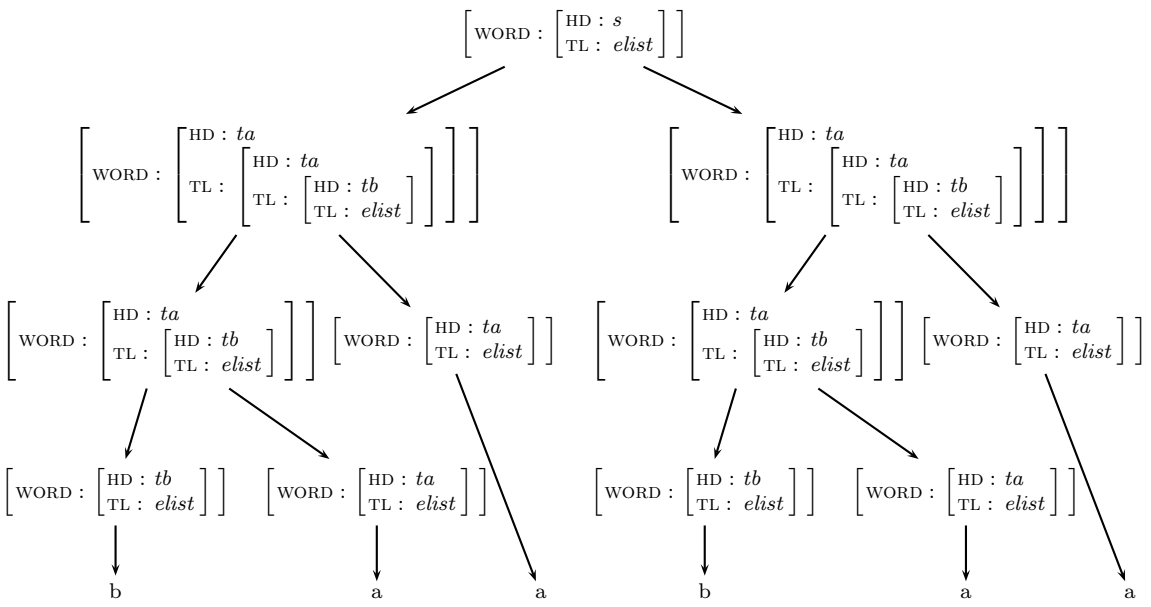


Figure 2.2: An example parse tree for $baabaa \in G_{ww}$

$$\langle \left[\text{WORD} : \left[\begin{matrix}\text{HD} : s \\ \text{TL} : \mathit{elist}\end{matrix}\right]\right] \left[\text{WORD} : \left[\begin{matrix}\text{HD} : ta \\ \text{TL} : \left[\begin{matrix}\text{HD} : ta \\ \text{TL} : \left[\text{HD} : tb \; \text{TL} : \mathit{elist}\right]\end{matrix}\right]\end{matrix}\right]\right] \left[\text{WORD} : \left[\begin{matrix}\text{HD} : ta \\ \text{TL} : \left[\begin{matrix}\text{HD} : ta \\ \text{TL} : \left[\text{HD} : tb \; \text{TL} : \mathit{elist}\right]\end{matrix}\right]\end{matrix}\right]\right] \rangle$$

then, each of the two non-empty lists feature structures are unifiable with the second rule's head until all lexical symbols are generated.

The expressive power of unification grammars: unification grammars are equivalent to Turing machines in their generative capacity. Determining whether a given string w is generated by a given grammar G is equivalent to deciding whether a Turing machine M halts on an empty input. Since the latter is undecidable, so is the recognition problem (Johnson, 1988).

General unification grammars are pure feature structure grammars. A variety of grammar formalisms which are based on feature structure unification exist, such as PATR-II (Shieber, 1986), Functional Unification Grammar, FUG (Kay,) , Definite Clause Grammar, DCG (Pereira and Warren, 1980), Lexical Functional Grammar, LFG (Kaplan and Bresnan, 1982), Generalized Phrase Structure Grammar, GPSG (Gazdar et al., 1985) and

Head-Driven Phrase Structure Grammar, HPSG (Pollard and Sag, 1986).

2.3 Skeletal grammars

Skeletal grammars are a variant of unification grammars which have an explicit *context-free backbone/skeleton*. These grammars can be viewed as an expansion of context-free grammars, where every category is associated with an informative feature structure. The **context-free backbone** of a skeletal grammar is obtained by ignoring all feature structures of the grammar rules and considering only the categories.

An **extended category** is a pair $\langle A, c\rangle$ where A is a feature structure and $c \in \textsc{Cats}$ is a category. An extended category $\langle A, c\rangle$ can be rewritten as a pure feature structure by **internalizing** the category into the feature structure, which is done by adding a new atom value c to Atoms and a new feature $\textsc{cat} \notin \textsc{Feats}$ to the set of features Feats, such that for every feature structure A, $\delta(A, \textsc{cat})$ is a sink and $\theta(\delta(A, \textsc{cat})) = c$
For example, $\langle \left[\textsc{word} : W\right], c\rangle$ is internalized into $\begin{bmatrix}\textsc{cat} : & c\\ \textsc{word} : & W\end{bmatrix}$

Definition 2.11. *A **skeletal grammar** (over* Feats, Atoms *and* Cats*) is a tuple* $G = \langle \mathcal{R}, \mathcal{L}, A^s\rangle$ *where:*

- $\mathcal{R}$ *is a finite set of rules, where each rule consists of an MRS of length* $n \geq 1$ *(the first element is 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 *(over a fixed finite set* Σ *of terminal words) a finite set* $\mathcal{L}(a)$ *of extended categories* $\langle A, C\rangle$*, where* A *is a feature structure and* $C \in \textsc{Cats}$ *is a category.*
- $A^s = \langle A, S\rangle$*, is the start symbol (an extended initial category).*

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 $1 \leq i \leq n$).

Definition 2.12 (Derivation). *Let* $\langle \sigma_A, \vec{c_A}\rangle$ *and* $\langle \sigma_B, \vec{c_B}\rangle$ *be forms such that* $\sigma_A = \langle A_1, \ldots, A_k\rangle$ *and* $\sigma_B = \langle B_1, \ldots, B_m\rangle$*.* $\langle \sigma_A, \vec{c_A}\rangle$ ***immediately derives*** $\langle \sigma_B, \vec{c_B}\rangle$ *iff there exist a skeletal rule* $\langle \rho', \vec{c_R}\rangle \in \mathcal{R}$ *of length* n*, an MRS* ρ*,* $\rho' \sqsubseteq \rho$*, such that:*

- $m = k + n - 2$;
- *ρ's head is some element i of σ_A:* $\rho^1 = \sigma_A^i$;
- *ρ's body is a sub-structure of σ_B:* $\rho^{2\ldots n} = \sigma_B^{i\ldots i+n-2}$;
- *The first $i-1$ elements of σ_A and σ_B are identical:* $\sigma_A^{1\ldots i-1} = \sigma_B^{1\ldots i-1}$;
- *The last $k-i$ elements of σ_A and σ_B are identical:* $\sigma_A^{i+1\ldots k} = \sigma_B^{m-(k-i+1)\ldots m}$;
- $\vec{c_B}$ *is obtained by replacing the i-th element of $\vec{c_A}$ by the body of $\vec{c_R}$.*

The reflexive transitive closure of '$\rightarrow$' is denoted '$\overset{*}{\rightarrow}$'.

A form $\langle \sigma_A, \vec{c_A} \rangle$ ***derives*** *$\langle \sigma_B, \vec{c_B} \rangle$ (denoted $\langle \sigma_A, \vec{c_A} \rangle \overset{*}{\Rightarrow} \langle \sigma_B, \vec{c_B} \rangle$) iff there exist MRSs σ'_A, σ'_B such that $\sigma_A \sqsubseteq \sigma'_A$, $\sigma_B \sqsubseteq \sigma'_B$ and $\langle \sigma'_A, \vec{c_A} \rangle \overset{*}{\rightarrow} \langle \sigma'_B, \vec{c_B} \rangle$.*

Definition 2.13 (Pre-terminals). *Let $w = a_1 \cdots a_n \in \Sigma^*$. $PT_w(j,k)$ is defined if $1 \leq j \leq k \leq n$, in which case it is the skeletal form,*
$\langle \langle A_j, A_{j+1}, \ldots, A_k \rangle, \langle C_j, C_{j+1}, \ldots, C_k \rangle \rangle$ where $\langle A_i, C_i \rangle \in \mathcal{L}(a_i)$ for $j \leq i \leq k$.

Definition 2.14 (Language). *The* ***language*** *of a skeletal grammar G is $L(G) = \{ w \in \Sigma^* \mid w = a_1 \cdots a_n$ and $\langle A^s \rangle \overset{*}{\Rightarrow} \langle \langle A_1, \ldots, A_n \rangle, \langle C_1, \ldots, C_n \rangle \rangle \}$, where $\langle A_i, C_i \rangle \in \mathcal{L}(a_i)$ for $1 \leq i \leq n$.*

Figure 2.3 lists an example skeletal grammar, G_{abc}, where $L(G_{abc}) = \{a^n b^n c^n \mid n > 0\}$.

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

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

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

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

Figure 2.3: An example skeletal grammar, G_{abc}

A derivation tree for skeletal grammar (also called a constituent-structure) is defined similarly to definition 2.10 of derivation trees for general unification grammars, except that every vertex consists of an extended category instead of a feature structure.

Definition 2.15 (Constituent structures (c-structures)). *Let $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$ be a skeletal grammar. A tree is a derivation tree admitted by G if:*

- *The root of the tree is the start symbol $A^s = \langle A, S \rangle$;*
- *The internal vertices are extended categories (over the same features, atoms and categories as the Grammar G);*
- *The leaves are pre-terminals of length* 1*;;*
- *If a vertex $\langle A, c \rangle$ has k descendants, $\langle B_1, c_1 \rangle, \langle B_2, c_2 \rangle, \ldots, \langle B_k, c_k \rangle$, then $\langle \langle A \rangle, \langle c \rangle \rangle$ immediately derives $\langle \langle B_1, \ldots, B_k \rangle, \langle c_1, \ldots, c_k \rangle \rangle$ with respect to some rule $\langle \rho, \vec{c_R} \rangle \in \mathcal{R}$.*

2.4 Comparison between General and Skeletal Unification Grammars

Any skeletal unification grammar can be represented as a general unification grammar by internalizing the categories into the feature structures as described in section 2.3.

General unification grammars do not necessarily assume an explicit context-free backbone. There exist no algorithm for obtaining a context-free backbone from any general unification grammar.

Constituent coordination cannot be represented by skeletal grammars; a conjunction /* CONTINUE */

2.5 Some grammar examples:

In this section we give some grammar examples which will be used in the rest of the thesis In order to simplify the examples and avoid the usage of complicated long lists' feature structure, the examples use a straightforward encoding of lists as features structures, where a list is represented by brackets, the 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*. E.g., (a transformation from a list feature structure to the brackets notation):

$$\left[\text{WORD} : \left[\begin{array}{l}\text{HD} : tb \\ \text{TL} : \left[\begin{array}{l}\text{HD} : tb \\ \text{TL} : \left[\begin{array}{l}\text{HD} : tb \\ \text{TL} : \; elist\end{array}\right]\end{array}\right]\end{array}\right]\right] \Rightarrow \left[\text{WORD} : \langle tb\,,\ tb\,,\ tb\,\rangle\right]$$

Figure 2.4: A unification grammar generating the language $\{b^+\}$. The feature CAT stands for category, and WORD is a list of lexical symbols. The string b is the only terminal item at the lexicon, therefore every string generated by the grammar consists of b's only. The grammar is unambiguous; a string of N occurrences of b has just one parse tree. The second rule adds items to a list at each derivation step. The fourth rule removes items from the list. With each removal derivation step a b is generated until a list of one item is reached. Once the third rule has been applied, the second rule may no longer be applied, thus no more items may be added to the list. A derivation tree for a string of N occurrences of b has a linear depth of $2N$.

Lemma 2.1. *There exist a derivation tree for the string b^l, ($l > 0$) of depth $2l$, in which the first rule is applied once, then the second rule is applied $l-1$ times resulting in a* WORD *list of l items, then an application of the third rule and then $l-1$ application of the fourth rule, each application of the fourth rule removes an item from the* WORD *list and a b is generated until a list of one item is reached.*

Proof. The proof is by induction on l the size of the derived string.

For $l = 1$, figure 2.6 lists an example derivation tree for the string b satisfying the conditions.

Assuming that the induction hypothesis holds for $l \leq l - 1$.

The induction step, $l = l$ ($l > 1$): by the induction hypothesis there exists a derivation tree for the string b^{l-1} in which the second rule is applied $l-2$ times, resulting in a list of $l-1$ items, applying the second rule once more would result in a list of l items. By the induction hypothesis at each application of the fourth rule an item is removed from the list and a b is generated. Thus applying the fourth rule $l-2$ times would result in $l-2$ b's and a list of two items. Since a b is mapped at the lexicon to the category Q ([CAT : q]) and a list of one item, one more application of the fourth rule would result in two more b's. Hence, in order to generate the string b^l, the first rule is applied once, the second rule

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

$$\mathcal{R} = \left\{ \begin{array}{ll} (1) & \begin{bmatrix} \text{CAT}: & s \\ \text{WORD}: & \langle s \rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \\ (2) & \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \boxed{1} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \mid \boxed{1} \rangle \end{bmatrix} \\ (3) & \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \boxed{1} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \boxed{1} \end{bmatrix} \\ (4) & \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \langle tb \mid \boxed{1} \rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \end{array} \right\}$$

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

Figure 2.4: A unification grammar, G_{FA}.

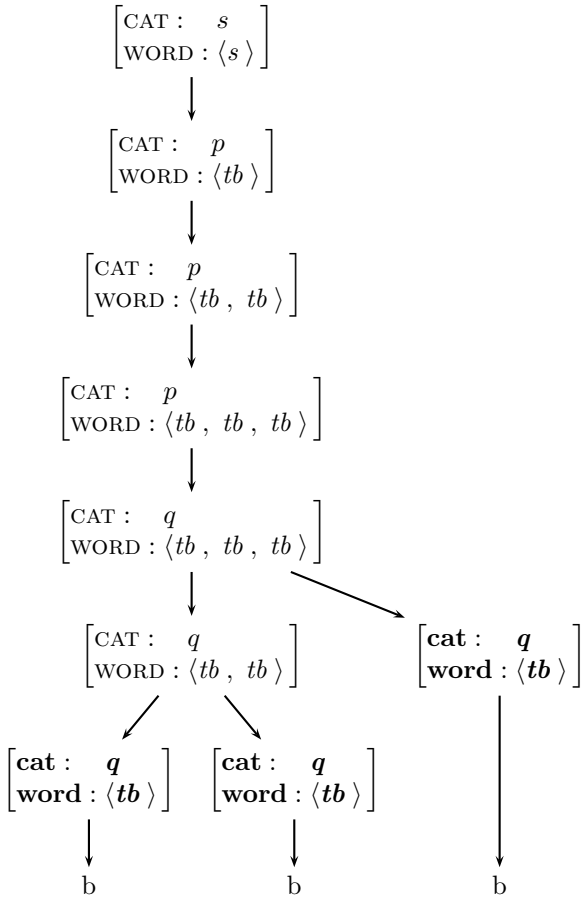


Figure 2.5: An example parse tree for the grammar G_{FA} of figure 2.4 and the string *bbb*.

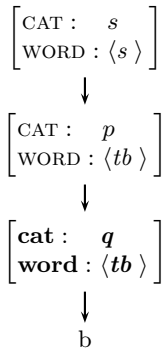


Figure 2.6: A derivation tree admitted by the grammar G_{FA} for the string b.

is applied $l-1$ times, resulting in a list of l items, the third rule is a applied once and then the fourth rule is applied $l-1$ times. □

Corollary 2.2. *The grammar G_{FA} generates the language $L = \{b^l \mid l > 0\}$*

Proof. Since the string b is the only terminal item at the lexicon, and the grammar contains no ϵ-rules, any derived string consists of b's only. By lemma 2.1, there exist a derivation tree for the string b^l for any $l > 0$. □

By the value of the feature CAT, the grammar rules must be applied according to their order. The first rule is applied at most once (in fact, exactly once, since the initial symbol's category is S), then since the current category is P, either the second or third rule may be applied. The second rule may be applied any number of times, adding an item to the list at each derivation step, but once the third rule has been applied, the second rule is no longer applicable since the current category is Q. The third rule is applied exactly once, and then the only applicable rule is the fourth rule. The fourth rule is applied repeatedly until a list of one item is reached, generating a lexical item at each derivation step. Therefore, the only possible derivation for the string b^l is by one application of the first rule, followed by $l-1$ consecutive applications of the second rule, followed by one application of the third rule and then $l-1$ applications of the fourth rule. Thus, a string of l occurrences of b has just one parse tree.

Figures 2.7: A unification grammar generating the language $\{b\}$. The feature CAT stands for category, and WORD is a list of lexical symbols. The second rule adds items to a

list at each derivation step. The fourth rule removes items from the list at each derivation step. There exist infinitely many derivation trees, of arbitrary depths, for the string b of the form listed in figure 2.8.

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

$$\mathcal{R} = \left\{ \begin{array}{ll} (1) & \begin{bmatrix} \text{CAT}: & s \\ \text{WORD}: & \langle s \rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \rangle \end{bmatrix} \\ (2) & \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \boxed{1} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \langle tb \mid \boxed{1} \rangle \end{bmatrix} \\ (3) & \begin{bmatrix} \text{CAT}: & p \\ \text{WORD}: & \boxed{1} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \boxed{1} \end{bmatrix} \\ (4) & \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \langle tb \mid \boxed{1} \rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & q \\ \text{WORD}: & \boxed{1} \end{bmatrix} \end{array} \right\}$$

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

Figure 2.7: A unification grammar, G_{inf}.

Lemma 2.3. *The grammar G_{inf} generates the language $L = \{b\}$.*

Proof. The grammar consists of unit-rules only, therefore it can only generate non-branching derivation trees whose frontier consists of one symbol. Since the string b is the only terminal item at the lexicon, and the grammar contains no ϵ-rules, any derived string consists of b's only.

Figure 2.9 lists a derivation tree for the string b, therefore $L(G_{inf}) = \{b\}$. □

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 WORD list, the fourth rule removes items from the list, a b is generated once the list consists of one item. Therefore there exist infinitely derivation trees for the string b for any natural number of applications of the second rule.

Figure 2.10: A unification grammar generating the language $\{b^+\}$. The string b is the only terminal item at the lexicon and there exist no ϵ-rules, therefore every string generated by the grammar consists of b's only. A string of N occurrences of b has exactly

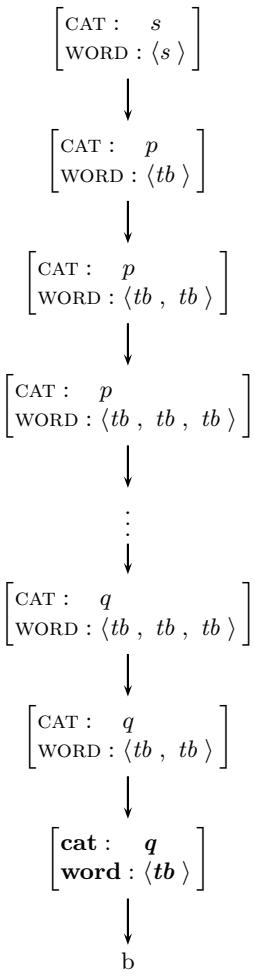
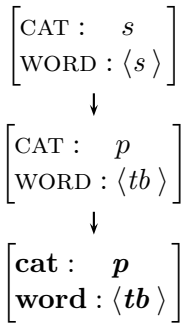


Figure 2.8: The derivation tree form of the grammar G_{inf} of figure 2.7.



b

Figure 2.9: An example derivation tree for the grammar of figure 2.7 and the string b.

one parse tree. The depth of the derivation tree is 2^N. The feature depthCount is a list that represents the current depth of the derivation tree; the number of derivation steps from the root. At 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. The third rule removes items from the innerCount list until an empty list is reached. Then another b is generated by either applying the fourth rule and terminate, or applying the second rule and refill the innerCount list. Figure 2.11 lists a derivation trees example.

Lemma 2.4. *There exists a derivation tree for the string b^l, $l > 1$ of depth 2^l.*

Proof. The proof is by induction on l, the number of b lexical symbols:

For $l = 1$, figure 2.12 lists an example derivation tree for the string b satisfying the conditions.

Assuming that the induction hypothesis holds for $l \leq l - 1$.

The induction step, $l = l$, by the induction hypothesis, a string of $l - 1$ occurrences of b has a derivation tree of depth 2^{l-1}.

By the lexicon, a b is generated once the innerCount list is empty. Thus the only possibilities for generating a b is by applying either the second or fourth (terminating) rules. Therefore, it can be deduced that the $(l-1)^{th}$ b was generated by the second rule.

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

$$\mathcal{R} = \left\{ \begin{array}{ll} (1) & \begin{bmatrix} \text{CAT}: & s \\ depthCount: & \langle\,\rangle \\ innerCount: & \langle\,\rangle \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ depthCount: & \langle tb\,\rangle \\ innerCount: & \langle\,\rangle \end{bmatrix} \\ (2) & \begin{bmatrix} \text{CAT}: & p \\ depthCount: & \boxed{1} \\ innerCount: & \langle\,\rangle \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ depthCount: & \langle tb \mid \boxed{1}\,\rangle \\ innerCount: & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{CAT}: & l \\ lex: & tb \end{bmatrix} \\ (3) & \begin{bmatrix} \text{CAT}: & p \\ depthCount: & \boxed{1} \\ innerCount: & \langle tb \mid \boxed{2}\,\rangle \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ depthCount: & \langle tb \mid \boxed{1}\,\rangle \\ innerCount: & \boxed{2} \end{bmatrix} \\ (4) & \begin{bmatrix} \text{CAT}: & p \\ depthCount: & \langle tb \mid \boxed{1}\,\rangle \\ innerCount: & \langle\,\rangle \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & l \\ lex: & tb \end{bmatrix} \end{array} \right\}$$

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

Figure 2.10: A unification grammar, G_{DB}.

By the induction hypothesis, before the immediate derivation of the $(l-1)^{th}$ symbol, the depthCount list contained $2^{l-1}-1$ items (since depthCount list represents a counter of the derivation's depth). While generating the $(l-1)^{th}$ symbol by an application of the second rule, the depthCount list was assigned into the innerCount list, therefore the innerCount list contains $2^{l-1}-1$ items.

After an application of the second rule (generating the $(l-1)^{th}$ symbol), since innerCount list is non-empty, the only applicable rule is the third rule. Then, $|innerCount|$ $(= 2^{l-1}-1)$ applications of the third rule must be applied until the innerCount list is empty.

Then one application of the fourth rule in order to generate the l^{th} terminating symbol.

There exist 2^{l-1} derivation steps for generating the first $l-1$ symbols, and 2^{l-1} derivation steps for generating the l^{th} symbol. Therefore, there exist a derivation tree for the string b^l of depth 2^l. □

Corollary 2.5. *The grammar G_{DB} generates the language $L = \{b^l \mid l > 0\}$*

The grammar derivation steps are as follows:

1. The first derivation step (and the only possible one) is by the first rule, adding one item to depthCount list, leaving the innerCount list empty.

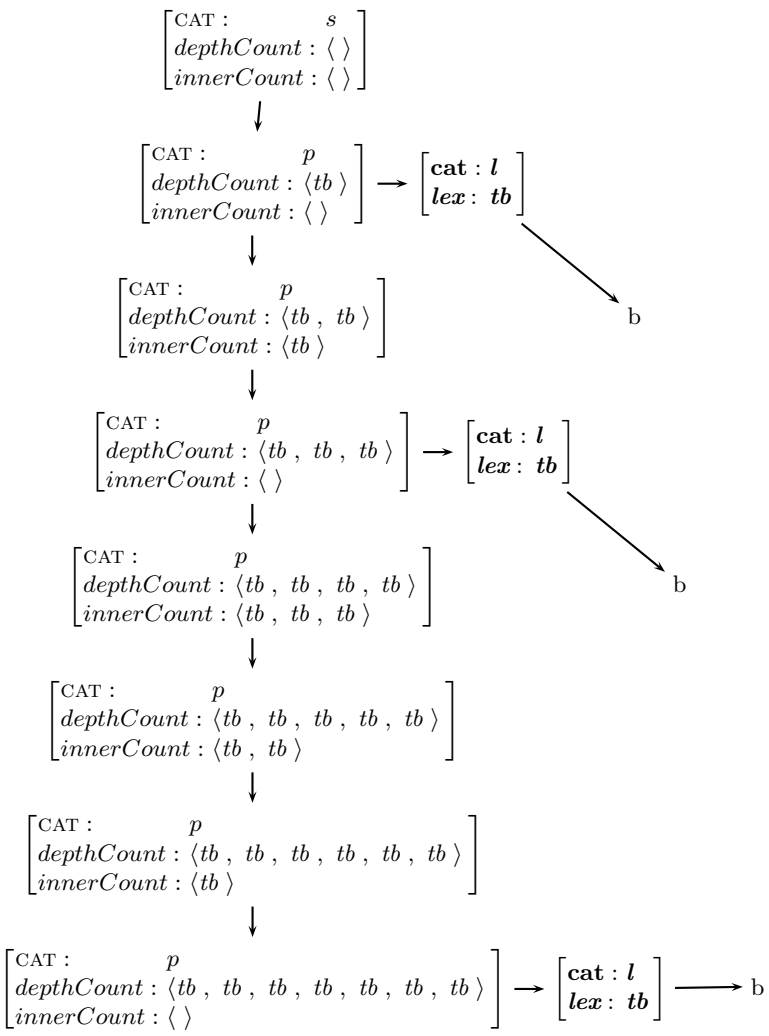


Figure 2.11: An example derivation tree for the grammar G_{DB} of figure 2.10 and the string bbb. The tree's depth is 2^3.

2. Since the innerCount list is empty, either the second or fourth rules may be applied: by applying the second rule, the depthCount list is assigned into the innerCount list, an item is added to depthCount list and a b is generated. Once the second rule has been applied, the third rule is the only applicable rule, since innerCount is non-empty. By applying the fourth rule, a b is generated and the parsing is terminated.

3. Then, the third rule is applied (assuming that the second rule has been previously applied), removing items from the innerCount list, until an empty list is reached, then go back to 2.

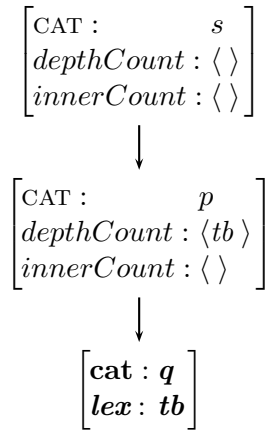


Figure 2.12: A derivation tree admitted by the grammar G_{DB} for the string b.

4. By applying the fourth rule the last b is generated.

The only possible derivation tree for a string of l occurrences of b consists of exactly $l-1$ applications of the second rule, each followed by $|innerCount|$ applications of the third rule (until innerCount list is empty) and then one application of the fourth (terminating) rule. Thus a string of l occurrences of b has just one parse tree.

Figure 2.13: A unification grammar generating the language $\{b^{2^l} \mid l > 0\}$. It is a variation of G_{DB} with an addition that in every derivation step a lexical item is generated thus the grammar generates each string of b^l symbols in b^l derivation steps.

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

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

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

Figure 2.13: A unification grammar, $G_{b^{2^n}}$.

Chapter 3

Off-line-parsability constraints

It is well known that unification based grammar formalisms are Turing-equivalent in their generative capacity (Pereira and Warren, 1983; Johnson, 1988, 87-93); determining whether a given string w is generated by a given grammar G is equivalent to deciding whether a Turing machine M halts on an empty input, which is known to be undecidable. Therefore, the recognition problem is undecidable in the general case. However for grammars that satisfy a certain restriction, called off-line parsability constraint (OLP), decidability of the recognition problem is guaranteed. In this section we present some different variants of the off-line parsability constraint 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 the term category is not well defined 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.

3.1 Off-Line-Parsability Variants

We begin the discussion with off-line parsability constraints for skeletal grammars. One of the first definitions was suggested by Pereira and Warren (1983). Their constraint was designed for DCGs (a skeletal unification grammar formalism which assumes an explicit context-free backbone) for guaranteeing termination of general proof procedures of definite clause sets. Rephrased in terms of skeletal grammars, the definition is as follows:

Definition 3.1 (Pereira and Warren's OLP constraint for skeletal grammars (OLP_{PW})). *A grammar is off-line parsable iff its context-free skeleton is not infinitely ambiguous.*

The *context-free skeleton* is obtained by ignoring all feature structures of the grammar rules and considering only the categories. In the next section we prove that the depth of every derivation tree generated by a grammar whose context-free skeleton is finitely ambiguous is bounded by the number of syntactic categories times the size of its yield, therefore the recognition problem is decidable.

Kaplan and Bresnan (1982) suggested a linguistically motivated off-line parsability constraint which refers to valid derivations for the Lexical-Functional Grammar formalism (LFG), a skeletal grammar formalism. During the research we have consulted Ron Kaplan in order to get a better view on OLP for LFG and realized that the notion of an OLP grammar is not available in LFG; OLP is in fact a property of derivation trees; for a given string w, a derivation tree is not OLP if it has two nodes, u and v, where u dominates v and u and v span exactly the same substring of w. Of course, there can be other derivations of w which are OLP.

LFG is intended for the special properties of natural languages. In LFG, the definition of $L(G)$ is not the standard one. Rather, w is in $L(G)$ if there exists an OLP derivation tree for w; the structures associated with w by G are the structures induced by the OLP derivations only.

In our analysis we are concerned with OLP grammars. The question of whether a grammar is OLP is completely irrelevant to LFG. Our motivation is not LFG only; in fact, we are trying to address what we refer to as "general unification grammars".

The next two OLP variants, are based on Kaplan and Bresnan's OLP for LFG and thus they impose a restriction on allowable c-structures, rather than on the grammar itself. There exist no explicit definition of an OLP grammar. Such a definition can be understood in (at least) two manners:

Definition 3.2 (OLP grammar).

1. *A grammar G is off-line parsable iff for every $w \in L(G)$* every *derivation tree for w satisfies the off-line parsability constraint.*

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

The first definition is very strict, it allows grammars which can generate only OLP derivation trees. The second definitions is more liberal, it allows non-OLP derivation trees as long as there exists at least one OLP derivation tree for every word of the grammar's language.

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 their constraint, suggested by Johnson (1988, 95-97), eliminating all ϵ's of any kind.

Definition 3.3 (Johnson's OLP constraint (OLP_{JO})).
A constituent structure satisfies the OFF-LINE PARSABILITY CONSTRAINT iff:

- *It does not include a non-branching dominance chain in which the same category appears twice.*
- *The empty string ϵ does not appear as a lexical form annotation of any (terminal) node.*

The first condition rules out any unbounded unary branching structures, whereas the second condition rules out any applications of ϵ-rules. In the next section we prove that the constraint bounds the depth of any off-line parsable derivation tree by a linear function of the size of its yield, thus ensuring decidability of the recognition problem.

Johnson's definition of off-line parsable constituent structures applies only to skeletal grammars, for general unification grammars the term category is not well defined since there may be infinitely many feature structures (there exists no mapping of the feature structures to a finite set of categories). In order to redefine the constraint for general unification grammars, the term category should be defined first in terms of untyped feature structures.

His definition is a restriction on allowable c-structures, rather than on the grammar itself, this is a restriction on the possible derivation trees generated by the grammar rules,

limiting the number of OLP derivation trees that have a given string as their yield to be finite. There exist no formal definition of OLP_{JO} grammars, thus we use definition 3.2 for OLP_{JO} grammars.

The next constraint is also based on Kaplan and Bresnan's constraint and is also dealing only with off-line parsable derivations, OLP grammar definitions are according to definition 3.2. The constraint uses the notion of categories and thus is applicable only to skeletal grammars.

X-bar theory grammars (Chomsky, 1975) have a strong linguistic justification in describing natural languages. Unfortunately neither Kaplan and Bresnan's nor Johnson's constraints allow such derivations, since they do not allow derivation trees in which the same category appears twice in a non-branching dominance chain. Kuhn (1999) refers to the problem from a linguist's point of view. The purpose of his constraint was to expand the class of grammars which satisfy Kaplan and Bresnan's constraint in order to allow X-bar derivations. The definition is not dealing with ϵ-rules, therefore we assume that ϵ does not represent a lexical item, as in Johnson's constraint.

Definition 3.4 (Kuhn's OLP constraint).
A c-structures derivation is valid iff no category appears twice in a non-branching dominance chain with the same f-annotation.

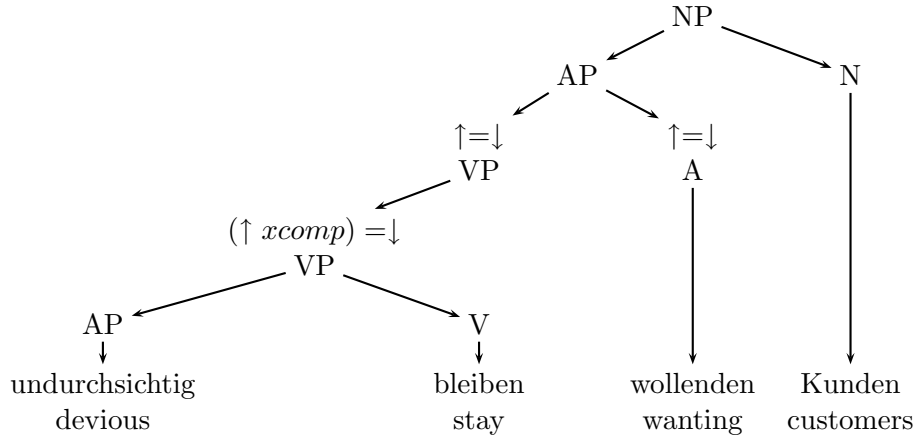
The c-structure in LFG represents the external structure of a sentence. It consists of a context-free derivation tree with an additional functional information about the constituents within a sentence. The functional information is represented by two meta-variables $\uparrow$ (up arrow) and $\downarrow$ (down arrow):

- Up arrow, $\uparrow$, represents the feature structure attached to the mother's node.
- Down arrow, $\downarrow$, represents the feature structure attached to the daughter node (the node itself).
- $(\uparrow \textsc{feat}) = \downarrow$, represents that the value of the feature FEAT at the feature structure attached to the mother's node is the feature structure attached to the daughter node; $\delta(M, \textsc{feat}) = D$, where M, D represent the feature structures attached to the mother and daughter nodes correspondingly.

- $\uparrow$=$\downarrow$, represents that the mother and daughter nodes share the same feature structure.

The c-structures are then mapped into *f-structures*. The f-structure is an acyclic feature structure which represents the internal structure of a sentence, it includes a representation of the higher syntactic and functional information of a sentence. A detailed description of LFG's c-structures, f-structures and the transformation between them is given in (Kaplan and Bresnan, 1982).

Figure 3.1 lists an example of an X-bar derivation tree (c-structure) taken from Kuhn's paper (Kuhn, 1999) and its corresponding f-structure. The derivation tree contains a non-branching dominance chain in which the category VP appears twice. Therefore it is an invalid OLP derivation (by both Kaplan and Bresnan's and Johnson's constraints). Since the functional constraint attached to the daughter node is $(\uparrow xcomp) = \downarrow$, the mother and daughter nodes do not share the same f-annotation. Therefore, the derivation satisfies Kuhn's OLP constraint.



$$\text{f-structure: } \begin{bmatrix} \text{SUBJ}: & customers \\ \text{COMP}: & \begin{bmatrix} \text{VERB}: & stay \\ \text{ADV}: & devious \end{bmatrix} \\ \text{ADJ}: & wanting \end{bmatrix}$$

Figure 3.1: An example X-bar theory c-structure and f-structure of a German sentence.

Our analysis deals with OLP grammars, Kuhn's definition is an improvement to Johnson's OLP allowing X-bar derivation trees. Since there exist no formal definition of Kuhn's

OLP grammars and his definition is very similar to Johnson's constraint, we exclude Kuhn's OLP constraint from further analysis.

The following definitions are applicable to both skeletal and general unification grammars. The first constraint was suggested by Haas (1989). Based on the fact that not every natural unification grammar has an obvious context-free backbone, Haas suggested a constraint for guaranteeing solvability of the parsing problem which is applicable to all unification grammar formalisms.

Haas' definition of a derivation tree is slightly different from the definition given in chapter 1 for unification grammars' derivation trees (definition 2.10). He allows derivation trees with nonterminals at their leaves, therefore a tree may represent a partial derivation.

Definition 3.5 (Haas' Depth-boundedness (DB)). *A unification grammar is depth-bounded iff for every $L > 0$ there is a $D > 0$ such that* ***every*** *parse tree for a sentential form of L symbols has depth less than D.*

Haas' definition of a depth-bounded grammar requires that there exists a function f such that every derivation tree's depth for a sentential form of l symbols is bounded by $f(l)$. Since he allows partial derivation trees, a sentential form of length l may contain some non-terminal symbols.

According to Haas (1989), "a depth-bounded grammar cannot build an unbounded amount of tree structure from a bounded number of symbols", therefore, for every sentential form of length l there exist a finite number of partial derivation trees, thus guaranteeing parsing termination.

Haas shows an algorithm that should verify whether a given grammar is depth-bounded. But the algorithm's termination is not guaranteed; if depth-boundedness does not hold the algorithm will enter an infinite loop. Haas states explicitly that depth-boundedness is undecidable, but provides no proof of it.

The OLP_{PW} definition applies only to skeletal grammars, general unification grammars do not necessarily yield an explicit context-free skeleton. The natural expansion of Pereira and Warren's definition in order to apply to all unification grammar formalisms is finite ambiguity for unification grammars:

Definition 3.6 (Finite ambiguity for unification grammars (FA)). *A unification grammar G is OLP iff for every sting w there exist a finite number of derivation trees.*

Shieber's OLP definition (Shieber, 1992, 79–82) is defined in terms of logical constraint based grammar formalisms. His constraint is defined in logical terms, such as models and operations on models. We reformulate the definition in terms of feature structures.

Definition 3.7 (Shieber's OLP (OLP_S)). *A grammar G is off-line parsable iff there exists a finite-ranged function F on feature structures such that $F(A) \sqsubseteq A$ for all A and there are no derivation trees admitted by G in which a node A dominates a node B, both are roots of sub-trees with an identical yield and $F(A) = F(B)$.*

The constraint is intended to bound the depth of every derivation tree by the range of F times the size of its yield. Thus the recognition problem is decidable.

Johnson's OLP constraint is too restrictive, since it excludes all repetitive unary branching chains and ϵ- rules, furthermore, it is applicable only to skeletal grammars, therefore, Torenvliet and Trautwein (1995) have suggested a more liberal constraint, which is applicable to all unification grammar formalisms.

Definition 3.8 (Honest parsability constraint (HP)). *A grammar G satisfies the Honest Parsability Constraint (HPC) iff there exists a polynomial p s.t. for each $w \in L(G)$ there **exists** a derivation with at most $p(|w|)$ steps.*

The definition guarantees that for every string of the grammar's language there exists at least one polynomial depth (in the size of the derived string) derivation tree and therefore the recognition problem can be solved in polynomial time. Furthermore, the definition allows X-bar theory derivation trees, since a category may appear twice in a non-branching dominance chain as long as the depth of the tree is bounded by a polynomial function of its yield.

3.2 Off-line-parsability analysis

In the following section we analyze the above given OLP definitions, define their properties and determine whether each of them guarantees decidability of the recognition problem.

Pereira and Warren's definition guarantees that the depth of every derivation tree admitted by an OLP_{PW} grammar is bounded by the number of syntactic categories times its yield. Thus guaranteeing decidability of both the recognition and parsing problems.

Lemma 3.1. *Let G be an OLP_{PW} grammar. Let G' be the context-free backbone of G (by definition G' is finitely ambiguous). G' cannot generate a derivation tree in which a sub-tree contains another sub-tree with the same root category and both have the same yield.*

Proof. Assume towards a contradiction that G', a finitely ambiguous context-free grammar, can generate such a derivation tree. Therefore, a sub-tree of some category A (over CATS) may be generated repeatedly many times (by applying the same grammar rules), as shown in figure 3.2, resulting in infinite ambiguity. □

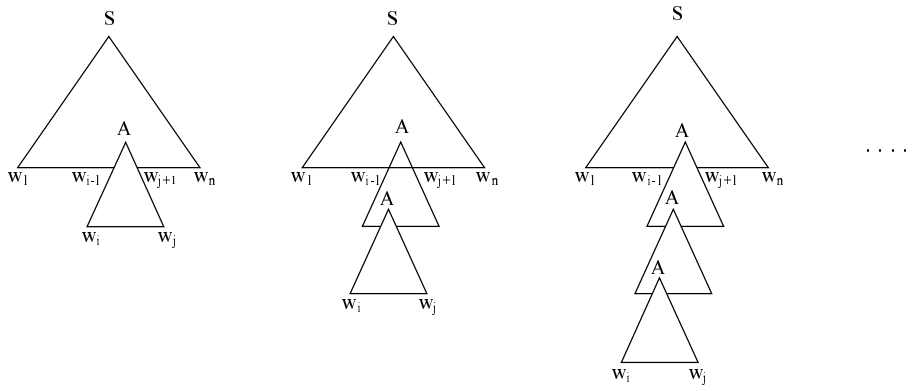


Figure 3.2: An example of derivation trees such that A is the root category of a sub-tree which derives another sub-tree of root A with the same yield.

A skeletal derivation tree is a context-free derivation tree extended by feature structures. Therefore, G itself cannot generate a derivation in which a sub-tree contains another sub-tree and both have the same root category and generate the same yield (otherwise by ignoring the feature structures, leaving just the categories, the context-free backbone is infinitely ambiguous).

It follows from lemma 3.1 that an OLP_{PW} grammar G satisfies the following properties:

- It cannot generate a derivation tree in which the same category appears twice in a non-branching dominance chain (since they both share the same yield).
- The depth of any sub-tree generating the empty string ϵ (a sub-tree whose yield is ϵ's only) admitted by G is bounded by the number of syntactic categories $|Cat|$ (otherwise, some descendant must have the same category as its root mother).

Corollary 3.2. *The depth of any OLP_{PW} derivation tree for a string of l symbols is bounded by a linear function of l.*

Proof. By lemma 3.1 and its following properties, every sub derivation tree of depth $|Cat|$ must generate at least one lexical item, otherwise a sub-tree of length greater than $|Cat|$ would generate the same yield, which is impossible since the context-free backbone is finitely ambiguous. In the worst case, every lexical symbol adds $|Cat|$ derivation steps to the derivation's depth, and after generating all symbols, there exist $|Cat| - 1$ derivation steps generating the empty string. Therefore, for generating a string of l lexical symbols, the depth of every derivation tree is at most $(|Cat| + 1) \times l$ $(O(l))$. □

Johnson's definition of off-line parsable derivation trees, guarantees that if there are no unary branching chains nor ϵ-rules, the number of rule applications in a derivation is linear in the length of the derived string and the size of the structure corresponding to the partial productions is polynomial. Thus guaranteeing decidability of the recognition problem.

Lemma 3.3. *The depth of any OLP_{JO} derivation tree for a string of l symbols is bounded by a linear function of l.*

Proof. Let d be an OLP_{JO} derivation tree for a string of length l:

- Due to the first condition of OLP_{JO}, d cannot contain a non-branching dominance chain in which the same category appears twice, therefore the size of any non-branching dominance chain is bounded by the number of different syntactic categories $|Cat|$.
- Due to the second condition of OLP_{JO}, an OLP_{JO} derivation tree's yield may not contain any ϵ's, therefore for a string of l symbols, the yield of any derivation tree is exactly of length l.

In order to reach a bound on the derivation tree's depth, the worst case, which generates the deepest tree, should be taken into considerations: suppose that, in the worst case, between any two binary branching nodes there exists a unary chain of length equal to $|Cat|$. Therefore the depth of d is at most $l \times |Cat|$. Therefore the depth of d is bounded by a linear function of l. □

Corollary 3.4. *For a given string there exist a finite number of OLP_{JO} derivation trees.*

Proof. By lemma 3.3 the depth of any derivation tree for a given string is bounded by a linear function of the length of the string. There exists a finite set of categories and a finite set of rules, therefore, there exist a finite number of all possible combinations of categories up to some linear depth. □

Corollary 3.5. *By the second definition of OLP_{JO}, if G is OLP_{JO} then for every string w there exists a derivation tree whose depth is at most $|w| \times |Cat|$ (where $|cat|$ is the number of syntactic categories).*

Proof. By lemma 3.3 the depth of any OLP_{JO} derivation tree for a given string of length l is bounded by a $l \times |Cat|$. By the second definition of OLP_{JO} grammars, for every $w \in L(G)$ there exists an OLP_{JO} derivation tree, thus for every string w there exists a derivation tree whose depth is at most $|w| \times |Cat|$. □

Haas's definition of depth-boundedness, guarantees that there exists a function bounding the depth of every derivation tree admitted by G by the size of its yield. Given such a function f, for every string w of length l it is enough to check the finite set of derivation trees up to depth $f(w)$, thus guaranteeing decidability of both the parsing and recognition problem.

Finite ambiguity may play a role as the natural expansion of the class of OLP_{PW} grammars for applying to general unification grammars, but it does not necessarily guarantee decidability of the recognition problem; the fact that a grammar induces a finite number of derivation trees on every string provides us with no upper bound of the size of each derivation tree nor any information about the actual number of derivation trees that the grammar induces on every string. Therefore, for every string w and a given grammar G,

it is possible to verify whether $w \in L(G)$ by generating a derivation tree for w, but since there exist no explicit number representing the finite number of derivation trees admitted by G for every string w, it is impossible to tell when a parsing algorithm may terminate while generating all derivation trees that G induces on w.

In order to verify whether $w \notin L(G)$, since there is no upper bound on the depth of the derivation trees, the parsing algorithm may never terminate; we cannot claim that $w \notin L(G)$ since there exist no derivation tree for w up to a certain (very high) depth d, there may be a derivation tree for w of depth $d+1$ for every natural number d. Thus the definition does not guarantee decidability of the parsing nor the recognition problems.

According to Shieber a grammar is OLP_S iff there exists a finite-ranged function mapping each two nodes sharing the same yield on every derivation tree to a different yield, thus imposing a restriction on the depth of every derivation tree admitted by an OLP_S grammar limiting the number of derivation trees that have a given string as their yield to be finite, hence guaranteeing decidability of both the recognition and parsing problems

Lemma 3.6. *The depth of any OLP_S derivation tree for a string of l symbols is bounded by a linear function of l.*

Proof. Let G be an OLP_S grammar, therefore there exists a finite-ranged function F satisfying the OLP_S conditions. In each derivation tree in order to generate each lexical item at most $|range(F)|$ derivation steps may be applied (otherwise at least two nodes are mapped to the same feature structure). Thus in order to generate a string of l symbols, the depth of every derivation tree is at most $|range(F)| \times l$. □

According to Torenvliet and Trautwein an HP grammar permits derivations, in which there exists a non-branching dominance chain where the same category appears more than once, as long as the depth of the tree is bounded polynomially by the size of the derived string. Furthermore, an HP grammar allows the **X-bar theory** derivation tree of figure 3.1 and other X-bar derivation trees presented by Kuhn (1999). Therefore, we exclude Kuhn's OLP constraint from further analysis.

HPC guarantees that there exists a polynomial function p such that for every $w \in L(G)$ there exists at least one derivation tree whose depth is bounded by $p(|w|)$. Thus the recognition problem is decidable.

The condition does not impose any bounds on the depth of every generated derivation tree, therefore it does not guarantee decidability of the parsing problem.

Chapter 4

OLP Definitions Correlation

There exist several variants of OLP in the literature, some of which do not recognize the existence of all other definitions. In the following section we make a comparison of the different definitions using the given grammar examples and lemmas. Such relationships were not investigated in the past. We first define for each of the given grammar examples their OLP properties. We then make comparative analysis of the OLP variants in terms OLP grammars. At the last section we compare the OLP variants to Johnson's OLP for the Lexical-Functional Grammar.

4.1 The grammar examples and OLP

Following we prove some lemmas regarding the given grammar examples and their OLP properties. In these specific general unification grammar examples it is possible to obtain a context free skeleton, therefore they are also used while investigating the definitions which are applicable only to skeletal grammars viewing the value of the feature CAT as the category.

The grammar G_{FA} of figure 2.4.

Lemma 4.1. *G_{FA} is not OLP_{PW}.*

Proof. Figure 4.1 lists the extracted context-free backbone of G_{FA}. The context-free backbone contains the rule $P \longrightarrow P$, therefore, it can generate a unary branching chain

of P's which immediately leads to infinite ambiguity. Therefore, G_{FA} is not an OLP_{PW} grammar. □

$$\mathcal{R} = \left\{ \begin{array}{l} S \longrightarrow P \\ P \longrightarrow P \\ P \longrightarrow Q \\ Q \longrightarrow Q\ Q \end{array} \right\}$$

Figure 4.1: The context-free backbone of the grammar G_{FA} of figure 2.4

Lemma 4.2. *G_{FA} is not OLP_{JO}.*

Proof. By lemma 2.1, in order to generate the string b^l for $l > 1$, exactly $l-1$ application of the second rule must be applied thus resulting in a non-branching dominance chain in which the category appears more than once. □

Lemma 4.3. *G_{FA} is HP and FA.*

Proof. Since the grammar rules must be applied according to their order, a string of l occurrence of b has just one parse tree and its depth is $2l$. Therefore, for every string w there exists a finite number of derivation trees and there exists a polynomial depth (of $|w|$) derivation tree for every word of the grammar's language. Therefore the grammar is both HP and FA. □

Lemma 4.4. *G_{FA} is not DB.*

Proof. Haas' definition of a derivation tree allows derivations with non-terminals at their leaves. The grammar's second rule can be applied repeatedly many times, adding items to the list, generating arbitrarily deep derivation trees whose frontier has only one symbol. Therefore G_{FA} is not a DB grammar. □

Lemma 4.5. *G_{FA} is not OLP_S.*

Proof. In order to generate the string b^n, exactly $n-1$ applications of the second rule must be applied, and then one application of the third rule, thus resulting in a non-branching dominance chain of length n. Suppose there exists an finite-ranged function, whose range is

of size n, mapping each two nodes sharing the same yield to a different range (e.g. mapping each feature structure to itself). In order to generate the string b^{n+1} the function must map at least 2 nodes to the same feature structure, therefore there exists no finite-ranged function satisfying the constraint and the grammar is not an OLP_S grammar. □

The grammar G_{inf} of figure 2.7.

Lemma 4.6. *G_{inf} is OLP_{JO} (by the second definition), but it is not OLP_{PW}.*

Proof. The grammar's language is $\{b\}$, figure 4.2 lists an example valid OLP_{JO} derivation for the string b, thus G_{inf} is OLP_{JO}. Figure 4.3 lists the extracted context-free backbone of G_{inf}. The context-free backbone contains the rule $P \longrightarrow P$, therefore, it can generate a unary branching chain of P's which immediately leads to infinite ambiguity. Therefore, G_{inf} is not an OLP_{PW} grammar. □

$$\begin{bmatrix} \text{CAT} : & s \\ \text{WORD} : & \langle s \rangle \end{bmatrix}$$
$$\downarrow$$
$$\begin{bmatrix} \text{CAT} : & p \\ \text{WORD} : & \langle tb \rangle \end{bmatrix}$$
$$\downarrow$$
$$\begin{bmatrix} \textbf{cat} : & \boldsymbol{q} \\ \textbf{word} : & \langle \boldsymbol{tb} \rangle \end{bmatrix}$$
$$\downarrow$$
$$\text{b}$$

Figure 4.2: An OLP_{JO} derivation tree for the grammar of figure 2.7 and the string b.

$$\mathcal{R} = \left\{ \begin{array}{l} S \longrightarrow P \\ P \longrightarrow P \\ P \longrightarrow Q \\ Q \longrightarrow Q \end{array} \right\}$$

Figure 4.3: The context-free backbone of the grammar of figure 2.7

Lemma 4.7. *G_{inf} is HP.*

Proof. The grammar's language is $\{b\}$, figure 4.2 lists a derivation tree for b of depth 3 generated by G_{inf}, therefore there exist a polynomial depth derivation tree for every $w \in L(G_{inf})$, thus the grammar is honest parsable. □

Lemma 4.8. *G_{inf} is not FA.*

Proof. The grammar generates infinitely many derivation trees for the string b for any natural number of applications of the second rule, therefore the grammar is infinitely ambiguous. □

Lemma 4.9. *G_{inf} is not DB.*

Proof. The grammar generates infinitely many non-branching derivation trees for the string b, thus generating arbitrarily deep derivation trees whose frontier has only one symbol. □

Lemma 4.10. *G_{inf} is not OLP_S.*

Proof. The grammar's language is $\{b\}$ there exist infinitely many derivation trees for the string b of arbitrary depths, each consisting of a non-branching dominance chain in which all nodes are sharing the same yield, therefore there exists no finite-ranged function mapping each two nodes sharing the same yield on every derivation tree to a different range. □

The grammar G_{DB} of figure 2.10.

Lemma 4.11. *G_{DB} is not OLP_{PW}.*

Proof. Figure 4.4 lists the extracted context-free backbone of G_{DB}. A context-free backbone containing the rule $P \longrightarrow P$ can generate a unary branching chain of P's which immediately leads to infinite ambiguity. Therefore G_{DB} is not an OLP_{PW} grammar.

□

Lemma 4.12. *G_{DB} is not OLP_{JO}.*

Proof. The grammar's language is $\{b^+\}$, by lemma **??**,in order to generate the string b^l for $l > 1$ at least one application of the third rule must be applied, resulting in a non-branching dominance chain of P's. Thus the grammar is not OLP_{JO}. □

$$\mathcal{R} = \left\{ \begin{array}{l} S \longrightarrow P \\ P \longrightarrow P\ Q \\ P \longrightarrow P \\ P \longrightarrow Q \end{array} \right\}$$

Figure 4.4: The context-free backbone of the grammar G_{DB} of figure 2.10

Lemma 4.13. *G_{DB} is DB and FA.*

Proof. A string of l occurrences of b has just one parse tree, and its depth is 2^l, the depth of a tree generating a string of $n+1$ occurrence of b is 2^{l+1}, between the generation of the l^{th} and $(l+1)^{th}$ b's there exist 2^n derivation steps of sentential forms of length l, therefore any derivation tree for a sentential form of length l is bounded by an exponential function of l, and the grammar is depth-bounded and finitely ambiguous. □

Lemma 4.14. *G_{DB} is not OLP_S.*

Proof. By lemma 2.4, in order to generate the string b^l exactly 2^l derivation steps must be applied (the last b is generated on the $2^l th$ derivation step), in order to generate the b^{l+1} exactly 2^{l+1} derivation steps must be applied. Since the grammar contains no ϵ-rules, between the generation of the l^{th} and $(l+1)^{th}$ symbols there exist 2^l derivation steps of nodes all sharing the same yield. Assume towards a contradiction that there exist a finite-ranged function of range 2^l mapping each two nodes on the non-branching dominance chain to a different yield, thus in order to generate the string b^{l+2} the function must map at least two such nodes to the same yield for every natural number l. Therefore, there exist no finite-ranged function satisfying the conditions and the grammar is not OLP_S. □

Lemma 4.15. *G_{DB} is not HP.*

Proof. The grammar's language is $\{b+\}$, for every string of l occurrences of b there exists exactly one derivation tree and its depth is 2^l, therefore not every $w \in L(G_{DB})$ has a polynomial depth (of l) derivation tree. □

4.2 The relationships between the OLP variants

OLP_{PW} *VS.* OLP_{JO} .

Proposition 4.16. *According to the first definition of OLP_{JO} grammars, an OLP_{JO} grammar G is not necessarily an OLP_{PW} grammar.*

Proof. The grammar $G_{J\backslash PW}$ of figure 4.5 is an OLP_{JO} grammar. It's easy to verify that the grammar can only generate two derivation trees (the two derivations shown by the figure) and both derivations are OLP_{JO} derivations. Figure 4.6 lists the extracted context-free backbone of the grammar. The context-free backbone can generate a unary branching chain of $P - Q - P$ and therefore it is infinitely ambiguous. Therefore $G_{J\backslash PW}$ is not an OLP_{PW} grammar. □

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

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

$$\mathcal{R} = \left\{ \begin{array}{lll} \begin{bmatrix} \text{STR}: & s \\ \text{WORD}: & \langle\,\rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{STR}: & ta \\ \text{WORD}: & \langle\,\rangle \end{bmatrix} & S \longrightarrow P \\ \begin{bmatrix} \text{STR}: & s \\ \text{WORD}: & \langle\,\rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{STR}: & tb \\ \text{WORD}: & \langle\,\rangle \end{bmatrix} & S \longrightarrow Q \\ \begin{bmatrix} \text{STR}: & ta \\ \text{WORD}: & \langle\,\rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{STR}: & tb \\ \text{WORD}: & \langle tb\,\rangle \end{bmatrix} & P \longrightarrow Q \\ \begin{bmatrix} \text{STR}: & tb \\ \text{WORD}: & \langle\,\rangle \end{bmatrix} \longrightarrow \begin{bmatrix} \text{STR}: & ta \\ \text{WORD}: & \langle ta\,\rangle \end{bmatrix} & Q \longrightarrow P \end{array} \right\}$$

$$\begin{array}{cc} \langle \begin{bmatrix} \text{STR}: & s \\ \text{WORD}: & \langle\,\rangle \end{bmatrix}, S \rangle & \langle \begin{bmatrix} \text{STR}: & s \\ \text{WORD}: & \langle\,\rangle \end{bmatrix}, S \rangle \\ \downarrow & \downarrow \\ \langle \begin{bmatrix} \text{STR}: & ta \\ \text{WORD}: & \langle\,\rangle \end{bmatrix}, P \rangle & \langle \begin{bmatrix} \text{STR}: & tb \\ \text{WORD}: & \langle\,\rangle \end{bmatrix}, Q \rangle \\ \downarrow & \downarrow \\ \langle \begin{bmatrix} \textbf{str}: & \boldsymbol{tb} \\ \textbf{word}: & \langle \boldsymbol{tb}\,\rangle \end{bmatrix}, \boldsymbol{Q} \rangle & \langle \begin{bmatrix} \textbf{str}: & \boldsymbol{ta} \\ \textbf{word}: & \langle \boldsymbol{ta}\,\rangle \end{bmatrix}, \boldsymbol{P} \rangle \\ \text{b} & \text{a} \end{array}$$

$$\mathcal{L}(a) = \left\{ \langle \begin{bmatrix} \text{STR}: & ta \\ \text{WORD}: & \langle ta\,\rangle \end{bmatrix}, P \rangle \right\} \quad \mathcal{L}(b) = \left\{ \langle \begin{bmatrix} \text{STR}: & tb \\ \text{WORD}: & \langle tb\,\rangle \end{bmatrix}, Q \rangle \right\}$$

Figure 4.5: An OLP_{JO} grammar $G_{J\backslash PW}$, $L(G_{J\backslash PW}) = \{a, b\}$

Proposition 4.17. *According to the second definition of OLP_{JO} grammars, an OLP_{JO} grammar G is not necessarily an OLP_{PW} grammar.*

Proof. Let G be an OLP_{JO} grammar, therefore for every $w \in L(G)$ there **exists** an OLP_{JO} derivation tree. There is no guarantee that for every w **every** derivation tree is an OLP_{JO}

$$\mathcal{R} = \left\{ \begin{array}{l} S \longrightarrow P \\ S \longrightarrow Q \\ P \longrightarrow Q \\ Q \longrightarrow P \end{array} \right\}$$

Figure 4.6: The context-free backbone of the grammar of figure 4.5

derivation tree. Therefore, a string that has an OLP_{JO} derivation tree may still have an infinite number of non-OLP_{JO} derivation trees.

By lemma 4.6, the grammar G_{inf} of figure 2.7 is an OLP_{JO} grammar (by the second definition) but since its context-free backbone is infinitely ambiguous, it is not an OLP_{PW} grammar. □

The proof can also be deduced from the proof of the above proposition (4.16), since an OLP_{JO} grammar by the first definition immediately satisfies the second definition of OLP_{JO} grammars.

Proposition 4.18. *an OLP_{PW} grammar G is not necessarily an OLP_{JO} grammar.*

Proof. The OLP_{PW} definition does not impose any explicit conditions on ϵ's. Figure 4.7 lists a counter example, an OLP_{PW} grammar G_ϵ which is not an OLP_{JO} grammar, and an example derivation tree (note that a context-free grammar can be viewed as a skeletal grammar with empty feature structures). G_ϵ is an OLP_{PW} grammar, its context-free backbone is $P \rightarrow Q\ R$. It is easy to verify that the derivation example given by the figure is the only possible derivation tree admitted by G_ϵ. Since every derivation tree's yield contains an ϵ, G_ϵ is not an OLP_{JO} grammar.

$$\begin{array}{l} P \rightarrow Q\ R \\ Q \rightarrow b \\ R \rightarrow \epsilon \\ \\ L(G_\epsilon) = \{b\} \end{array} \qquad \begin{array}{ccc} & P & \\ \swarrow & & \searrow \\ Q & & R \\ \downarrow & & \downarrow \\ b & & \epsilon \end{array}$$

Figure 4.7: An example OLP_K grammar, G_ϵ.

- According to the first definition of OLP_{JO} grammars, the grammar is not OLP_{JO} since it may generate non-OLP_{JO} derivation trees.

- According to the second definition of OLP_{JO} grammars, there exists no OLP_{JO} derivation tree for the string b, therefore not every $w \in L(G_{PW_1})$ has an OLP_{JO} derivation.

□

$OLP_{PW} \quad VS. \quad Depth - Boundedness$.

Proposition 4.19. *A DB grammar G is not necessarily an OLP_{PW} grammar.*

Proof. By lemma 4.13, the grammar G_{DB} of figure 2.10 is depth-bounded. Figure 4.4 lists the extracted context-free backbone of figure 2.10. The context-free backbone can generate a unary branching chain of P's which immediately leads to infinite ambiguity. Therefore G_{DB} is not an OLP_{PW} grammar. □

Proposition 4.20. *If G is OLP_{PW} then G is DB.*

Proof. Let G be an OLP_{PW} grammar, by corollary 3.2, the depth of any derivation tree generated by G is bounded by a linear function of the size of the derived string. Therefore, for every sentential form of length l every derivation tree's depth is bounded by a function of l. □

$OLP_{PW} \quad VS. \quad FiniteAmbiguity$.

Proposition 4.21. *An FA grammar G is not necessarily an OLP_{PW} grammar.*

Proof. By lemma 4.13, the grammar G_{DB} of figure 2.10 is finitely ambiguous. Figure 4.4 lists the extracted context-free backbone of figure 2.10. The context-free backbone can generate a unary branching chain of P's which immediately leads to infinite ambiguity. Therefore G_{DB} is not an OLP_{PW} grammar. □

Proposition 4.22. *If G is OLP_{PW} then G is FA.*

Proof. Let G be an OLP_{PW} grammar, by corollary 3.2, the depth of any derivation tree generated by G is bounded by a linear function of the size of the derived string $((|Cat| + 1) \times |yield|)$. There exist only a finite number of derivation trees up to a certain depth, therefore G is finitely ambiguous. □

OLP_{PW} *VS.* OLP_S .

Proposition 4.23. *An OLP_S grammar G is not necessarily an OLP_{PW} grammar.*

Proof. Figure 4.8 lists an example OLP_S grammar and a derivation tree. 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 there exists a mapping function satisfying the OLP_S conditions. The grammar is not OLP_{PW}, its context-free backbone can generate a unary branching chain of P's which immediately leads to infinite ambiguity. Therefore G_{S_1} is an OLP_S grammar but not OLP_{PW}.

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

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

$$\mathcal{R} = \left\{ \begin{array}{ll} \left[\textsc{f} : s\right] \longrightarrow \left[\textsc{f} : p\right] & S \longrightarrow P \\ \left[\textsc{f} : \boxed{1}\right] \longrightarrow \left[\textsc{f} : \left[\textsc{f} : \boxed{1}\right]\right] & P \longrightarrow P \end{array} \right\}$$

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

$$\langle \left[\textsc{f} : s\right], S\rangle$$
$$\downarrow$$
$$\langle \left[\textsc{f} : p\right], P\rangle$$
$$\mathbf{b}$$

Figure 4.8: An example OLP_S grammar, G_{S_1}

□

Proposition 4.24. *If G is OLP_{PW} then G is OLP_S*

Proof. By lemma 3.1, if G is OLP_{PW} then it can generate only derivation trees in which every two nodes sharing the same yield are of a different category, therefore there exists a finite ranged function F, whose range is of size $|Cat|$, mapping each feature structure on the derivation tree to its category value,

$$\forall X \in \textsc{Cats} : F\left(\begin{bmatrix} \textsc{cat} : X \\ \vdots \end{bmatrix} \right) = \left[\textsc{cat} : X\right].$$

Thus for every A, $F(A) \sqsubseteq A$ and it is guaranteed that, on every derivation tree, every two nodes sharing the same yield are mapped by F to a different range. □

OLP_{PW} *VS.* $HonestParsability$.

Proposition 4.25. *An HP grammar G is not necessarily an OLP_{PW} grammar.*

Proof. By lemma 4.7, the grammar G_{inf} of figure 2.7 is an HP grammar, the grammar's language is $\{b\}$ and there exists a polynomial depth derivation tree for the string b. The grammar's context-free backbone is listed in figure 4.3. The context-free backbone contains the rules $P \to P$ and $Q \to Q$, and therefore it is infinitely ambiguous. □

Proposition 4.26. *If G is OLP_{PW} then G is HP.*

Proof. Let G be an OLP_{PW} grammar, by corollary 3.2, the depth of any derivation tree generated by G is bounded by a linear function of the size of the derived string. Therefore, for every $w \in L(G)$ there exists a derivation tree of less than a polynomial depth. □

OLP_{JO} *VS.* $Depth - Boundedness$.

Proposition 4.27. *According to the first definition of OLP_{JO} grammars, if G is OLP_{JO} then G is DB.*

Proof. An OLP_{JO} grammar G can only generate OLP_{JO} derivation trees. By lemma 3.3, any OLP_{JO} derivation tree is of a linear depth by the size of its yield, therefore, for every sentential form of length n, every derivation tree's depth is bounded by a function of n, thus G is a DB grammar. □

Proposition 4.28. *According to the second definition of OLP_{JO} grammars, an OLP_{JO} grammar G is not necessarily a DB grammar.*

Proof. An OLP_{JO} grammar requires that for every $w \in L(G)$ there **exists** an OLP_{JO} derivation tree. In contrast, a DB grammar requires that for every $w \in L(G)$ **every** derivation is bounded by a function of $|w|$.

Having an OLP_{JO} derivation tree does not necessarily imply that every derivation tree generating a given string is of a bounded depth by a function of the size of its yield. As a counter example, consider the grammar G_{inf} of figure 2.7, the grammar's language is $\{b\}$, there exists an OLP_{JO} derivation for the string b (as shown in figure 4.2), but by lemma 4.9 the grammar G_{inf} is not a DB grammar; there exists no function of the size of the yield bounding each derivation tree's depth. □

Proposition 4.29. *A DB grammar G is not necessarily an OLP_{JO} grammar.*

Proof. A *DB* grammar G may generate non-OLP_{JO} derivation trees. Furthermore, there may exist some $w \in L(G)$ for which there exists no OLP_{JO} derivation. By lemma 4.13, the grammar G_{DB} of figure 2.10 is a depth-bounded grammar, generating the language $\{b^+\}$.

- According to the first definition of OLP_{JO} grammars, the grammar is not OLP_{JO} since it may generate non-OLP_{JO} derivation trees (such as the derivation of figure 2.11).
- According to the second definition of OLP_{JO} grammars, the only string that has an OLP_{JO} derivation is the string b, therefore not every $w \in L(G_{DB})$ has an OLP_{JO} derivation tree.

□

OLP_{JO} *VS.* *FiniteAmbiguity* .

Proposition 4.30. *According to the first definition of OLP_{JO} grammars, if G is OLP_{JO} then G is FA.*

Proof. By lemma 3.3, the depth of any OLP_{JO} derivation tree is bounded by a linear function of the size of its yield. For a given string, there exist only a finite number of derivation trees up to some bounded depth (by the size of the string). Therefore for every string w there exist a finite number of derivation trees. □

Proposition 4.31. *According to the second definition of OLP_{JO} grammars, an OLP_{JO} grammar G is not necessarily an FA grammar.*

Proof. A string that has an OLP_{JO} derivation tree may still have an infinite number of non-OLP_{JO} derivations. Figure 4.2 lists an example OLP_{JO} derivation tree for the grammar G_{inf} of figure 2.7. By lemma 4.8, the grammar is infinitely ambiguous; there exist infinitely many derivation trees for the string b. □

Proposition 4.32. *An FA grammar G is not necessarily an OLP_{JO} grammar.*

Proof. By lemma 4.3, the grammar G_{FA} of figure 2.4 is FA. Any string has a unique derivation tree.

- The grammar can generate non-OLP_{JO} derivation trees, therefore it is not an OLP_{JO} grammar by the first definition.
- Not every $w \in L(G)$ that has a finite number of derivation trees, has an OLP_{JO} derivation tree. Therefore the grammar is not an OLP_{JO} grammar by the second definition.

□

OLP_{JO} *VS.* OLP_S.

Proposition 4.33. *According to the first definition of OLP_{JO} grammars, an OLP_{JO} grammar G is OLP_S.*

Proof. Since G is OLP_{JO} it cannot generate derivation trees in which the same category appears twice in a non-branching dominance chain. Therefore, there exists a finite ranged function F, whose range is of size $|Cat|$, mapping each feature structure on the derivation tree to its category value,

$$\forall X \in \textsc{Cats} : F\left(\begin{bmatrix}\textsc{cat} : X \\ \vdots\end{bmatrix}\right) = \Big[\textsc{cat} : X\Big].$$

Thus for every A, $F(A) \sqsubseteq A$ and it is guaranteed that, on every derivation tree, every two nodes sharing the same yield are mapped by F to a different range.

□

Proposition 4.34. *According to the second definition of OLP_{JO} grammars, an OLP_{JO} grammar G is not necessarily an OLP_S grammar.*

Proof. By lemma 4.6, the grammar G_{inf} of figure 2.7 is OLP_{JO}, there exist an OLP_{JO} derivation tree for every $w \in L(G_{inf})$. By lemma 4.10, the grammar is not OLP_S since there exists no finite-ranged function satisfying the OLP_S conditions.

□

Proposition 4.35. *An OLP_S grammar G is not necessarily an OLP_{JO} grammar.*

Proof. Figure 4.9 lists an example OLP_S grammar and a derivation tree. It is easy to verify that the given derivation is the only possible derivation admitted by G_{S_2}. The derivation is of depth 2, therefore there exists a mapping function satisfying the OLP_S conditions. The grammar is not OLP_{JO} since its only derivation tree consists of a non-branching dominance chain in which the category S appears twice. □

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

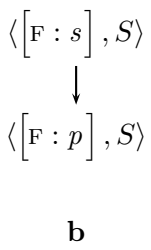


Figure 4.9: An example OLP_S grammar, G_{S_2}

OLP_{JO} *VS.* *HonestParsability* .

Proposition 4.36. *If G is OLP_{JO} then G is HP.*

Proof. The proof of the proposition will be separated for each of the OLP_{JO} grammar definitions:

- According to the first definition: an OLP_{JO} grammar G can only generate OLP_{JO} derivations. By lemma 3.3, any OLP_{JO} derivation's depth, for a string of l symbols, is bounded by a linear function of l, therefore, for every $w \in L(G)$ there exists a polynomial function of $|w|$ bounding the derivation's depth.
- According to the second definition: for every $w \in L(G)$ there exists at least one OLP_{JO} derivation. By lemma 3.3, 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.

Therefore, every OLP_{JO} grammar is also an HP grammar. □

Proposition 4.37. *An HP grammar G is not necessarily an OLP_{JO} grammar.*

Proof. Since the honest parsability constraint is not a syntactic property of grammars, the other direction is not necessarily true. Given an HP grammar G, for every $w \in L(G)$ there exists a derivation of a polynomial depth, but, not every w has an OLP_{JO} derivation. Figure 4.7 lists a counter example, an HP grammar, G_ϵ, which is not an OLP_{JO} grammar, and an example of G_ϵ's only possible derivation tree. Each of G_ϵ's derivations' yield consists of an ϵ. Therefore it does not satisfy OLP_{JO}'s constraint.

- According to the first definition of OLP_{JO} grammars, the grammar is not OLP_{JO} since it may generate non-OLP_{JO} derivation trees.
- According to the second definition of OLP_{JO} grammars, the grammar cannot generate any OLP_{JO} derivations, therefore not every $w \in L(G)$ has an OLP_{JO} derivation tree.

Furthermore, the grammar G_{FA} of figure 2.4 is an HP grammar (as proven by lemma 4.3) generating the regular language $\{b^+\}$. The grammar may generate non-OLP_{JO} derivations and not every $w \in L(G)$ can be generated by an OLP_{JO} derivation (There exists an OLP_{JO} derivation only for the string b). Therefore, G_{FA} is not an OLP_{JO} grammar. □

Depth − Boundedness VS. FiniteAmbiguity .

Proposition 4.38. *If G is DB then G is FA.*

Proof. According to Haas (1989), "a depth-bounded grammar cannot build an unbounded amount of tree structure from a bounded number of symbols", therefore, any depth-bounded grammar is also finitely ambiguous. G is DB, therefore there exists a function bounding each derivation tree's depth by the size of its yield. Since there is only a finite set of rules, only a finite number of derivation trees may be generated up to some bounded depth. Therefore every w has a finite number of derivation trees and G is an FA grammar. □

Proposition 4.39. *An FA grammar G is not necessarily a DB grammar.*

Proof. As a counter example we show an FA grammar which is not a DB grammar. Haas gives in his article (Haas, 1989) an example grammar which satisfies finite ambiguity but is not depth-bounded. The grammar G_{FA} of figure 2.4 is a general unification grammar

variation of his grammar example. By lemma 4.3, the grammar is finitely ambiguous. The grammar is not depth-bounded; Haas allows partial derivation trees with nonterminals at their leaves, therefore the second rule may be applied repeatedly many times, generating arbitrarily deep parse trees whose frontier is a nonterminal of length 1. □

Depth − Boundedness VS. OLP_S .

Proposition 4.40. *An OLP_S grammar G is not necessarily a DB grammar.*

Proof. The grammar of figure 4.10 is OLP_S, the grammar's language is $\{b\}$, there exists only one derivation for the string b (the one shown in the figure), the derivation is of depth 2, by applying the second rule no lexical items may be generated, therefore no complete derivation tree contains any application of this rule. Therefore, there exists a finite-ranged function mapping each two feature structures sharing the same yield to a different range. The grammar is not depth-bounded, by applying the second rule repeatedly many times the grammar can generate arbitrarily deep parse trees whose frontier has only one symbol. □

$$\left\{ \begin{array}{lcl} \left[\text{CAT} : S\right] & \longrightarrow & \left[\text{CAT} : P\right] \\ \left[\text{CAT} : \boxed{1}\right] & \longrightarrow & \left[\text{CAT} : \left[\text{CAT} : \boxed{1}\right]\right] \end{array} \right\}$$

$$\mathcal{L}(b) = \left\{\left[\text{CAT} : P\right]\right\}$$

$$\begin{array}{c} \left[\text{CAT} : S\right] \\ \downarrow \\ \left[\text{CAT} : P\right] \\ \downarrow \\ b \end{array}$$

Figure 4.10: An example OLP_S grammar, G_{S_3}

Proposition 4.41. *A DB grammar G is not necessarily a OLP_S grammar.*

Proof. By lemma 4.13, the grammar G_{DB} of figure 2.10 is depth-boundedness, by lemma 4.14, the grammar is not OLP_S, there exist no finite range function mapping each two nodes sharing the same yield on each derivation tree to a different range. □

Depth − Boundedness VS. HonestParsability .

There exist two main differences between the two definitions. HP requires that for every $w \in L(G)$ there **exists** a **polynomial** depth derivation, whereas DB requires that **every**

derivation for a sentential form of $|w|$ symbols is of a bounded depth by a function of $|w|$ (not necessarily polynomial).

Proposition 4.42. *A DB grammar G is not necessarily an HP grammar.*

Proof. An HP grammar requires that for every $w \in L(G)$ there **exist** at least one derivation of a polynomial depth in the size of w. Therefore, a grammar for which **every** derivation tree is of an exponential depth is not an HP grammar.

By lemma 4.13, the grammar G_{DB} of figure 2.10 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 nonterminals at their leaves.

By lemma 4.15, the grammar is not an HP grammar; for a given string every parse tree is of an exponential depth. Not every string may be generated by a polynomial depth derivation tree. □

Proposition 4.43. *An HP grammar G is not necessarily a DB grammar.*

Proof. The definition of a DB grammar requires that for every $w \in L(G)$ **every** derivation tree be of a bounded depth in the size of w. Therefore, a grammar for which there **exists** a polynomial depth derivation for every string, but may also generate arbitrarily deep partial derivation trees whose frontier consists of non-terminals, is not a depth-bounded grammar.

By lemma 4.3, the grammar G_{FA} of figure 2.4 is an HP grammar; for every string there exists a derivation tree of a polynomial depth in the size of the string.

By lemma 4.4, the grammar is not a DB grammar; the grammar can generate arbitrarily deep parse trees whose frontier has only one symbol. □

FiniteAmbiguity VS. OLP_S**.**

Proposition 4.44. *If G is* OLP_S *then G is FA*

Proof. By lemma 3.6, 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 the finite-ranged function satisfying the constraint). There exist only a finite number of derivations up to a certain depth (using a finite set of rules), therefore, for every w there exist a finite number of derivation trees and G is FA. □

Proposition 4.45. *An FA grammar G is not necessarily an OLP_S grammar.*

Proof. By lemma 4.3, the grammar G_{FA} of figure 2.4 is an FA grammar, for every $w \in L(G_{FA})$ there exists exactly one derivation tree. By lemma 4.5, G_{FA} is not an OLP_S grammar; there exists no finite-ranged function mapping each two feature structure sharing the same yield to a different range. □

$FiniteAmbiguity \quad VS. \quad HonestParsability$.

Proposition 4.46. *An HP grammar G is not necessarily an FA grammar.*

Proof. An HP grammar G requires that for every $w \in L(G)$ there **exist** at least one derivation of a polynomial depth in the size of w. But, a string may still have infinitely many derivation trees.

The grammar G_{inf} of figure 2.7 is an HP grammar; there exists a polynomial depth derivation tree for the string b (as shown in figure 4.2), therefore there exists a polynomial depth derivation tree for every word of the grammar's language. By lemma 4.8, the grammar is infinitely ambiguous; there exist infinitely many derivation trees for the string b. □

Proposition 4.47. *An FA grammar G is not necessarily an HP grammar.*

Proof. According to Haas, A depth-bounded grammar is also finitely ambiguous. Therefore the depth-bounded grammar G_{DB} of figure 2.10 is finitely ambiguous, but as proven by lemma 4.15 the grammar is not HP; each derivation tree is of an exponential depth by the size of the derived string. □

$OLP_S \quad VS. \quad HonestParsability$.

Proposition 4.48. *If G is OLP_S then G is HP.*

Proof. By lemma 3.6, the depth of any OLP_S derivation tree for a string of l symbols is bounded by a linear function of l, therefore, for every $w \in L(G)$ there exists a polynomial function of $|w|$ bounding the derivation's depth. □

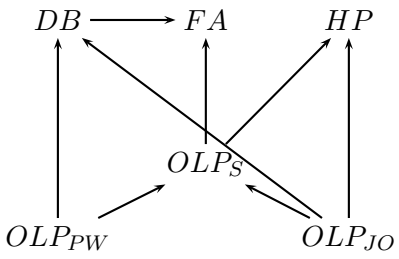
Proposition 4.49. *An HP grammar G is not necessarily an OLP_S grammar.*

Proof. By lemma 4.3, the grammar G_{FA} of figure 2.4 is an HP grammar, for every $w \in L(G_{FA})$ there exists a polynomial depth derivation by size of w. By lemma 4.5, G_{FA} is not an OLP_S grammar; there exists no finite-ranged function mapping each nodes on a derivation tree sharing the same yield to a different range. □

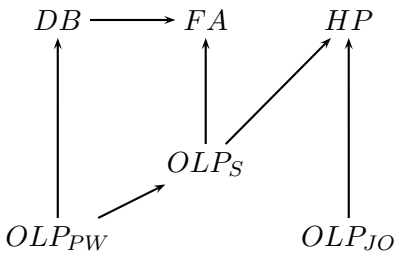
The definitions correlation hierarchy graph.

Hierarchy for **skeletal** grammars:

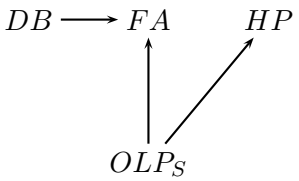
First definition of OLP_{JO} .



Second definition of OLP_{JO} .



Hierarchy for general **unification** grammars:



Chapter 5

Undecidability Proofs

Many researchers conjecture that some of the OLP variants are undecidable; it is undecidable whether a grammar satisfies the constraint, although none of them provides any proof of it. In this chapter we present one of our main contribution of this thesis. We provide proofs of undecidability to three of the undecidable OLP definitions: Finite Ambiguity (FA), Depth-Boundedness (DB) and Shieber's OLP (OLP_S).

In order to prove that depth-boundedness and OLP_S are undecidable we use a reduction from the Turing machines halting problem on the empty input to unification grammars: we show that an algorithm that decides depth-boundedness or OLP_S can also solve the halting problem. We first define (a variant of) Turing machines below: it is a machine with a single head, a two-way infinite tape and three operations: rewriting a symbol on the tape (without moving the head), a left head move and a right head move (without changing the contents of the tape). The machine accepts an input by a single finite state.

Definition 5.1 (Turing machines). *A (deterministic)* **Turing machine** $(Q, \Sigma, \flat, \delta, s, h)$ *is a tuple such that:*

- Q *is a finite set of states*
- Σ *is an alphabet, not containing the symbols* L, R *and elist*
- $\flat \in \Sigma$ *is the blank symbol*
- $s \in Q$ *is the initial state*

- $h \in Q$ *is the final state*

- $\delta : (Q \setminus \{h\}) \times \Sigma \to Q \times (\Sigma \cup \{L, R\})$ *is a total function specifying transitions.*

Johnson (1988) had proven that the recognition problem is undecidable by showing that every Turing machine can be translated into an *attribute-value grammar* and then by solving the recognition problem, the Turing machines halting problem may also be solved. Francez and Wintner (In preperation) had rephrased his proof in terms of feature structures. They show how grammars can simulate the operation of a Turing machine. They define a unification grammar, G_M, for every Turing machine, M, such that the grammar generates the word halt if and only if the machine accepts the empty input string.

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

Below is a short description of their transformation from a Turing machine to a unification grammar, a detailed description can be found in (Francez and Wintner, In preperation)

Let $M = (Q, \Sigma, \flat, \delta, s, h)$ be a Turing machine. Define a unification (skeletal) grammar G_M as follows: let FEATS be {*left, right, curr, first, rest*}, ATOMS $= \Sigma \cup \{elist\}$ and CATS $= Q \cup S$ such that $S \notin Q$. There is one terminal symbol, halt. The grammar rules can be divided to four groups. First, two rules are defined for every Turing machine:

$$\begin{array}{lcl} & & s \\ S & \longrightarrow & \begin{bmatrix} curr : & \flat \\ right : & elist \\ left : & elist \end{bmatrix} \\ h & \longrightarrow & \mathsf{halt} \end{array}$$

The first rule simulates the initial configuration of a Turing machine that operates on an empty input string: its state is s, the initial state; its tape is empty; and the head points to a blank symbol. The second rule simulates termination: when the machine is in the final, accepting state h, the grammar generates its only word, halt; equivalent to $\mathcal{L}(\mathsf{halt}) = \{h\}$.

The second group of rules are defined for rewriting transitions. For every q, σ such

that $\delta(q, \sigma) = (p, \sigma')$ and $\sigma' \in \Sigma$, the following rule is defined:

$$\begin{matrix} q \\ \begin{bmatrix} curr: & \sigma \\ right: & X \\ left: & Y \end{bmatrix} \end{matrix} \quad \longrightarrow \quad \begin{matrix} p \\ \begin{bmatrix} curr: & \sigma' \\ right: & X \\ left: & Y \end{bmatrix} \end{matrix}$$

That is, if the current state is q and the head points to σ, the next state is p and the head points to σ', while the left and the right portions of the tape are not changed.

A third group of rules is defined for right movement of the head. This case is slightly more complicated, as the situation in which the right portion of the tape is empty must be carefully taken care of. For every q, σ such that $\delta(q, \sigma) = (p, R)$ we define two rules, the first for this extreme case and the second for the default case:

$$\begin{matrix} q \\ \begin{bmatrix} curr: & \sigma \\ right: & elist \\ left: & X \end{bmatrix} \end{matrix} \quad \longrightarrow \quad \begin{matrix} p \\ \begin{bmatrix} curr: & \flat \\ right: & elist \\ left: & \begin{bmatrix} first: & \sigma \\ rest: & X \end{bmatrix} \end{bmatrix} \end{matrix}$$

$$\begin{matrix} q \\ \begin{bmatrix} curr: & \sigma \\ right: & \begin{bmatrix} first: & X \\ rest: & Y \end{bmatrix} \\ left: & W \end{bmatrix} \end{matrix} \quad \longrightarrow \quad \begin{matrix} p \\ \begin{bmatrix} curr: & X \\ right: & Y \\ left: & \begin{bmatrix} first: & \sigma \\ rest: & W \end{bmatrix} \end{bmatrix} \end{matrix}$$

The first rule is only triggered in case the *right* feature of the head, q, has the value *elist* (any other value it can have is bound to be a complex feature structure, and hence incompatible with the atom *elist*). Since the head moves to the right, the contents of the left portion of the tape are shifted left in the next state, p: its first character is the current σ, and its rest is the current states' left portion. Since the right portion of the current tape is empty, in the next configuration the head points to a blank symbol and the right portion of the tape remains empty.

The second rule is only triggered when the *right* feature of q is *not* empty: it must be a list, as the equations in the rule refer to its *first* and *rest* features. The last two

equations of this rule shift the right portion of the tape leftwards: the next configuration's *curr* feature is the first element in the current configuration's right tape; and the rest of the current configuration's right tape becomes the next configuration's *right*.

The last group of rules handle left movements in a symmetric fashion. For every q, σ such that $\delta(q, \sigma) = (p, L)$ we define two rules:

$$\begin{bmatrix} q \\ curr: \sigma \\ right: X \\ left: elist \end{bmatrix} \longrightarrow \begin{bmatrix} p \\ curr: \flat \\ right: \begin{bmatrix} first: \sigma \\ rest: X \end{bmatrix} \\ left: elist \end{bmatrix}$$

$$\begin{bmatrix} q \\ curr: \sigma \\ right: X \\ left: \begin{bmatrix} first: Y \\ rest: W \end{bmatrix} \end{bmatrix} \longrightarrow \begin{bmatrix} p \\ curr: Y \\ right: \begin{bmatrix} first: \sigma \\ rest: X \end{bmatrix} \\ left: W \end{bmatrix}$$

G_M has the following properties:
G_M can only generate unary branching derivation trees. By the above construction, G_M contains only unit-rules, therefore, it can only generate non-branching derivation trees.

Lemma 5.1. *G_M can generate at most one complete derivation tree.*

Proof. The Turing machine transitions function, δ, is a total, deterministic function; δ is defined uniquely for every state and symbol (except the final state h).
While transforming a Turing machine operation into a unification grammar, every state becomes a category and every alphabet symbol becomes an atomic value for the feature *curr* (representing the current tape content the Turing machine's head points to). Since each of G_M's rules (except the first and terminating rules) represents an entry in the transitions function, each rule's head consists of a category and feature structure containing a value for the feature *curr*. Given a category and a *curr* value, they can only unify with a single rule's head, therefore at each derivation step only one rule may be applied, until the category h is reached, resulting in a unique possible complete derivation tree. □

Lemma 5.2. *G_M generates a complete derivation tree iff M terminates on the empty input.*

Proof. By Francez and Wintner (In preperation), $\mathsf{halt} \in L(G)$ iff M terminates on the empty input. Therefore, if M terminates on the empty input, then there exists a derivation tree for halt admitted by G_M. If M does not terminate on the empty input then there exists no derivation tree for halt admitted by G_M.
Furthermore, if M does not terminate on the empty input then M never reaches the state h, therefore no derivation tree generated by G_M may ever reach category h. Since δ is a total function and G_M simulates δ, at any derivation step there is always a next applicable rule (given a state, other than h, and a symbol there is always a next configuration, therefore a category and a *curr* symbol may always unify with some rule's head), therefore G_M generates infinitely many partial derivation trees, but non of them ends with a terminal. □

5.1 Undecidability of finite ambiguity

Theorem 5.3. *Given a grammar G and a string w, it is undecidable whether w has a finite number of derivation trees admitted by G.*

Proof idea. *The proof is by contradiction. We assume that it is decidable whether G generates a finite number of derivation trees for a string w and use that assumption to show that the membership problem is decidable, contradicting Johnson's theorem (Johnson, 1988).*

Proof. Assume towards a contradiction that there exists an algorithm, A, deciding whether w has a finite number of derivation trees admitted by G. Construct an algorithm, B, to decide the membership problem, with B 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 $\mathcal{R}' = \mathcal{R} \cup A^s \rightarrow A^s$.
2. Run algorithm A on G', w.
3. If G' can generate only a finite number of derivation trees for w then $w \notin L(G)$; Otherwise, $w \in L(G)$.

By the construction of G', since $\mathcal{R} \subset \mathcal{R}'$ and they share the same lexicon and start symbol, any derivation tree generated by G can also be generated by G'.

We show below that G' generates an infinite number of derivation trees for w if and only if $w \in L(G)$.

If G' can only generate a finite number of derivation trees for a string w, then there exist no derivation tree for w admitted by G. Suppose that there exists at least one derivation tree for w admitted by G and thus there exist a derivation tree for w admitted by G', therefore, by applying the rule $A^s \rightarrow A^s$, G' can generate an infinite number of tree structures for w, contradicting the algorithm's outcome. Since G cannot generate any derivation tree for w, $w \notin L(G)$.

If G' can generate infinitely many derivation trees for a string w, then there exist at least one derivation tree for w admitted by G' which does not contain any applications of the rule $A^s \rightarrow A^s$ and therefore there exists a derivation tree for w in G. Suppose that there exist no derivation tree for w in G' without any applications of the rule $A^s \rightarrow A^s$ (which immediately implies that there exist no derivation tree for w in G), since the rule $A^s \rightarrow A^s$ only loops over the start symbol, applying it would not result in generating a derivation tree for w. Therefore G' can generate no derivation tree for w, contradicting the algorithm's outcome. Since there exists a derivation tree for w in G, $w \in L(G)$. □

Corollary 5.4. *Finite Ambiguity is undecidable.*

Proof. By theorem 5.3, 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. □

5.2 Undecidability of depth-boundedness

A grammar, G, is depth-bounded if there exists a function, f, such that for every sentential form of length n, the depth of every derivation tree is bounded by $f(n)$. A grammar G is not depth-bounded if for every function f there exists a derivation tree for a sentential form of length n whose depth is greater then $f(n)$.

Haas allows partial derivation trees with non-terminals at their leaves as legal derivation trees. A depth-bounded grammar cannot build an unbounded number of tree struc-

ture from a bounded number of symbols (terminals/non-terminals). A grammar generating an infinite number of tree structures for a sentential form of length n is not depth-bounded.

Theorem 5.5. *depth-boundedness is undecidable.*

Proof. Assume towards a contradiction that there exists an algorithm, A, for deciding depth-boundedness, that is, deciding whether there exists a function f bounding each derivation tree's depth for a sentential form of length n by $f(n)$. Construct an algorithm ,B, to decide the universal 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 depth-bounded then M terminates on the empty input;
   Otherwise, M does not terminate on the empty input.

If the algorithm claims that G_M is depth-bounded then G_M can only generate a finite number of partial (non-branching) derivation trees. One of these derivation trees must end with a terminal, otherwise, since δ is a total function and each of the grammar rules simulate δ, each partial derivation tree's leaf (of the finite set of possible partial derivation trees) may unify with some rule's head infinitely many times resulting in an infinite number of partial derivation trees. Therefore, G_M generates a complete derivation tree and by lemma 5.2, M terminates on the empty input.

If the algorithm claims that G_M is not depth-bounded, then G_M generates infinitely many non-branching partial derivation trees, while non of them ends with a terminal. Assume towards a contradiction that G_M can generate a complete derivation tree, therefore, since δ is deterministic and no rule's head may unify with category h, G_M may only generate a finite set of partial derivation trees (where each is a sub-tree of the complete derivation tree), in contradiction to the claim that G_M is not depth-bounded. Therefore, G_M may not generate any complete derivation trees and by lemma 5.2, M does not terminate on the empty input. □

5.3 Undecidability of OLP_S

Theorem 5.6. *OLP_S is undecidable.*

Proof. Assume towards a contradiction that there exists an algorithm, A, 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 $\langle A \rangle$ dominates a node $\langle B \rangle$, both are roots of sub-trees with an identical yield and $F(A) = F(B)$. Construct an algorithm ,B, to decide the universal 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. Construct G'_M, by adding the rule $A^s \rightarrow A^s$ to G_M's set of rules, $\mathcal{R}'_M = \mathcal{R}_M \cup A^s \rightarrow A^s$.

3. Run algorithm A on G'_M.

4. If G'_M is OLP_S then M terminates on the empty input;
   Otherwise, M does not terminate on the empty input.

If the algorithm claims that G'_M is OLP_S then there exists a finite-ranged function mapping each two descendant nodes to a different range, therefore there exist no derivation tree for ϵ admitted by G'_M. Suppose that there exists a derivation tree for ϵ admitted by G'_M therefore, by applying the rule $A^s \rightarrow A^s$, G'_M can generate infinitely many derivation trees for the string ϵ, thus there exist no finite-ranged function mapping each two nodes on a derivation with the same yield to a different range, contradicting the algorithm's outcome. Therefore, there exist no derivation tree for ϵ admitted by G_M, $\epsilon \notin L(G_M)$.

If the algorithm claims that G'_M is not OLP_S, then G'_M can generate a derivation tree for ϵ in which every finite-ranged function maps at least two nodes with the same yield to the same range. Therefore there exist a derivation tree for ϵ admitted by G'_M. By the construction of G'_M, there exist a derivation tree for ϵ admitted by G_M, $\epsilon \in L(G_M)$. □

Chapter 6

A Novel OLP Constraint - OLP_{JA}

In this chapter we present our main contribution, a decidable OLP constraint. Our constraint is more liberal than the existing decidable constraints. Furthermore, unlike the existing decidable constraints, the constraint is applicable to all unification grammar formalisms, yet there exists an algorithm for deciding whether a grammar satisfies it.

6.1 A decidable definition of OLP (version 1)

In the following section we present the first version of our OLP constraint and prove that it guarantees decidability of the recognition problem. In this version we assume that ϵ does not appear as a lexical form annotation of any (terminal) node; the grammars contain no ϵ-rules, in the improvements section we provide a more liberal constraint allowing grammars which contain ϵ-rules.

Definition 6.1. *A sequence of unit-rules $R_1, \ldots, R_k$ ($k \geq 1$) is* cyclicly unifiable *iff there exists a sequence of MRSs $\sigma_1, \ldots, \sigma_{k+2}$ of length 1 (feature structures) 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 6.1 lists two grammar rules, ρ_1, ρ_2. The sequence $\langle \rho_1, \rho_2 \rangle$ is cyclicly unifiable, e.g. by $\langle \sigma_1 = \begin{bmatrix} \text{CAT} : P \\ \text{F} : \quad a \end{bmatrix}, \sigma_2 = \begin{bmatrix} \text{CAT} : Q \\ \text{F} : \quad a \end{bmatrix}, \sigma_3 = \begin{bmatrix} \text{F} : b \end{bmatrix}, \sigma_4 = \begin{bmatrix} \text{CAT} : Q \\ \text{F} : \quad b \end{bmatrix} \rangle$.
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 $\left[\text{F} : b\right]$; then, applying ρ_1 necessarily yields $\begin{bmatrix}\text{CAT} : Q \\ \text{F} : \quad b\end{bmatrix}$, which is incompatible with the head of ρ_2. Hence ρ_2 cannot be applied again.

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

Figure 6.1: An example grammar rules

Definition 6.2 (Jaeger's OLP constraint (OLP_{JA_1})). *A grammar G is OLP iff it contains no cyclicly unifiable sequences.*

Lemma 6.1. *If a grammar G contains no cyclicly unifiable sequences, G does not permit 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. □

Lemma 6.2. *The depth of every derivation tree whose yield is of length n admitted by an OLP_{JA_1} grammar G is bounded by $u \times n$, where u is the number of G's unit-rules.*

Proof. Since the grammar contains no cyclicly unifiable sequences, by lemma 6.1 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 terminating node 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 times the size of its yield. □

Corollary 6.3. *Parsing termination is guaranteed for OLP_{JA_1} grammars.*

Proof. Since the depth of every derivation tree admitted by an OLP_{JA_1} grammar is bounded by a linear function of the size of its yield, it is possible to enumerate the derivation trees that have a given string as their yield, therefore parsing termination and decidability of the recognition problem are guaranteed. □

Theorem 6.4. *It is decidable whether a grammar is* OLP_{JA_1}.

Proof. In the next section we present an algorithm for deciding OLP_{JA_1}. □

6.1.1 An algorithm for deciding OLP_{JA_1}

In order to detect cyclicly unifiable sequences, only unit-rules should be considered; we use a graph annotation and search for cycles in the graph.

We first create a *unit-rules transitions graph, UTG*, a directed transitions 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.

Then, we look for cycles in the UTG, which may indicate a cyclicly unifiable sequence. For each cycle, we simulate its operation by consecutively applying all its vertices in order to verify whether they form a cyclicly unifiable sequence. Simulation is done by applying the rules according to the order of the sequence, whereas the initial symbol is the empty feature structure.

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, as it is not guaranteed that after applying several rules, unifiability between the resulting feature structure and the head of the next rule still holds. Simulation of the cycle should be applied beginning each time with a different cycle vertex, as it is possible that by beginning the simulation with some vertex, the cycle's vertices form a cyclicly unifiable sequence, but for others they do not as noted in figure 6.1 above.

The algorithm is listed in figure 6.2.

6.1.2 Correctness of the algorithm

In order to look for cyclicly unifiable sequences in a grammar G, only unit-rules should be taken into consideration. We first make a transformation from a grammar to a graph

annotation, thus we can use some graph algorithms. We then search for cycles in the graph, and check whether any of these cycles' vertices form a cyclicly unifiable sequence.

Lemma 6.5. *If a sequence of unit-rules does not appear as a cycle in the UTG, 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 UTG. Therefore, if a sequence of rules does not form a cycle in the UTG, it is not a cyclicly unifiable sequence. □

Since any cyclicly unifiable sequence is represented as a cycle in the UTG, only cycles of vertices should be considered. Once a cycle is detected, it represents unifiability between every two consecutive vertices; unifiability between the head and body of two consecutive rules. It still does not necessarily imply that the cycle's vertices form a cyclicly unifiable sequence. We simulate the cycle's operation using the function *is_cyclicly_unifiable*, whose input is a sequence of rules (some rotation of the cycle's vertices), beginning each time with a different cycle vertex. The function applies each of the rules consecutively using unification in context (as defined in Francez and Wintner (In preperation, 121-122)), and if one of the rules may not be applied (the resulting feature structure is not unifiable with the rule's head) then it returns false; if all rules have been applied successfully, the function returns true.

Lemma 6.6. *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 6.7. *The algorithm returns true iff G is OLP_{JA_1}.*

Proof. In order to check whether G contains cyclicly unifiable sequences, only unit-rules should be considered. By lemma 6.5, while checking if a sequence of unit-rules is cyclicly unifiable, only cycles of rules should be taken into consideration, and since a cyclicly unifiable sequence is always represented by a cycle in the UTG, all cyclicly unifiable sequences are always detected.

On each cycle, *is_cyclicly_unifiable* is applied from each of the cycle's vertices, and by lemma 6.6, 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_{JA_1}.

If the algorithm returns false then *is_cyclicly_unifiable* returned true on a set of vertices, by lemma 6.6, this set represents a cyclicly unifiable sequence, thus the grammar contains at least one cyclicly unifiable sequence and is not OLP_{JA_1}. □

6.2 Evaluation

OLP_{JA_1} is applicable to both skeletal and general unification grammars. By omitting ϵ-rules, it is more liberal than the existing decidable definitions that are limited to skeletal formalisms only, and unlike all definitions that are applicable to general unification grammars, it can be tested efficiently.

The grammars G_{ww} of figure 2.1 and G_{abc} of figure 2.3 are OLP_{JA_1}; they contain no unit-rules, therefore no non-branching dominance chains can be generated. The grammar

$G_{b^{2^n}}$ of figure 2.13 is OLP_{JA_1}, as it contains only one unit-rule, which clearly is not cyclicly unifiable.

The class of OLP_{JA_1} grammars contains the class of OLP_{JO_1} grammars; since an OLP_{JO_1} 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. Thus, any OLP_{JO_1} grammar G is also OLP_{JA_1}.

An OLP_{JO_2} grammar G is not necessarily OLP_{JA_1}. In an OLP_{JO_2} grammar for every $w \in L(G)$ there exists an OLP derivation tree, but it does not imply that the grammar cannot generate derivation trees in which the same rule may applied more than once sharing the same yield. The grammar G_{fin} of figure 2.7 is OLP_{JO_2}, since there exist an OLP derivation tree for the string b. G_{fin} is not OLP_{JA_1}, its second rule may be applied repeatedly infinitely many times, thus the grammar contains cyclicly unifiable sequences.

Since an OLP_{PW} grammar may contain ϵ-rules, an OLP_{PW} grammar G is not necessarily OLP_{JA_1}.

OLP_{JA_1} admits grammars whose c-structure may contain a non-branching dominance chain in which the same category may appear twice as long as it is generated by a sequence of unit-rules that is not cyclicly unifiable. Furthermore, It does not assume an explicit context-free skeleton.

The following grammar is OLP_{JA_1} but neither OLP_{JO} nor OLP_{PW}:

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

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

The following discussion shows that neither HP nor DB nor FA imply OLP_{JA_1}.

The grammars G_{HP-b^+} of figure 2.4, which is HP and FA, and G_{HP-b} of figure 2.7, which is HP, are not OLP_{JA_1}; e.g. by their second rule and the following set of feature structures:

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

Their second rule may be applied repeatedly many times, resulting in a non-branching dominance chain in which the same rule may be applied more than once.

The grammar G_{DB-b^+} of figure 2.10 is DB and FA, but it is not OLP_{JA_1}; e.g. by the third rule and the following set of feature structures:

$$\left\{ \begin{bmatrix} \text{CAT} : & p \\ depthCount : & \langle t, t, t \rangle \\ innerCount : & \langle t, t, t \rangle \end{bmatrix}, \begin{bmatrix} \text{CAT} : & p \\ depthCount : & \langle t, t, t, t \rangle \\ innerCount : & \langle t, t \rangle \end{bmatrix}, \begin{bmatrix} \text{CAT} : & p \\ depthCount : & \langle t, t, t, t, t \rangle \\ innerCount : & \langle t \rangle \end{bmatrix} \right\}$$

It can generate a non-branching dominance chain in which the third rule may be applied more than once.

An OLP_S grammar G is not necessarily OLP_{JA_1}. Figure 6.3 lists 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, furthermore, there exist only four possible feature structures for each node in every derivation tree admitted by G_S, 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 sharing the same yield are mapped to the same feature structure. The grammar is not OLP_{JA_1}, since the first rule may be applied twice consecutively, resulting in a cyclicly unifiable sequence. In section 6.3 we present an improvement to OLP_{JA_1} which allows G_S.

Every OLP_{JA_1} grammar G is also HP; by lemma 6.2, the depth of every derivation tree for a string of n symbols is bounded by a linear function of n, therefore for every $w \in L(G)$ there exists a derivation tree whose depth is polynomial in the size of w.

By lemma 6.2, every derivation tree for a sentential form of length n is bounded by a linear function of n, therefore, an OLP_{JA_1} grammar G is also DB. Thus, by the transitivity of implicature, An OLP_{JA_1} grammar G is also FA.

OLP_{JA_1} is a restriction on derivation trees such that no two nodes sharing the same yield on a derivation tree are unifiable with the same rule's head. OLP_S is a restriction on derivation trees such that no two nodes on a derivation tree sharing the same yield are mapped to the same range. We tend to believe that an OLP_{JA_1} is not necessarily OLP_S, meaning that there exist an OLP_{JA_1} grammar for which there exists no finite-ranged mapping function satisfying the OLP_S conditions, but we have not been able to come up with one. Our intuition is as follows: by definition in an OLP_S grammar there exist a finite-ranged mapping function f mapping each two nodes on a derivation tree sharing the same yield to a different range. f is finite and maps each feature structure to a certain value, satisfying the constraint, without considering its occurrences on the derivation tree.

If a grammar G is OLP_{JA_1} then the depth of every sub-derivation sharing the same yield is bounded by the number of G's unit-rules. We think that there might be some cases in which the same feature structure must be mapped to a different range in different chains in order to satisfy the constraint. Thus the G is OLP_{JA_1} but it might not be OLP_S.

Figure 6.4 depicts the revised inter-relations hierarchy diagram of the OLP definitions including OLP_{JA_1}.

6.2.1 Limitations of OLP_{JA_1}

The class of OLP_{JA_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_{JA_1}, then using the algorithm for deciding OLP_{JA_1}, we could also decide whether a grammar satisfies the other constraint which is undecidable.

OLP_{JA_1} guarantees parsing termination, but there exist non-OLP_{JA_1} grammars for which parsing termination holds. The grammar of figure 6.3 is OLP_S, thus parsing termination is guaranteed but it is not OLP_{JA_1}.

Assume that a grammar G contains a cyclicly unifiable sequence, $R_1, \ldots, R_k$. Whether any of $R_1, \ldots, R_k$ may ever be applied in any derivation tree admitted by G is an undecidable problem. Therefore, G is not OLP_{JA_1} although parsing termination may still be guaranteed for it.

OLP_{JA_1} does not allow any unit-rules sequences in which the same rule may be applied more than once. There might be some unit-rules sequences in which at some point, after applying the sequence rules repeatedly several times, unification between the resulting feature structure and the head of the next rule may no longer hold, hence parsing will terminate. In the next section we propose an improvement to OLP_{JA_1} called OLP_{JA-l}, which allows such grammars.

6.3 Improvements

6.3.1 A decidable definition of OLP (version 2)

In the following section we present the second version of our OLP constraint. The previous version of our constraint is more liberal than Johnson's constraint, but since it does not

permit grammars which contain ϵ-rules, it is incomparable with Pereira and Warren's constraint. Furthermore, ϵ's play a major role in many natural languages descriptions. In this version we allow ϵ-rules and prove that our constraint is more liberal then the existing decidable constraints.

We first create a set of *ϵ-derivables* E, consisting of the heads of all the rules that are not considered to never derive an ϵ. We later use the set for defining our constraint.

In order to create the set E we provide a notion of a local-unifiability tree for each of the grammar rules; the tree is constructed beginning each time with a different grammar rule. The tree is used to verify whether there exists a path in which some rule may be applied more than once sharing the same yield, ϵ.

Definition 6.3 (A local-unifiability tree). *Let $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$ be a unification grammar. Let $\rho \in \mathcal{R}$ be some grammar rule. A tree is a local unifiability tree admitted by G iff:*

- *The root is ρ's head;*
- *The vertices are feature structures;*
- *If a vertex A has k descendants, $B_1, B_2, \ldots, B_k$, then there exists a rule $B \rightarrow B_1, B_2, \ldots, B_k \in \mathcal{R}$ and A is unifiable with B.*

The edges represent unifiability between a vertex and some rule's head, and the vertex' daughters are the unifiable rule's body. Thus the daughters are the most general feature structures after applying the rule. Therefore the tree can be calculated off-line.

Definition 6.4 (ϵ-derivables set, E). *Given a grammar $G = \langle \mathcal{R}, \mathcal{L}, A^s \rangle$, let E_d be the set of rule's heads that are not considered to never derive an ϵ by a local-unifiability tree of depth at most d. Let d_{max} be the the size of $\mathcal{R}$.*

- $E_1 = \{A \mid A \rightarrow \epsilon \in \mathcal{R}\}$
- $E_d = \{A \mid A \rightarrow A_1, \ldots A_n \in \mathcal{R}$ *and there exists a sequence* $B_1 \ldots B_n$ *such that for each* $1 \leq i \leq n$, A_i *is unifiable with* $B_i \in E_{l_i}$ *where* $l_i < d\}$

Calculate E_d for $d \leq d_{max}$. Let E be the union of all E_d's for $1 \leq d \leq d_{max}$.

Lemma 6.8. *E is finite and can be calculated in a polynomial time.*

Proof. E contains at most all rules' heads, therefore the size of E is at most $|\mathcal{R}|$.

Each E_i is calculated incrementally beginning with E_1. In order to calculate each E_i, 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 calculated in at most $(l-1) \times |\mathcal{R}|^2$ steps and therefore E is calculated in at most $(l-1) \times |\mathcal{R}|^3$ steps ($(l-1) \times |\mathcal{R}|^2 \times d_{max}$). □

Lemma 6.9. *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, such that all internal vertices up to level l are unifiable with elements of E.

For $l = 1$, since all internal vertices of depth n are ϵ's, all of their mothers (at depth $n-1$) are feature structures unifiable with ϵ-rule's head. Thus all internal vertices on level 1 are unifiable with elements of E

Assuming that the induction hypothesis holds for $l < n$, so that all internal vertices on each up to level l are unifiable with elements of E.

The induction step $l = n$, level n consists of A only therefore all internal vertices $N_1, \ldots, N_m$ on level $l - 1$ are derived by the rule $A \rightarrow N_1, \ldots, N_m$. By the induction hypothesis, each of these vertices is unifiable with elements of E, and thus, by the construction of E, $A \in E$, contradiction. □

We next construct the set UR which we later use in order to decide whether G is OLP. UR is constructed as follows:

- UR contains all of G's unit-rules.
- For any rule $A \rightarrow A_1 \ldots A_n \in \mathcal{R}$ where $n > 1$, if $A \in E$, for $1 \leq i \leq n$ add the rule $A \rightarrow A_i$ to UR.

- For any rule $A \rightarrow A_1 \dots A_n \in \mathcal{R}$ if all of the bodies' elements but one (A_i) are unifiable with elements of E, add the rule $A \rightarrow A_i$ to *UR*.

In this OLP version, as in the previous one, we would to prevent grammars which generate derivation trees in which the same rule may be applied more than once sharing the same yield. Since the grammar may contain ϵ-rules, it is not enough to search for unit-rules chain. The purpose of the second bullet is to prevent applications of the same rule more than once in a sub-derivation tree whose yield is ϵ (for example, the context-free grammar of figure 6.5a). Thus preventing grammars generating infinitely deep sub-derivation trees whose yield consists of ϵ's only. The purpose of the third bullet is to consider rules that all of their bodies' elements but one are unifiable with elements in E as unit-rules, thus preventing an application of the same rule more than once in a sub-derivation tree sharing the same yield (For example the context-free grammar of figure 6.5b).

We later verify whether there exist *UR* chains which may indicate the possibility of applying its dual rules more than once sharing the same yield.

Definition 6.5. *Let $R = R_1, \dots, R_k$ ($k \geq 1$) be a sequence of UR rules. R is* cyclicly unifiable *iff there exists a sequence of MRSs $\sigma_1, \dots, \sigma_{k+2}$ of length 1 (feature structures) 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 6.1 lists two grammar rules, ρ_1, ρ_2. The sequence $\langle \rho_1, \rho_2 \rangle$ is cyclicly unifiable; ρ_1 may be applied twice, the sequence $\langle \rho_2, \rho_1 \rangle$ is not cyclicly unifiable; ρ_2 cannot be applied more than once.

Definition 6.6 (Jaeger's OLP constraint (OLP_{JA_2})). *A grammar G is OLP iff it contains no cyclicly unifiable sequences.*

Lemma 6.10. *Let G be an OLP_{JA_2} grammar. G does not permit any derivation trees in which the same rule is applied more than once sharing the same yield.*

Proof. Assume towards a contradiction that an OLP_{JA_2} grammar G can generate a derivation tree which contains a sub-derivation in which the same rule is applied more than once sharing the same yield. Therefore there exists a derivation tree such that some rule ρ is applied more than once sharing the same yield as shown in figure 6.6. Let A be the dominating feature structure from which the first application of ρ is applied. Let B be its descendant from which ρ may be applied again sharing the same yield.

Such a sub-derivation can result only by consecutive applications of unit-rules, ϵ-deriving rules and rules whose all of their bodies' elements but one may derive an ϵ (all other rules are branching rules in which at least two of the bodies' element are non ϵ-derivable, thus extending the 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 on the $j-th$ place 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 bodies' 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 share the same yield).

Thus each of $V'_1, \ldots, V'_k$ is a *UR* rule 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 thus V'_1 may be applied again resulting in FS. Therefore, the sequence $V'_1, \ldots, V'_k$ is cyclicly unifiable by $A, FS_1, \ldots, FS_{k-1}, B, FS$. Hence G contains a cyclicly unifiable sequence and it is not OLP_{JA_2}, contradiction. □

Lemma 6.11. *The depth of any OLP_{JA_2} derivation tree for a string of n symbols is bounded by $|\mathcal{R}| \times n$.*

Proof. Let G be an OLP_{JA_2} grammar. By lemma 6.10, the maximum depth of a sub-derivation sharing the same yield is bounded by $|R|$, thus in every derivation tree after at most R derivation steps sharing 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 6.12. *Parsing termination is guaranteed for OLP_{JA_2} grammars.*

Proof. Since the depth of every derivation tree admitted by an OLP_{JA_2} grammar is bounded by $|\mathcal{R}| \times n$, it is possible to enumerate the derivation trees that have a given

string as their yield, therefore parsing termination and decidability of the recognition problem are guaranteed. □

Theorem 6.13. *It is decidable whether a grammar is* OLP_{JA_2}.

Proof. The algorithm is listed in figure 6.7. □

Since the set of *UR* rules consists of unit-rules only, the algorithm for deciding OLP_{JA_1} of section 6.1.1 can be also used for deciding OLP_{JA_2} given a *UR* set.

Evaluation

OLP_{JA_2} is more liberal than OLP_{JA_1}; since the set of unit-rules is a subset of UR, any grammar satisfying OLP_{JA_1} would also satisfy OLP_{JA_2}.

Unlike version 1, any OLP_{PW} grammar G is also OLP_{JA_2}, otherwise, G contains cyclicly unifiable sequences and thus the same rule may be applied more than once sharing the same yield resulting in an infinitely ambiguous context-free backbone (by lemma 3.1).

OLP_{JA_2} is still not as liberal as the undecidable constraints, but it can be tested efficiently.

Figure 6.8 depicts the revised inter-relations hierarchy diagram of the OLP definitions including OLP_{JA_2}.

6.3.2 A decidable definition of OLP, OLP_{JA-l}

We extend the class of OLP_{JA_2} grammars by allowing *UR* rules sequences whose equivalent rules may be applied at most a constant number of times. The improved constraint is called OLP_{JA-l}, where l is an arbitrary number:

Definition 6.7. *A sequence of UR rules* $R_1, \ldots, R_k$ *is* l-cyclicly-unifiable *iff* $(R_1, \ldots, R_k)^l$ *is cyclicly unifiable.*

Definition 6.8. *A grammar* G *is* OLP_{JA-l} *iff it contains no l-cyclicly-unifiable sequences.*

The grammar of figure 6.3 is not OLP_{JA_2}, but it is OLP_{JA-2}, its first rule may be applied repeatedly at most twice, therefore the sequence $\langle \rho_1, \rho_1 \rangle$ is not cyclicly unifiable.

Parsing termination is guaranteed for OLP_{JA-l} grammars; since the grammar contains no l-cyclicly-unifiable sequences, the depth of any sub-derivation sharing 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_{JA-l} grammar G is bounded by $(l \times |\mathcal{R}|) \times n$.

OLP_{JA-l} is decidable; the algorithm for deciding OLP_{JA_2} of figure 6.7 can be extended in order to decide OLP_{JA-l}, the only difference being that on each cycle *is_cyclicly_unifiable* is called with some cyclic rotation of $(V_1, \ldots, V_k)^l$. *is_cyclicly_unifiable* is unchanged. The algorithm for deciding OLP_{JA-l} is listed in figure 6.9.

As shown above, the grammar of figure 6.3 is not OLP_{JA_2}, but it is OLP_S and OLP_{JA-2}. OLP_{JA-l} is an improvement to OLP_{JA_2}, but it is still not equivalent to the class of OLP_S grammars; a grammar G is OLP_S if there exists a finite ranged function satisfying the constraint, meaning there exists some l such that $|range(F)| = l$. Given a natural number l it is decidable whether G is OLP_{JA-l}. But the question whether there exist some value of l such that G is OLP_{JA-l} is undecidable.

6.3.3 A decidable definition of OLP, OLP_{JA-rot}

A cyclicly unifiable sequence of *UR* rules may have some cyclic rotation which is not cyclicly unifiable. A grammar containing such sequences is not OLP_{JA_2} (e.g. the grammar of figure 6.1: the sequence $\langle \rho_1, \rho_2 \rangle$ is cyclicly unifiable, but the sequence $\langle \rho_2, \rho_1 \rangle$ is not cyclicly unifiable). As we prove below, if a sequence of *UR* rules has some cyclic rotation which is not cyclicly-unifiable, then none of its rotations is 2-cyclicly-unifiable, thus in every rotation none of the rules may be applied more than twice (when applied consecutively together). We would like to extend the class of OLP_{JA_2} grammars by allowing such sequences.

Proposition 6.14. *Let $R = R_1, \ldots, R_k$ be a cyclicly unifiable sequence. If there exists some cyclic rotation $V = V_1, \ldots, V_k$ of R such that V is not a cyclicly unifiable sequence, then it is possible to apply each rule at the sequence consecutively at most twice. Therefore, neither R nor any of its rotations is 2-cyclicly-unifiable.*

Proof. Let R be a cyclicly unifiable sequence. Let V be a cyclic rotation of R i positions to the right which is not cyclicly unifiable. Consider a sequence of MRSs, $\sigma_1, \sigma_2, \ldots, \sigma_{k+2}$, satisfying R's cyclicly-unifiability. V is not cyclicly unifiable, therefore there exist some rule, V_j, which may not be applied. Consider the most general sequence of MRSs, $\rho_1 =$

$\langle [\,] \rangle, \rho_2, \ldots, \rho_j$, such that for $x < j$, each ρ_x is unifiable with R_x's head resulting in ρ_{x+1} and for $x = j$, ρ_j is not unifiable with R_j's head.

- For each $1 \leq x \leq j$, $\rho_x \sqsubseteq \sigma_{x+i}$:

  By induction on j,

  For $j = 1$, $[\,] \sqsubseteq \sigma_{i+1}$

  Assuming that the induction hypothesis holds for $j \leq j - 1$, so that for each $1 \leq x \leq j$, $\rho_x \sqsubseteq \sigma_{x+i}$.

  The induction step, $j = j$, by the induction hypothesis, $\rho_{j-1} \sqsubseteq \sigma_{j-1+i}$. ρ_j, σ_{j+i} are resulted by applying the same rule, R_{i+j-1}, on ρ_{j-1}, σ_{j-1+i} correspondingly. Thus $\rho_j \sqsubseteq \sigma_{j+i}$.

- $i + j > k + 1$:

  Assume towards a contradiction that $i + j \leq k + 1$, by the above, $\rho_j \sqsubseteq \sigma_{i+j}$, ρ_j is not unifiable with R_{i+j}'s head, therefore σ_{i+j} is not unifiable with R_{i+j}'s head, in contradiction to the fact that R is cyclicly unifiable.

- $i + j < i + k + 2$: since V is not a cyclicly unifiable sequence j must be less than $k + 2$, otherwise all of the sequence rules may be applied.

- None of the sequence rules may be applied consecutively more than twice:

  By the above, $\rho_1 = \langle [\,] \rangle, \rho_2, \ldots, \rho_j$ is the most general sequence of MRSs after applying $V_1, \ldots, V_{j-1}$ (or alternatively $R_{i+1}, \ldots, R_k, R_1, \ldots, R_{i+j-1}$).

  Since $\rho_j \sqsubseteq \sigma_{i+j}$ and ρ_j is not unifiable with V_j's head (or alternatively R_{i+j}) then neither σ_{i+j} nor any other resulting MRS after applying V_{j-1} is unifiable with V_j's head (since $i + j > k + 1$, V_{j-1} is applied after applying at least all of $V_1, \ldots, V_{j-2}$ consecutively). Thus, the sequence rules cannot be applied consecutively more than once.

Therefore, for each cyclicly unifiable rotation of the sequence none of the rules may be applied more than twice when applied consecutively together, thus none of the sequence rotations is 2-cyclicly-unifiable. □

Thus OLP_{JA_2} can be improved into OLP_{JA-rot}:

Definition 6.9. *A sequence of UR rules* $R = R_1, \ldots, R_k$ *(*$k \geq 1$*) is* rotating cyclicly unifiable *iff every cyclic rotation of* R *is cyclicly unifiable.*

Definition 6.10. *A grammar* G *is* OLP_{JA-rot} *iff it contains no rotating cyclicly unifiable sequences.*

The grammar of figure 6.1 is not OLP_{JA_2}, but it is OLP_{JA-rot}, since its only cycle $\{\rho_1, \rho_2\}$ is not a rotating cyclicly unifiable sequence.

An OLP_{JA-rot} grammar contains no rotating cyclicly unifiable sequences, therefore any OLP_{JA-rot} grammar G is also OLP_{JA-2} and thus parsing termination is guaranteed for OLP_{JA-rot} grammars.

OLP_{JA-rot} is decidable, the algorithm for deciding OLP_{JA-rot} is listed in figure 6.10.

An algorithm for deciding OLP_{JA_1}

scan_grammar(G): Boolean

Input: A unification grammar G.
Output: True iff G is OLP_{JA_1}.

Construct a directed unit-rules transitions graph, UTG, where

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

There exists a directed edge from vertex $\langle A_1, A_2 \rangle$ toward vertex $\langle B_1, B_2 \rangle$ iff B_1 is unifiable with A_2.

For each cycle $C = C_1, \ldots, C_k, C_1$ in UTG:

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

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

if $FS \sqcup H_1$ fails

Return $FALSE$

Return $TRUE$

Figure 6.2: An algorithm for deciding OLP_{JA_1}.

$$A^s = \Big[\text{F} : \langle\, tb\,,\, tb\,\rangle\Big]$$

$$\mathcal{R} = \left\{ \begin{array}{lll} \rho_1: & \Big[\text{F} : \langle\, tb \mid \boxed{1}\,\rangle\Big] & \longrightarrow \quad \Big[\text{F} : \boxed{1}\,\Big] \\ \rho_2: & \Big[\text{F} : \langle\,\rangle\Big] & \longrightarrow \quad \Big[\text{F} : \langle\, tb\,,\, tb\,\rangle\Big]\Big[\text{F} : tb\Big] \end{array} \right\}$$

$$\mathcal{L}(b) = \Big\{\Big[\text{F} : tb\Big]\Big\}$$

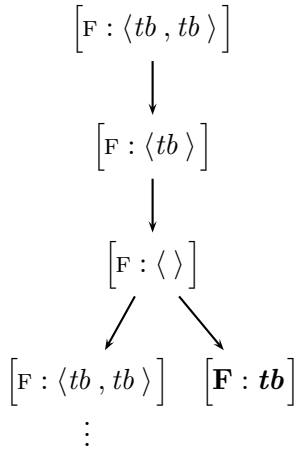
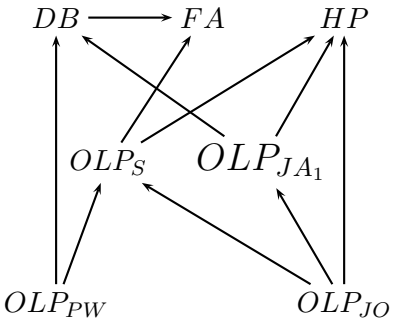


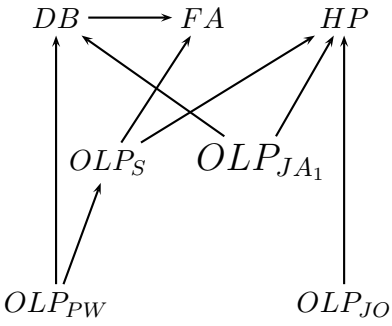
Figure 6.3: An example OLP_S grammar, G_S and its derivation form.

Hierarchy for **skeletal** grammars:

First definition of OLP_{JO} .



Second definition of OLP_{JO} .



Hierarchy for general **unification** grammars:

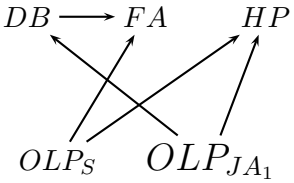


Figure 6.4: Revised hierarchy diagram, OLP_{JA_1}

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

Figure 6.5: Motivation for *UR* rules

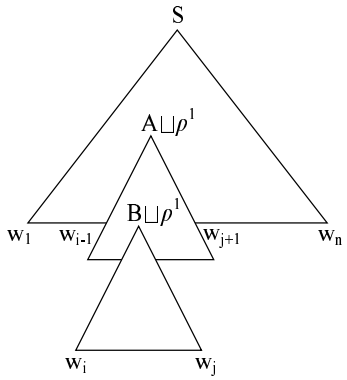


Figure 6.6: An example of derivation tree such that A dominates B, and they are both sharing the same yield and unifiable with ρ's head.

An algorithm for deciding OLP_{JA_2}

scan_grammar(G): Boolean

Input: A unification grammar G.
Output: True iff G is OLP_{JA_2}.

Construct E and UR as instructed above.

Construct a directed UR rules transitions graph, UTG, where

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

There exists a directed edge from vertex $\langle A_1, A_2 \rangle$ toward vertex $\langle B_1, B_2 \rangle$ iff B_1 is unifiable with A_2.

For each cycle $C = C_1, \ldots, C_k, C_1$ in UTG:

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

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

if $FS \sqcup H_1$ fails

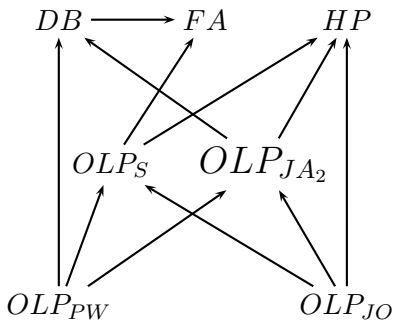
Return $FALSE$

Return $TRUE$

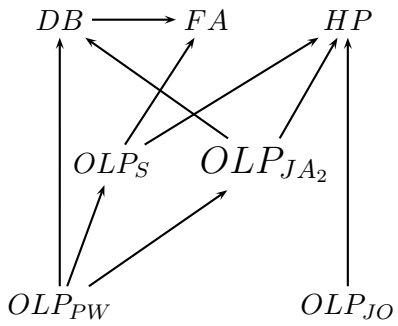
Figure 6.7: An algorithm for deciding OLP_{JA_2}.

Hierarchy for **skeletal** grammars:

First definition of OLP_{JO} .



Second definition of OLP_{JO} .



Hierarchy for general **unification** grammars:

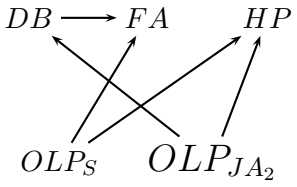


Figure 6.8: Revised hierarchy diagram, OLP_{JA_2}

An algorithm for deciding OLP_{JA-l}

$is\_olp_{ja-l}(G, l)$**: Boolean**

Input: A unification grammar G and a natural number $l > 0$.
Output: True iff G is OLP_{JA-l}.

Construct E and UR as instructed above.

Construct a directed UR rules transitions graph, UTG, where

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

There exists a directed edge from vertex $\langle A_1, A_2 \rangle$ toward vertex $\langle B_1, B_2 \rangle$ iff B_1 is unifiable with A_2.

For each cycle $C = C_1, \ldots, C_k, C_1$ in UTG:

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*$((V_1, \ldots, V_k)^l)$

Return $FALSE$

Return $TRUE$.

Figure 6.9: An algorithm for deciding OLP_{JA-l}.

An algorithm for deciding OLP_{JA-rot}

$is\_olp_{ja-rot}(G)$**: Boolean**

Input: A unification grammar G.
Output: True iff G is OLP_{JA-rot}.

Construct E and UR as instructed above.

Construct a directed UR rules transitions graph, UTG, where

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

There exists a directed edge from vertex $\langle A_1, A_2 \rangle$ toward vertex $\langle B_1, B_2 \rangle$ iff B_1 is unifiable with A_2.

For each cycle $C = C_1, \ldots, C_k, C_1$ in UTG:

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 $TRUE$

Return $FALSE$.

Figure 6.10: An algorithm for deciding OLP_{JA-rot}.

Chapter 7

Conclusions

In this research we have examined variants of the off-line parsability constraint. We used some grammar examples to emphasize the differences between the several OLP variants. There exist several variants of OLP suggested by several researchers some of which which were suggested without recognizing the existence of all other variants. We have made a comparative analysis of the several different variants for the first time and reached some hierarchy among them in terms of the classes of grammars each variant allows.

Several researchers (Haas, 1989; Torenvliet and Trautwein, 1995) conjecture that OLP is undecidable; it is undecidable whether a grammar satisfies the constraint, although none of them provides any proof of it. In chapter 5, we provide one of our main contributions, undecidability proofs for three of the undecidable OLP variants: Finite Ambiguity, Depth-Boundedness and Shieber's constraint. We use a reduction from the Turing machines halting problem on the empty input to a unification grammar and show that by deciding whether a grammar is OLP we are also able to decide whether a Turing machine M terminates on the empty input ϵ.

In chapter 6 we provide our main contribution, a novel OLP constraint, a decidable constraint. Our constraint is applicable to all unification grammar formalisms. It is more liberal than the existing decidable constraint and unlike all definitions that are applicable to general unification grammars it can be tested efficiently. In this chapter we also provide an algorithm for deciding whether a grammar satisfies the constraint as well as an evaluation of our constraint compared with the other OLP variants and place our constraint in the OLP hierarchy diagram. We also provide some improvements to our

constraint.

As for future work, while comparing our OLP constraint to Shieber's constraint, we have proved that a grammar satisfying Shieber's does not necessarily satisfy our constraint, but we have not been able to prove that other direction. We tend to believe that since these two condition impose unrelated restriction, they are incomparable; non of them contains one another, meaning there might also be some grammars satisfying our constraint which do not satisfy Shieber's constraint, but we have not been able to come up with such grammars.

We provide some improvements to our constraint, but we would like to get some more opinions and views on direction on how our constraint may be improved even more.

There exist some OLP variants which are applicable only to unification grammar formalisms which assume an explicit context-free backbone. General unification grammars do not necessarily yield an explicit context-free backbone. We intend to better understand these skeletons and perhaps devise an algorithm for extracting such a skeleton when it is not explicit.

References

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

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

Nissim Francez and Shuly Wintner. In preperation. Feature structure based linguistic formalisms.

G. Gazdar, E. Klein, G. Pullum, and I. Sag. 1985. *Generalized Phrase Structure Grammar.* Harvard University Press.

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

Mark Johnson. 1988. *Attribute-Value Logic and the Theory of Grammar*. CSLI Lecture Notes. CSLI.

Ronald M. Kaplan and Joan Bresnan. 1982. Lexical-functional grammar: A formal system for grammatical representation. *The MIT Press*, page 266.

Martin Kay. Functional unification grammar: A formalism for machine translation. pages 75–78.

Jonas Kuhn. 1999. Towards a simple architecture for the structure-function mapping. *Proceedings of the LFG99 Conference*.

Fernando C. N. Pereira and David H. D. Warren. 1980. Definite clause grammars for language analysis. In Karen Sparck-Jones Barbara J. Grosz and Bonnie Lynn Webber, editors, *Readings in Natural Language Processing*, pages 101–124. Morgan Kaufmann, Los Altos.

Fernando C. N. Pereira and David H. D. Warren. 1983. Parsing as deduction. *Proceedings of ACL - 21*.

Carl Pollard and Ivan A. Sag. 1986. *Head Driven Phrase Structure Grammar*. Center for the Study of Language and Information, Stanford, CA, USA.

Stuart M. Shieber. 1986. *An Introduction to Unification Based Approaches to Grammar*. CSLI Lecture Notes. CSLI.

Stuart M. Shieber. 1992. Constraint-based grammar formalisms. *MIT Press*.

Leen Torenvliet 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*.

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