GENERALIZED ID/LP GRAMMAR:
A FORMALISM FOR PARSING LINEARIZATION-BASED HPSG GRAMMARS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

By

Michael W. Daniels, B.A., B.S., M.A.

*****

The Ohio State University
2005

Dissertation Committee:
Professor W. Detmar Meurers, Advisor
Professor Christopher H. Brew
Professor Carl J. Pollard

ABSTRACT

This thesis motivates and describes the Generalized Immediate Dominance/Linear Precedence (GIDLP) formalism: a formalism capable of serving as a processing backbone for linearization-based grammars in the Head Driven Phrase Structure Grammar (HPSG) framework. Complementing the work on the formalism, the thesis defines and implements an efficient parsing algorithm for GIDLP grammars.

Representing a prominent tradition within HPSG, linearization-based HPSG assumes that the domain of word order can be larger than the local tree. This supports elegant and general linguistic analyses for (relatively) free word order languages, including the possibility of licensing discontinuous constituents.

For processing with an HPSG grammar, most systems depend on parsing algorithms that make use of a phrase structure backbone – a part of the grammar that has been set aside and given a distinguished role in the parsing process – thereby contrasting with those that view parsing as a general constraint solving task, where general methods for logical reasoning are to be applied to the constraints present in an HPSG grammar. Processing backbones support efficient parsing algorithms, but they restrict the class of HPSG theories that can be encoded to those employing a phrase structure backbone, which excludes linearization-HPSG grammars.

The GIDLP formalism solves the dilemma between the desire to encode linguistically general and elegant linearization-HPSG analyses and the need for a processing backbone. GIDLP allows linguists to specify grammars with linear precedence constraints that operate within explicitly declared word order domains extending beyond the local tree as well as immediate dominance rules in which the grammar writer can arrange the right-hand side as to minimize the number of parsing hypotheses that must be explored. The GIDLP parsing algorithm developed in the thesis supports efficient processing by making direct use of linear precedence constraints during parsing.

Dedicated to my parents

ACKNOWLEDGMENTS

No academic work, least of all a dissertation, is written in a vacuum, and it is my pleasure to thank those people who have helped me along the way. To distill six years into a short set of thank-yous is a daunting process, and I apologize in advance to those I have accidentally omitted.

I first thank my advisor, Detmar Meurers, for supporting and counseling me on countless levels, for serving as a source of motivation when I needed it, and for copious amounts of feedback on this thesis and everything else I've written.

I thank Chris Brew for serving on my committee, providing feedback, and for always getting me to think about the bigger picture – how my research would fit into the field as a whole.

I thank Carl Pollard as well for serving on my committee, for his insight and feedback on the most theoretical aspects of this thesis, and for leading the coordination seminar in AU00/WI01, in which I learned about linguistics than I had in any other class.

I thank Jerry Morgan for getting me interested in linguistics in the first place, during Fall 1995, and for teaching my very first class in computational linguistics. I similarly thank Georgia Green for introducing me to syntax and for serving as my undergraduate advisor. It was primarily through the experiences I had with them that I decided to pursue graduate study in linguistics – a decision I have never regretted.

I thank my three quarters' worth of LING 201 students for reminding me of what it's like to see linguistics from the outside.

Portions of this research have been presented at NLULP02, HPSG04, and COLING04. I thank the audiences at those conferences and all others who have provided comments and feedback, most notably Chris Brew, Wesley Davidson, Paul Davis, Dale Gerdemann, Jirka Hana, Detmar Meurers, Stefan Müller, Gerald Penn, and Carl Pollard.

I also thank the OSU Department of Linguistics and the Edward J. Ray Travel Award committee for the financial support that allowed me to travel to these conferences; and the OSU Graduate School and the National Science Foundation for providing the fellowships that supported my studies in general.

For general friendship, camaraderie, and (occasional) commiseration, I thank Wes Davidson, Hope Dawson, Anna Feldman, Vanessa Metcalf, Julia Papke, and Andrea Sims.

Finally, for providing me with an intensely rewarding set of experiences and diversions from my studies, I thank the Council of Graduate Students and those I have had the pleasure of working with on committees or elsewhere: Cathy Baack, Marilyn Blackwell, J. Briggs Cormier, Deb Cunningham, Zita Divis, Susan Fisher, Kerry Hodak, Allyson Lowe, Ron Meyers, Barb Pletz, Alan Randall, and W. Randy Smith.

Contents

Abstract iii

Dedication iv

Acknowledgments v

Contents vi

List of Figures ix

1 Introduction 1

2 Discontinuous Constituents in HPSG 3
- 2.1 Discontinuous Constituency . . . 3
  - 2.1.1 The Descriptivist Approach . . . 3
  - 2.1.2 The Transformational Response . . . 4
  - 2.1.3 Non-Derivational Approaches . . . 4
- 2.2 Linearization in HPSG . . . 6
  - 2.2.1 Reape's Domain Objects . . . 6
  - 2.2.2 Partial Compaction . . . 10
  - 2.2.3 Topological Fields . . . 12
    - 2.2.3.1 Topological Field Hierarchies . . . 14
  - 2.2.4 The Construction-based Approach . . . 15
- 2.3 Constituency and Locality . . . 17
- 2.4 Conclusion . . . 20

3 Existing Formalisms for Parsing With Discontinuous Constituents 21
- 3.1 Early Work . . . 21
- 3.2 Dependency Grammar . . . 22
- 3.3 DPSG . . . 25
- 3.4 Topological Parsing . . . 26
  - 3.4.1 Topological Dependency Parsing . . . 26
  - 3.4.2 Penn's Topological Parser . . . 30
- 3.5 Suhre's Linear Specification Language . . . 31
- 3.6 Conclusion . . . 32

4 The GIDLP Formalism 33
- 4.1 Generalizing ID/LP Grammar . . . 33
  - 4.1.1 Overview . . . 33
  - 4.1.2 Lexical Entries . . . 34
  - 4.1.3 Grammar Rules . . . 34
  - 4.1.4 Start Declaration . . . 34
  - 4.1.5 LP Constraints . . . 34
  - 4.1.6 Domain Declarations . . . 35
- 4.2 Examples . . . 36
  - 4.2.1 Mimicking a CFG . . . 36
  - 4.2.2 Illustrating Domain Formation . . . 37

4.3 Formal Model 38
4.4 Conclusion 39

5 Parsing GIDLP Grammars With Atomic Categories 41
5.1 Earley's Algorithm 41
5.1.1 Modifying Earley's Algorithm 43
5.2 The GIDLP Parsing Algorithm – Atomic Categories 43
5.2.1 Design Considerations 43
5.2.1.1 Edge Coverage 43
5.2.1.2 The Dot 44
5.2.1.3 LP Constraint State Representation 46
5.2.1.4 Domains in Earley's Algorithm 47
5.2.1.5 Edge Subsumption and Ambiguity Packing 48
5.2.1.6 Backpointer Classes 49
5.2.2 The Algorithm Itself 50
5.2.2.1 Grammar Compilation 50
5.2.2.2 Minimum Yield 51
5.2.2.3 Maximum Yield 52
5.2.2.4 Reachability 53
5.2.2.5 Parser Agenda 54
5.2.2.6 Prediction 55
5.2.2.7 Mask Computation 55
5.2.2.8 Completion 56
5.2.2.9 Order Constraint Updating 56
5.2.2.10 Activating dormant constraints 57
5.2.2.11 Updating token-based constraints 57
5.2.2.12 Updating and merging description-based constraints . . . 57
5.2.2.13 Completion – Final Steps 58
5.3 Sample Parses 58
5.3.1 Relatively-free Word Order 58
5.3.2 Domains and Dormant Constraints 64
5.4 Conclusion 68

6 Parsing GIDLP Grammars With Feature Structure Categories 69
6.1 Unification-Based Parsing 69
6.1.1 Restriction 70
6.1.1.1 Restriction To Ensure Termination 70
6.1.1.2 Restriction as Optimization 72
6.1.1.3 Dangers of Improper Restriction 73
6.1.2 Quick Check Vectors 73
6.2 Non-Local Information Flow 74
6.3 The GIDLP Parsing Algorithm for Feature Structure Categories 75
6.3.1 Grammar Compilation 75
6.3.2 Prediction 77
6.3.3 Completion 77
6.3.3.1 Non-Modular LP Constraints 78
6.3.3.2 Constraint Stores 79
6.4 Conclusion 79

7 Evaluation **80**
7.1 Linguistic Expectations . . . 80
7.1.1 Discontinuous Constituency as Grammar Optimization . . . 80
7.1.2 The Value of RHS Ordering . . . 81
7.1.3 Computational Complexity . . . 82
7.2 Implementation Notes . . . 82
7.2.1 Performance on Suhre's Grammars . . . 83
7.3 Efficiency . . . 83
7.3.1 Performance on Context-free Grammars . . . 83
7.3.2 Larger-scale Grammar Evaluation . . . 85
7.4 Conclusion . . . 88

8 Conclusion **89**
8.1 Future Work . . . 89
8.2 Conclusion . . . 90

Bibliography **91**

Citation Index **99**

List of Figures

2.1 English Particle Verbs 5
2.2 Parentheticals as Inducing Discontinuous Constituency 5
2.3 Blevins's Mobile Grammars 7
2.4 Domain Insertion and Domain Union 9
2.5 German Cross-Serial Dependency Tectogrammatic Structure 9
2.6 One Possible Phenogrammatic Structure 9
2.7 German Cross-Serial Dependency Domain Tree 11
2.8 Relative Clause Extraposition Within NP 11
2.9 Relative Clause Extraposition Within NP With Partial Compaction 13
2.10 Topological Fields 13
2.11 Topological Fields for Serbo-Croatian Clitics 16
2.12 Argument Attraction 19
2.13 Invisibility of Subject Properties 19

3.1 Three modes of combination 23
3.2 Modes of combination as used in parsing 23
3.3 Phrase Structure and Dependency Grammar Trees 23
3.4 Non-Projective Dependency Tree 24
3.5 Discontinuous Phrase Structure Grammar Tree 26
3.6 Dependency Tree 27
3.7 Topological Dependency Parse: Step 1 29
3.8 Topological Dependency Parse: Step 2 29
3.9 Topological Dependency Parse: Step 3 29
3.10 Topological Dependency Parse: Step 4 30

4.1 Parse Tree Showing Tectogrammatic Structure 38
4.2 Parse Tree Showing Phenogrammatic Structure 38
4.3 Formal Presentation of a GIDLP Grammar 40
4.4 Formal Derivation with respect to a GIDLP Grammar 40

5.1 Earley's Algorithm 42
5.2 Common Operations on Bitvectors 45
5.3 GIDLP Parsing Algorithm 51
5.4 Minimum Yield Example 52
5.5 Grammar Graph 53
5.6 Strongly Connected Component Graph 53
5.7 Example Grammar for Reachability 54
5.8 Category Reachability 54
5.9 Domain Reachability 55
5.10 Compiled Grammar 55

5.11 Mask Computation 57
5.12 Mask Computation Example 57
5.13 Immediate Precedence Frontier Updating 59
5.14 Immediate Precedence Frontier Merging 59

6.1 Example feature structure grammar 70
6.2 Parse tree for string *aaa* 70
6.3 Harrison and Ellison's grammar 74
6.4 Parse trees for strings *a* and *b* 74
6.5 Seiffert's grammar 76
6.6 Two passive edges created during parse of *edfg* 76
6.7 Result of completing passive edges in Figure 6.6 via rule 1 76
6.8 GIDLP Parsing Algorithm 77

7.1 Average Chart Size per Sentence Length 87
7.2 Chart Size per Sentence Length for Individual Sentences 87

8.1 Storing Grammar in a Trie 90

CHAPTER 1

Introduction

This thesis motivates and describes the Generalized Immediate Dominance/Linear Precedence (GIDLP) formalism: a formalism capable of serving as a processing backbone for linearization-based grammars in the Head Driven Phrase Structure Grammar (HPSG; Pollard and Sag 1994) framework. Representing a prominent tradition within HPSG, linearization-HPSG (Reape 1989, 1990, 1991b, 1993, 1994; Pollard et al. 1994; Kathol 1995; Richter and Sailer 1995; Reape 1996; Müller 1999a; Penn 1999; Donohue and Sag 1999; Bonami et al. 1999; Richter and Sailer 2001) centrally holds that the domain of word order (that is, the region within which word order is determined) may be larger than the local tree. As such, linearization-HPSG provides a way of modelling discontinuous constituents in HPSG.

The notion that constituents may be discontinuous, or that string concatenation might not be the only mode of constituent combination, has been a part of syntactic theory for some time, typically with a goal of analyzing languages with relatively free word order. Its use can be seen in frameworks such as Dependency Grammar (Bröker 1998; Plátek et al. 2001), Tree Adjoining grammar (Kroch and Joshi 1987; Rambow and Joshi 1994), Categorial Grammar (Dowty 1982; Bach 1983; Dowty 1996; Hepple 1994; Morrill 1995), Head Grammar (Pollard 1984), and those positing tangled trees (McCawley 1982; Huck 1985; Ojeda 1987; Blevins 1990). Furthermore, two German treebanks (Skut et al. 1997; Hinrichs et al. 2000) assume discontinuous constituents, and Müller (2004) argues that HPSG grammars for German which license discontinuous constituents should also be preferred on computational grounds. Put together, these strands of work justify the development of efficient processing strategies for linearization-HPSG grammars.

In the HPSG architecture, all features and types are equal – no constraint is encoded as being more important than any other. It is exactly this quality that gives linguists the freedom to propose pervasive changes to linguistic theory along the lines of linearization. But on the computational side, many systems, like the Linguistic Knowledge Builder (LKB; Copestake 1992) and the Attribute Logic Engine (ALE; Haji-Abdolhosseini and Penn 2003), depend on parsing algorithms that recognize a *processing backbone* – a part of the grammar that has been set aside and given a distinguished role in the parsing process. These systems contrast with those, like ConTroll (Götz and Meurers 1995, 1997a,b), that consider the task to be performed one of *constraint solving*, applying general methods for logical reasoning to the constraints present in an HPSG grammar.

While processing backbones lead to improved parsing efficiency, they also restrict the class of grammars that the system can handle. For instance, both LKB and ALE use a backbone of phrase-structure rules, which encode immediate dominance and linear precedence information in local trees. As phrase-structure rules cannot directly model discontinuous constituency, any system using a phrase-structure backbone cannot directly encode a linearization-HPSG grammar.

No grammar formalism is currently available that can provide a processing backbone for linearization-based HPSG grammars, and as a result, linguists who feel that discontinuous constituency will make their grammars more explanatory or more elegant are placed at a disadvantage by current technology. This thesis takes the remedy of this problem as its goal, and as such must accomplish two tasks: the description of a formalism that can serve as such a processing backbone, and the development of a parsing algorithm that uses this formalism.

The structure of the thesis is as follows: chapter 2 presents the basic aspects of linearization theory and illustrates how linearization has been used to analyze a wide variety of phenomena. Chapter 3 discusses previous work on parsing discontinuous constituents in a variety of frameworks and explains why a new formalism is necessary. Chapter 4 defines and illustrates the GIDLP formalism, and chapters 5 and 6 give the GIDLP parsing algorithm over atomic categories and feature-structure categories, respectively. Finally, chapter 7 characterizes the expressivity and efficiency of the formalism and parser through a variety of evaluations.

CHAPTER 2

Discontinuous Constituents in HPSG

To understand the choices made in designing the GIDLP formalism, one must understand the nature of discontinuous consitituency and the ways linguists state constraints on word order in the linearization-HPSG tradition. This chapter begins in section 2.1 with an introduction to the concept of discontinuous constituency. Section 2.2 then discusses the attempts to accomodate discontinuous constituency within GPSG and HPSG, including Reape's initial formulation of linearization for HPSG and the various refinements to Reape's approach that have been offered in the literature. The chapter concludes in section 2.3 with a discussion of the interactions between the concepts of discontinuity and locality.

2.1 Discontinuous Constituency

2.1.1 The Descriptivist Approach

Phrase structure grammar is centered around the notion that sentences are composed of subunits, or constituents, which themselves are either lexical items or can be further broken down into subsconstituents. The notion that some of these constituents could be discontinuous has been a part of the notion of constituency from the beginning. Bloomfield (1933), for instance, states that "the common form of any (two or more) complex forms is a . . . constituent of these complex forms" – making no requirement that a constituent be continuous.

One of the earliest restrictions on discontinuous constituents comes from Wells (1947), who states that discontinuous constituents should only be allowed in the event that the same series of words, if they appeared as a continuous sequence in some other environment, would make the same contribution to that environment. Under this principle, Wells gives analyses involving discontinuous constituents for the sentences in (1)a – (4)a respectively justified by the continuous counterparts in (1)b – (4)b.

(1) a. a **better** movie **than I expected**
    b. this movie is **better than I expected**

(2) a. an **easy** book **to read**
    b. this book is **easy to read**

(3) a. **His father**, according to John, **is the richest man in Scarsdale**.
    b. **His father is the richest man in Scarsdale.**

(4) a. **wake** your friend **up**
    b. **wake up** your friend

Under the assumption that each of the bold expressions in each pair makes the same contribution to its sentence, those in the (a)-sentences are justified as discontinuous constituents.

2.1.2 The Transformational Response

With the advent of transformational grammar (Chomsky 1957), however, discontinuous constituents – when they were discussed at all – were relegated to epiphenomena. Postal (1964), for instance, states that discontinuous constituents are merely continuous constituents that have been displaced by a permuting transformation. For instance, since *turned down* is a constituent in the base form in Figure 2.1a, it need not remain a constituent in the output in Figure 2.1b. Postal also objected to analyses positing discontinuous constituents on the grounds that such analyses were merely producing trees without describing how they could be generated – and that, in fact, the trees posited by such analyses were simply incapable of ever being generated.

The main exception to this view is McCawley (1982), who points out that this latter belief is an artifact of viewing trees as transcripts of a rewriting process, instead of as objects in their own right. From a set-theoretic perspective, following Partee et al. (1990), a *tree* is just a set of nodes linked so that one can start at any node and follow a single path to the *root* of the tree. Formally, a tree is a partially-ordered set (where the order is the inverse of the dominance relation) with a top in which every principal filter is a finite chain. An ordered tree then adds a second relation *precedence* and two conditions: (1) any two elements that are not dominance-comparable must be precedence-comparable; and conversely (2) if A precedes B, anything dominated by A must also precede anything dominated by B. Once trees are understood in this way, transformations can be seen as operating directly on trees, changing the dominance and precedence relations (but always subject to the constraint that the result remain a tree).

In McCawley's view, condition (2) unnecessarily rejects discontinuous constituents. Once it is abandoned, a transformation is free to modify the dominance relation without affecting the precedence relation; he classifies such transformations as *order-changing*, in constrast to *relation-changing* transformations, which only alter the precedence relation when required to by a concomitant change in dominance. Using this framework, McCawley presents the discontinuous analysis of English parentheticals shown in Figure 2.2. In essence, McCawley argues that the lack of evidence (in terms of traditional constituency tests like *do so* ellipsis) that the parenthetical *of course* ever acts as part of the verb phrase requires that it initially adjoin to the sentence as a whole before an order-changing transformation puts it in the proper linear order.

Thus discontinuous constituents were excluded for the most part from transformational grammar, despite having been a part of descriptivist approaches to syntax. These grounds for exclusion, however, relied on the derivational nature of transformational grammar, leaving subsequent non-derivational approaches free to adopt them.

2.1.3 Non-Derivational Approaches

Generalized Phrase Structure Grammar (GPSG) (Gazdar et al. 1985) was built upon the Immediate Dominance/Linear Precedence (ID/LP) (Gazdar and Pullum 1981) framework, which was originally developed as a way to abbreviate phrase structure grammars. ID/LP grammars include two types of rules. ID rules establish local trees: for instance, the rule A $\rightarrow$ B, C, D states that an A may "immediately and exhaustively dominate a B, a C, and a D" in some order. LP rules then specify the order in which categories are allowed to occur within each local tree; in particular, a rule of the form A $<$ B has the force "if A and B both appear on the righthand side of a[n ID] rule, then A precedes B".

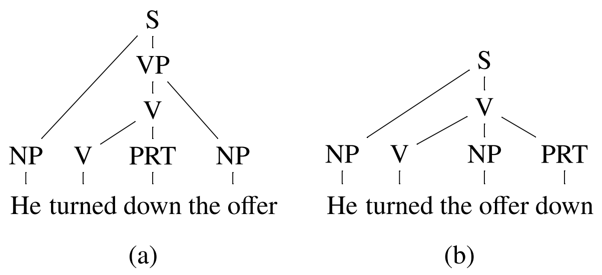


Figure 2.1: English Particle Verbs

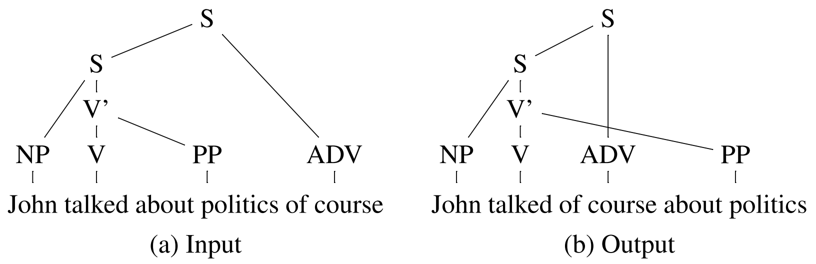

S
S
V'
NP
V
PP
ADV
John talked about politics of course
(a) Input
S
S
V'
NP
V
ADV
PP
John talked of course about politics
(b) Output

Figure 2.2: Parentheticals as Inducing Discontinuous Constituency

Initially, these rules were not taken to define a grammar directly; instead, they were seen as the input to a CFG-constructing procedure. Namely, for each ID rule, the set of its RHS permutations was formed and each checked for LP acceptability; each acceptable permutation became a rule in the resulting CFG. Since in the worst case, this transformation leads to a factorial expansion in the number of rules, considerable descriptive efficiency could be achieved through the ID/LP formalism.

It should be noted that the exclusion of discontinuous constituents in ID/LP grammar arises from this expansion procedure, not from the semantics of the ID and LP rules themselves – there is no explicit statement that the LHS of an ID rule must dominate a contiguous subset of the terminal yield, for instance. While it is true that LP constraints only apply within a local tree, this does not rule out discontinuous constituency – it only prevents the grammar writer from constraining the relative order of categories in different local trees.

This remains the case even in Shieber's (1984) formalization of the notion of *direct* ID/LP parsing (as opposed to the indirect mechanism described in the previous paragraphs); the only difference is that the notion of LP-acceptable permutation is moved into the definition of a derivation. Specifically, the string $\alpha A \beta$ derives $\alpha \gamma \beta$ whenever an ID rule $A \rightarrow \delta$ exists and γ is an LP-acceptable permutation of δ. The fact that the $\alpha \ldots \beta$ context remains unchanged enforces contiguity among constituents.

The fact that ID/LP grammar excludes discontinuous constituency was not seen as a disadvantage at that time, as it was seen as important at the time that ID/LP remain strongly equivalent to CFG (see, e.g. Pullum 1982). Yet while GPSG was being developed, research was also being conducted on potential modifications to ID/LP that would allow discontinuous constituents. In Blevins's (1990) mobile grammar formalism, LP rules are not restricted to apply within the local trees, but instead are taken to apply within each maximal projection. Under this approach, the rules in (5) – (8) would license either tree in figure 2.3.

(5) VP → V, NP

(6) V → V, PRT

(7) V < NP

(8) V < PRT

In evaluating the LP-acceptability of this tree, each maximal projection is examined, ignoring intermediate projections. Thus in both trees in figure 2.3, the VP node is seen to dominate the NP and PRT nodes, as well as the lowest V node. The relative order of these three constituents does not violate either LP constraint, and each tree is thereby licensed by the grammar.

2.2 Linearization in HPSG

The idea of discontinuous consitituency has been present from the beginnings of HPSG; it was even a part of Head Grammar (Pollard and Sag 1983; Pollard 1984), which served as the immediate predecessor of HPSG. In particular, Pollard and Sag's (1987) Scrambling Principle, given in (9), officially signaled that discontinuity was acceptable in HPSG.

(9) $$\textit{phrasal-sign} \Rightarrow \begin{bmatrix} \text{PHON} & \texttt{interleave-constituents(}\boxed{1}\texttt{)} \\ \text{DTRS} & \boxed{1} \end{bmatrix}$$

The function `interleave-constituents` is taken to be language specific, but is not otherwise defined. This gives HPSG grammars the capability to be constructed for any language regardless of their position on the scale from fixed word-order to completely-free word-order.

2.2.1 Reape's Domain Objects

The first presentation of linearization-HPSG appears in a series of papers by Reape (1989, 1990, 1991b, 1993, 1994, 1996), who challenges the tradition of giving configurational accounts of word order for German sentences – in effect proposing that German be treated as a relatively free word-order language rather than a relatively fixed one.

Reape's system can be seen as a generalization of Blevins's proposal to allow LP constraints to have domains larger than the local tree: while Blevins fixes the domain of LP constraints in the framework, Reape allows the grammar itself to specify the domains through a system of *domain objects*, represented by the DOM feature on a sign. A domain object is a list of signs, each of whose domain objects may or may not be empty, obeying the constraint that the concatenation of the PHON values of each sign is the PHON value of the sign itself. These domain objects are then interpreted as the domains within which linear precedence constraints apply. A sample DOM value is shown in (10); this will be abbreviated as in (11) throughout the remainder of this thesis.

(10) $$\left\langle \begin{bmatrix} \textit{sign} & \\ \text{PHON} & a \\ \text{DOM} & \langle\rangle \end{bmatrix}, \begin{bmatrix} \textit{sign} & \\ \text{PHON} & bc \\ \text{DOM} & \left\langle \begin{bmatrix} \textit{sign} & \\ \text{PHON} & b \\ \text{DOM} & \langle\rangle \end{bmatrix}, \begin{bmatrix} \textit{sign} & \\ \text{PHON} & c \\ \text{DOM} & \langle\rangle \end{bmatrix} \right\rangle \end{bmatrix}, \begin{bmatrix} \textit{sign} & \\ \text{PHON} & d \\ \text{DOM} & \langle\rangle \end{bmatrix} \right\rangle$$

(11) $\langle a, \langle b, c \rangle, d \rangle$

The key operation involved in building domain objects is *sequence union*, notated by $\bigcirc$ and defined in (12) (let ϵ be the empty list, ℓ_1, ℓ_2, ℓ_3 be lists, and $\circ$ be the list construction operator). (Note that while sequence union is formally a relation among three lists, it is presented here in functional notation for clarity.)

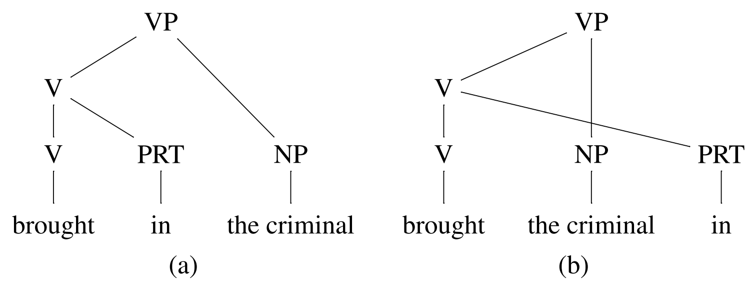


Figure 2.3: Blevins's Mobile Grammars

(12)
$$\begin{array}{lcl}\epsilon \bigcirc \epsilon = \epsilon & & \\ (x \circ \ell_1) \bigcirc \ell_2 = x \circ \ell_3 & \leftarrow & \ell_1 \bigcirc \ell_2 = \ell_3 \\ \ell_1 \bigcirc (x \circ \ell_2) = x \circ \ell_3 & \leftarrow & \ell_1 \bigcirc \ell_2 = \ell_3\end{array}$$

Informally, sequence union blends each of the input lists together into a single output list in such a way as to preserve the order of each input list, as illustrated in (13).

(13)

$$\begin{aligned} A &= \langle a, \langle b, c\rangle\rangle \\ B &= \langle\langle d, e\rangle, f\rangle \\ A \bigcirc B &= \langle a, \langle b, c\rangle, \langle d, e\rangle, f\rangle \\ &= \langle a, \langle d, e\rangle, \langle b, c\rangle, f\rangle \\ &= \langle a, \langle d, e\rangle, f, \langle b, c\rangle\rangle \\ &= \langle\langle d, e\rangle, a, \langle b, c\rangle, f\rangle \\ &= \langle\langle d, e\rangle, a, f, \langle b, c\rangle\rangle \\ &= \langle\langle d, e\rangle, f, a, \langle b, c\rangle\rangle \end{aligned}$$

This relation is then used to form the domain object of the mother in each local tree. Each daughter's domain object is taken – either as a list of signs directly, or as a singleton list containing the list of signs – and sequence unioned to form the mother's order domain. The former case is referred to as *domain union*, and the latter as *domain insertion*. This is formalized in the Domain Principle, given in (14).

(14) *functor-argument-structure* $\Rightarrow$

$$\left[\begin{array}{ll}\textsc{dtrs} & \left[\begin{array}{ll}\textsc{fun-dtr} & \boxed{1} \\ \textsc{arg-dtrs} & \left\langle \boxed{2}\left[\textsc{unioned}\ -\right], \ldots, \boxed{i}\left[\textsc{unioned}\ -\right]\right\rangle \bigcirc \left\langle \left[\begin{array}{ll}\textsc{unioned} & + \\ \textsc{dom} & \boxed{i+1}\end{array}\right], \ldots, \left[\begin{array}{ll}\textsc{unioned} & + \\ \textsc{dom} & \boxed{n}\end{array}\right]\right\rangle\end{array}\right] \\ \textsc{dom} & \left\langle\boxed{1}\right\rangle \bigcirc \left\langle\boxed{2}\right\rangle \bigcirc \cdots \bigcirc \left\langle\boxed{i}\right\rangle \bigcirc \boxed{i+1} \bigcirc \cdots \bigcirc \boxed{n}\end{array}\right]$$

The feature UNIONED marks its corresponding domain object for domain union when positive, and for domain insertion when negative. The effect of the principle is therefore to collect the domain objects of the daughters to be inserted, wrap each in a singleton list, and sequence union the results together with the domain objects of the daughters to be unioned. An element of a daughter domain object that gets unioned into its mother's domain object is often referred to as *liberated* (following Zwicky (1986)) or *emancipated*, although Reape did not use those terms. A domain object that undergoes insertion has similarly come to be called *isolated* or *compacted*.

In essence, domain insertion prevents the appearance of intervening material from other constituents, while domain union allows this. To see the difference, consider a local tree with mother S and two daughters: an NP with domain object $\langle \textit{the}, \textit{dog}\rangle$ and a VP with domain object $\langle \textit{ran}, \textit{quickly}\rangle$. If the NP is domain-inserted and the VP is domain-unioned, then the domain object for A must be a sequence union of $\langle\langle \textit{the}, \textit{dog}\rangle\rangle$ and $\langle \textit{ran}, \textit{quickly}\rangle$. The three possibilities (15) – (17) correspond to the trees shown in Figure 2.4.

(15) $\langle\langle \textit{the}, \textit{dog}\rangle, \textit{ran}, \textit{quickly}\rangle$

(16) $\langle \textit{ran}, \langle \textit{the}, \textit{dog}\rangle, \textit{quickly}\rangle$

(17) $\langle \textit{ran}, \textit{quickly}, \langle \textit{the}, \textit{dog}\rangle\rangle$

Thus to give a linguistic analysis using domain objects, a linguist must specify when daughters are domain unioned or domain inserted into their mother's domain. In Reape's approach, functor daughters – for instance, the heads in head-complement structures – are always domain-inserted, while argument daughters may be either inserted or unioned into the mother's domain.

It is important to notice that this approach does not force discontinuous constituency – by always using domain insertion, all constituents will be continuous: if each constituent is inserted into its mother's order domain as an uninterruptable unit, there will be no opportunity for parts of one constituent to occur amidst another. Discontinuity only arises when there is at least one application of domain union: material attached at a higher level of the syntactic tree will not be prevented from intervening between the contents of the unioned domain. In this light, domain union can be seen as a kind of bracket erasure, encoding the fact that the constituent being domain unioned is not a word order domain in and of itself.

We can illustrate the basics of this approach with the sentences in (18), which feature German cross-serial dependencies. In Reape's analysis, all share the same phrase structure, shown in the "syntax tree" – Reape's term for an unordered tree that only displays dominance – given in Figure 2.5. In each case *das Buch* and *zu lesen* form one (discontinuous) constituent that combines with *dem Mann* and *versprochen* to form another, which combines with *die Frau* and *hat* to form a sentence. As a result, each sentence in (18) corresponds to a different way of tangling Figure 2.5; Figure 2.6 shows the tangling that corresponds to (18a).

(18)	a.	daß das Buch$_1$	dem Mann$_2$	die Frau$_3$	zu lesen$_1$	versprochen$_2$	hat$_3$
		that the book	the man	the woman	to read	promised	has
	b.	daß das Buch$_1$	die Frau$_3$	dem Mann$_2$	zu lesen$_1$	versprochen$_2$	hat$_3$
		that the book	the woman	the man	to read	promised	has
	c.	daß dem Mann$_2$	das Buch$_1$	die Frau$_3$	zu lesen$_1$	versprochen$_2$	hat$_3$
		that the man	the book	the woman	to read	promised	has
	d.	daß dem Mann$_2$	die Frau$_3$	das Buch$_1$	zu lesen$_1$	versprochen$_2$	hat$_3$
		that the man	the woman	the book	to read	promised	has
	e.	daß die Frau$_3$	das Buch$_1$	dem Mann$_2$	zu lesen$_1$	versprochen$_2$	hat$_3$
		that the woman	the book	the man	to read	promised	has
	f.	daß die Frau$_3$	dem Mann$_2$	das Buch$_1$	zu lesen$_1$	versprochen$_2$	hat$_3$
		that the woman	the man	the book	to read	promised	has

'… that the woman promised the man to read the book.'

Reape uses the constraints in (19) to account for the orders in (18).

(19) a. $\left[\textsc{dom}\right] \Rightarrow \left[\textsc{dom}\ \ \text{NP} \prec \text{V}\right]$

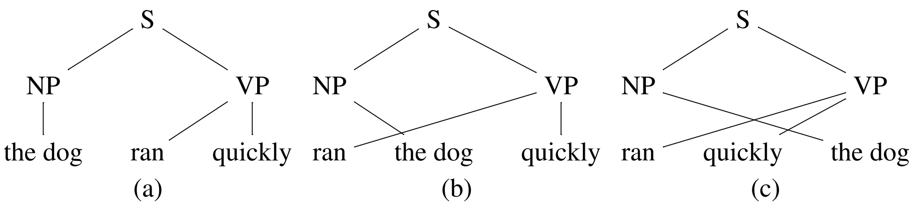


Figure 2.4: Domain Insertion and Domain Union

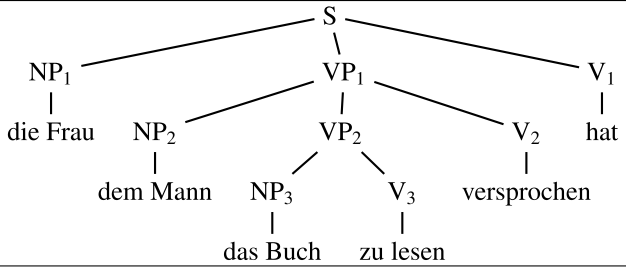


Figure 2.5: German Cross-Serial Dependency Tectogrammatic Structure

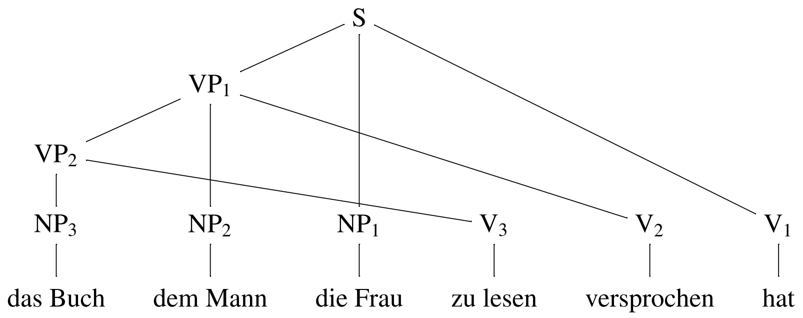


Figure 2.6: One Possible Phenogrammatic Structure

b. $\left[\text{DTRS|HEADDTR } \boxed{1}\text{V}\left[\text{INV } -\right]\right] \Rightarrow \left[\text{DOM } \text{V} \leq \boxed{1}\right]$

Constraint (19a) can be read "if a feature structure contains the attribute DOM, the value of that attribute must respect the constraint $NP < V$"; it indicates noun phrases precede verbs within all domains. Constraint (19b) similarly requires non-inverted verbs to precede all other verbs in the same domain.

One possible linearization of Figure 2.5 (corresponding to the tangling in Figure 2.6 that licenses (18a)) under these constraints is illustrated in the "domain tree" – Reape's term for an unordered tree whose nodes are domain objects and where one node immediately dominates another iff the former's category immediately dominates the latter's in the corresponding syntax tree – shown in Figure 2.7. In each node, the outer bracket represents the sign itself, while its contents show the contents of the sign's domain object. The infinitival verb *zu lesen* has been inserted into its mother's domain object (as have all the noun phrases); as a result, no other material may occur between *zu* and *lesen*. The verb phrases in this tree (namely, *das Buch zu lesen* and *das Buch dem Mann zu lesen versprochen*) are unioned into their mothers' domains, allowing material from a higher level to intervene; as a result, the verb phrase constituents are absent from the higher domain objects. The linear precedence constraints that Reape provides ensure that all of the nouns occur before all of the verbs, and that each verb occurs after the verb it governs. The relative order of the nouns, however, is not constrained, allowing for all six of the sentence possibilities described earlier to be licensed.

2.2.2 Partial Compaction

Reape's initial work on linearization allows a daughter's domain object to contribute to its mother's domain object in two ways: domain insertion and domain union. Kathol and Pollard (1995) criticize this dichotomy as being too coarse, disallowing the possibility that a construction could have some arguments inserted into their mother's domain and others unioned in. From this observation developed a strand of work (Kathol and Pollard 1995; Kathol 1995; Yatabe 1996; Lee 1999; Campbell-Kibler 2002; Maekawa 2004) that introduced and developed the notion of *partial compaction*, in which only a portion of the signs on the daughters' domain objects are inserted as a unit into the mother's domain object, the remainder being domain unioned in. In effect, while Reape's approach allows word order domains that are larger than the local tree, via constituents that do not correspond to domain objects, partial compaction allows word order domains that are smaller than the local tree, via domain objects that do not correspond to constituents.

To illustrate the need for this, Kathol and Pollard take as their exemplar the phenomenon of extraposition in German, as shown in (20).

(20)

einen	Hund	füttern	der	Hunger	hat
a	dog	feed	that	hunger	has

'to feed a dog that is hungry'

The relative clause *der Hunger hat* has been extraposed from the noun phrase *einen Hund der Hunger hat*, allowing the verb *füttern* to intervene. The structure assumed for the structure (following (Nerbonne 1994)) is shown in Figure 2.8. In essence, the relative clause itself is marked $\left[\textsc{unioned}\ -\right]$, indicating that it should be inserted into its mother's domain as a single unit. The modified NP is then marked $\left[\textsc{unioned}\ +\right]$, indicating that it is unioned into its mother's domain, allowing *füttern* to intervene. Furthermore, their modification of Nerbonne's LP constraint[1] in (21) requires the relative clause to appear last in its domain, so *füttern* must precede the relative clause.

(21) $\left[\ \right] < \left[\textsc{extra}\ +\right]$

The problem, as Kathol and Pollard point out, arises when one considers the related phrase in (22).

(22)

an	einen	Hund	denken	der	Hunger	hat
of	a	dog	think	that	hunger	has

'to think of a dog that is hungry'

They observe that the preposition *an* must combine with the rest of the noun phrase at a higher level than the relative clause (since the relative clause only modifies the NP within the PP). At the same time, the sequence *an einen Hund* must correspond to a single domain object, based on the general ordering properties of German prepositional phrases. This

[1]Nerbonne stated his version of the constraint as $\left[\textsc{extra}\ -\right] < \left[\textsc{extra}\ +\right]$; the version in (21) allows there to be more than one $\left[\textsc{extra}\ +\right]$ object in a domain.

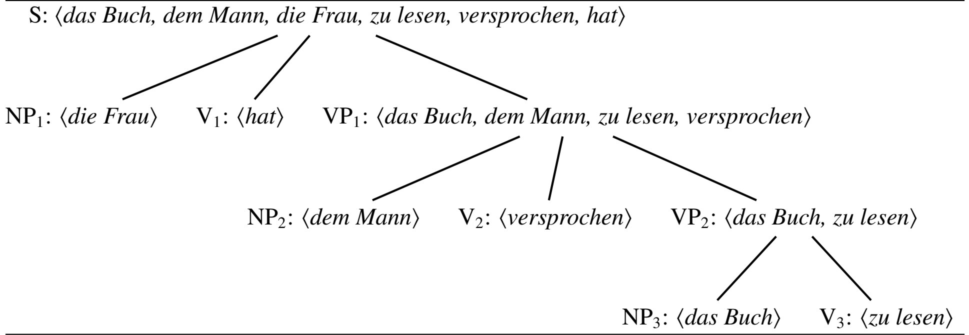

S: ⟨das Buch, dem Mann, die Frau, zu lesen, versprochen, hat⟩
NP1: ⟨die Frau⟩
V1: ⟨hat⟩
VP1: ⟨das Buch, dem Mann, zu lesen, versprochen⟩
NP2: ⟨dem Mann⟩
V2: ⟨versprochen⟩
VP2: ⟨das Buch, zu lesen⟩
NP3: ⟨das Buch⟩
V3: ⟨zu lesen⟩

Figure 2.7: German Cross-Serial Dependency Domain Tree

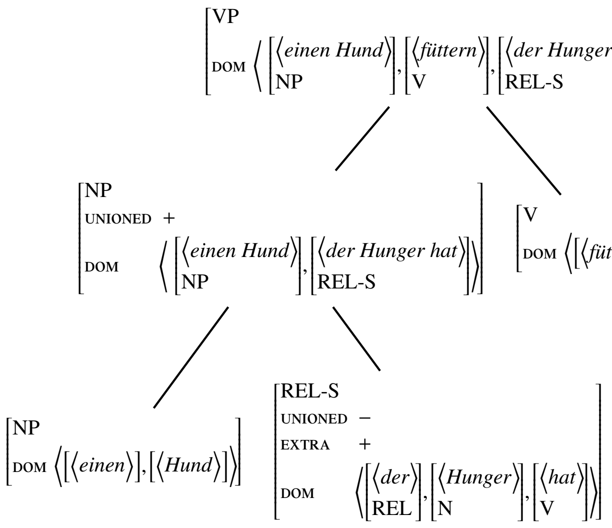

VP
DOM ⟨[⟨einen Hund⟩ NP], [⟨füttern⟩ V], [⟨der Hunger hat⟩ REL-S]⟩
NP
UNIONED +
DOM ⟨[⟨einen Hund⟩ NP], [⟨der Hunger hat⟩ REL-S]⟩
V
DOM ⟨[⟨füttern⟩]⟩
NP
DOM ⟨[⟨einen⟩], [⟨Hund⟩]⟩
REL-S
UNIONED −
EXTRA +
DOM ⟨[⟨der⟩ REL], [⟨Hunger⟩ N], [⟨hat⟩ V]⟩

Figure 2.8: Relative Clause Extraposition Within NP

indicates that the preposition should attach at a lower level than the relative clause, since in Reape's approach domain objects can only be formed from local trees. As a result, this framework cannot analyze such a construction.

Their solution, referred to as partial compaction, is to propose a system in which it is possible for some of the arguments to be inserted and the rest unioned into the mother domain. Partial compaction is defined through the `p-compaction` relation, given in (23).

(23) $\texttt{p-compaction}(\boxed{1}, \boxed{2}, \boxed{3}) \equiv$

$$\boxed{1}\begin{bmatrix} \textit{sign} & \\ \text{SYNSEM} & \boxed{4} \\ \text{DOM} & \boxed{6} \end{bmatrix} \wedge$$
$$\boxed{2}\begin{bmatrix} \textit{dom-obj} & \\ \text{SYNSEM} & \boxed{4} \\ \text{PHON} & \boxed{7} \end{bmatrix} \wedge$$
$$\boxed{6} = \boxed{5} \bigcirc \boxed{3} \wedge$$
$$\boxed{7} = \texttt{joinphon}(\boxed{5})$$

Here, the `joinphon` operation simply takes a list of domain object entries and concatenates each PHON value. Seen procedurally, the effect of `p-compaction` is to remove some items

($\boxed{3}$) from the sign's ($\boxed{1}$) domain object ($\boxed{6}$) and create a new domain object ($\boxed{2}$) from the items remaining on the sign's domain list ($\boxed{5}$) – in other words, all of $\boxed{1}$ except for $\boxed{3}$ will be compacted into $\boxed{2}$. Defined this way, it is clear that partial compaction generalizes the distinction between domain union and domain insertion: setting $\boxed{3}$ to empty gives the effect of domain insertion and setting $\boxed{3} = \boxed{1}$ gives the effect of domain union.

Thus in (22), the desired effect is obtained by choosing to union (rather than insert) the relative clause when the noun phrase combines with the verb. This is expressed through the schema in (24).

(24)
$$\begin{bmatrix} \textsc{dom} & \boxed{5} \\ \textsc{hd-dtr} & \begin{bmatrix} \textsc{dom} & \boxed{4} \end{bmatrix} \\ \textsc{comp-dtrs} & \boxed{1} \end{bmatrix} \wedge$$
$$\texttt{p-compaction}(\boxed{1}, \boxed{2}, \boxed{3}) \wedge$$
$$\boxed{5} = \langle \boxed{2} \rangle \bigcirc \boxed{3} \bigcirc \boxed{4} \wedge$$
$$\boxed{3}\, \mathit{list}(\begin{bmatrix} \textsc{synsem} & \begin{bmatrix} \textsc{extra} & + \end{bmatrix} \end{bmatrix}) \wedge$$
$$(\boxed{3}\, \mathit{elist} \vee \boxed{2} \begin{bmatrix} \textsc{synsem} & \neg \begin{bmatrix} \textsc{head} & \mathit{verb} \\ \textsc{subcat} & \mathit{elist} \end{bmatrix} \end{bmatrix})$$

The effect of the third attached clause is to exclude any extraposed elements from domain insertion, while the fourth clause forces the list of excluded elements to be empty when the complement is itself a clause.

This is illustrated in Figure 2.9. As before, the relative clause is domain-inserted into the $\overline{\text{N}}$, which is domain-unioned into the NP and PP domains. At the level where the PP combines with the verb, the schema in (24) applies. The argument domains for *an*, *einen*, and *Hund* are compacted, with the result being unioned with the domains for *denken* and *der Hunger hat*.

2.2.3 Topological Fields

The LP constraints seen so far make primary reference to syntactic categories: nouns precede verbs, or verbs follow their arguments. A quite different set of ordering constraints is found in the order domain-based theory developed by Kathol (1995, 2000) to account for the basic clause structure of German. In particular, Kathol takes as his basis the topological field approach prevalent in traditional analyses of German grammar (see, e.g. Drach 1937; Höhle 1986). In this approach, any German sentence can be divided from left to right into fields whose contents will be homogeneous across sentences, despite variation in word order. Linguists differ as to the number and nature of the fields; Kathol illustrates his approach with a simplified system of five fields – the *forefield* (vf), *left bracket* (lb), *middle field* (mf), *right bracket* (rb), and *postfield* (nf). These fields analyze the three major clause types in German as illustrated in Figure 2.10 (taken from Kathol 2000).

In a declarative main clause like (A), the subject occupies the forefield, the finite verb is the left bracket, and complements and modifiers typically appear in the middle field. Sentence (B) is similar, except that here the the forefield holds the direct object of the sentence. In general, the forefield may hold no more than one of the verb's arguments or modifiers; the question of which one is generally taken to be contextually determined. The right bracket holds the verbs (recursively) governed by the finite verb, if any. Each of these sentences illustrates the *verb-second* pattern.

In a subordinate clause like (C), the fields are occupied in a slightly different fashion. Here, the left bracket contains the complementizer *daß*, and both the finite verb and the verbs it (recursively) governs occur in the right bracket; the postfield contains extraposed elements. This sentence illustrates the *verb-final* pattern.

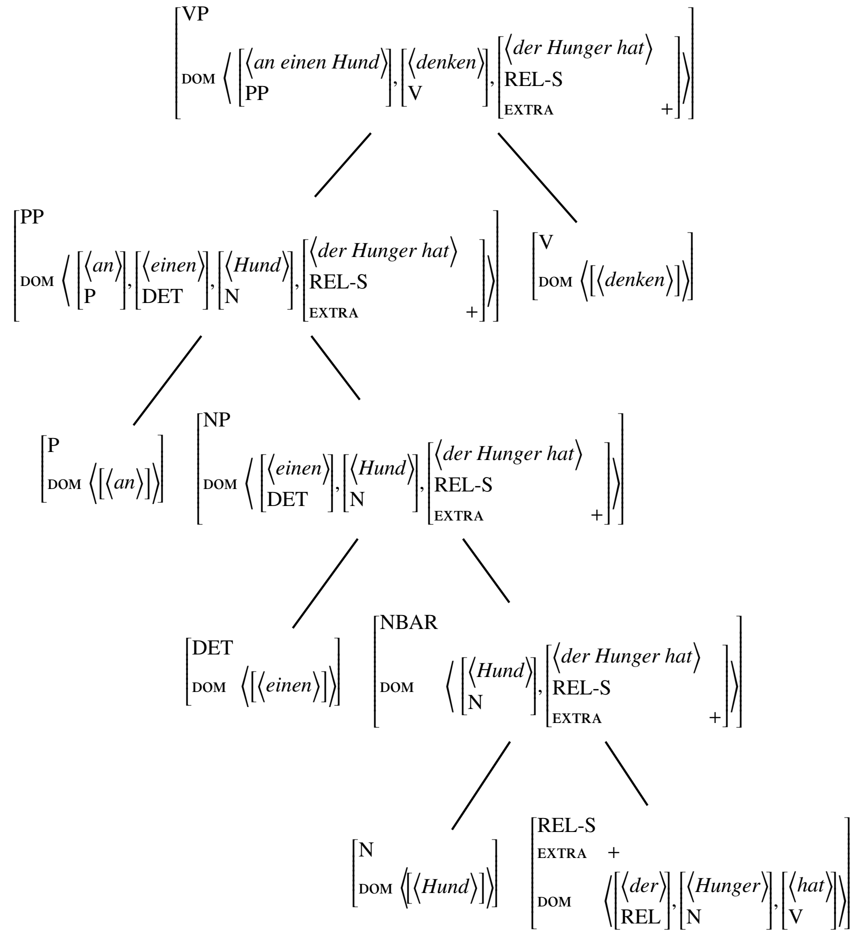


Figure 2.9: Relative Clause Extraposition Within NP With Partial Compaction

	vf	lb	mf	rb	nf
A)	Lisa	gießt	die Blume		
	Lisa	waters	the flower		
B)	die Blume	muß	Lisa	gießen	
	the flower	must	Lisa	water	
C)		daß	Lisa die Blume	gießen würde	morgen abend
		that	Lisa the flower	water would	tomorrow night
D)		würde	jemand die Blume	gießen	morgen abend?
		would	someone the flower	water	tomorrow night?

Figure 2.10: Topological Fields

Finally, (D) is an example of the *verb-initial* pattern typical of main clause polar interrogative, imperative, and exclamatory sentences, as well as antecedents of conditionals. These are characterized by an empty forefield and a left bracket containing the finite verb.

The key to this approach is that knowing which words can appear in which fields is enough to largely determine the word order of the sentence. Thus using Reape's domain objects, Kathol need only state a single linear precedence constraint: $vf \prec lb \prec mf \prec rb \prec pf$. The root domain objects for the three sentence patterns described above therefore appear as follows:

(25) $\left\langle \begin{bmatrix} \textit{vf} & \\ \textsc{phon} & \textit{`die Blume'} \end{bmatrix}, \begin{bmatrix} \textit{lb} & \\ \textsc{phon} & \textit{`gießt'} \end{bmatrix}, \begin{bmatrix} \textit{mf} & \\ \textsc{phon} & \textit{`Lisa'} \end{bmatrix} \right\rangle$

(26) $\left\langle \begin{bmatrix} \textit{lb} & \\ \textsc{phon} & \textit{`daß'} \end{bmatrix}, \begin{bmatrix} \textit{mf} & \\ \textsc{phon} & \textit{`Lisa'} \end{bmatrix}, \begin{bmatrix} \textit{mf} & \\ \textsc{phon} & \textit{`die Blume'} \end{bmatrix}, \begin{bmatrix} \textit{rb} & \\ \textsc{phon} & \textit{`gießt'} \end{bmatrix} \right\rangle$

(27) $\left\langle \begin{bmatrix} \textit{lb} & \\ \textsc{phon} & \textit{`gießt'} \end{bmatrix}, \begin{bmatrix} \textit{mf} & \\ \textsc{phon} & \textit{`Lisa'} \end{bmatrix}, \begin{bmatrix} \textit{mf} & \\ \textsc{phon} & \textit{`die Blume'} \end{bmatrix} \right\rangle$

As before, the noun phrase constituents have been inserted into the verb phrase's domain object rather than unioned, representing the fact that (in the general case) no material from another constituent may intervene between a determiner and its noun. The domain shown in (26) is one of two possible orders: barring any other constraints on middle-field constituents and in the presence of the right pragmatic conditions, the alternate order [*daß*, *die Blume*, *Lisa*, *gießt*] would also be licensed.

2.2.3.1 Topological Field Hierarchies

Penn (1999) presents an analysis of Serbo-Croatian clitic placement that illustrates a more complicated use of topological fields centering around the notion of *Wackernagel's position*. Broadly defined, phenomena involving Wackernagel's position require a clitic or pronoun to appear after either the first constituent (termed *2D placement*, following (Halpern 1995)) or the first word in a clause (termed *2W placement*. Penn discusses a certain class of clitics in Serbo-Croatian that have this property; one of them is the *je* that appears in the following examples:

(28)	Taj	čovek	je	video	Mariju
	that	man	3SG	saw	Mary

'That man saw Mary.'

(29)	Taj	je	čovek	video	Mariju
	that	3SG	man	saw	Mary

'That man saw Mary.'

(30)	U	lepi	grad	je	stigao
	in	beautiful	city	3SG	arrived

'He has arrived in the beautiful city.'

(31)	U	lepi	je	grad	stigao
	in	beautiful	3SG	city	arrived

'He has arrived in the beautiful city.'

Comparing (28) and (29), we see that the clitic *je* can have both 2D and 2W placement, interrupting the constituent *taj čovek* in the latter case. Examples (30) and (31) show how this interacts with the presence of the proclitic *u*, which does not "count" as a word when determining second-word position. Phenomena like this were one of the motivations behind the advocacy of head wrapping (see, e.g., Pollard 1984; Dowty 1997) as a mode of combination in which a functor and argument combine by placing the argument adjacent to the head of the functor.

As with Kathol, Penn's analysis involves a topological field model; his fields, however, are arranged into a hierarchy. For example, the topological fields used to analyze (30) are given in Figure 2.11. The *pre-cf* and *post-cf* fields divide up the constituent interrupted by the clitic: prosodic constraints determine what can fit in *pre-cf* and the rest of that constituent is assigned to *post-cf*. The full *cf* (clitic field) contains subfields for each clitic that can occur in the clitic field; this allows Penn to account for the fact that, in sentences where more than one Wackernagel clitic is present, the clitics always appear in a fixed relative order. Finally, the remainder field *rf* contains all other top-level domain objects (which are relatively unconstrained in terms of word order).

2.2.4 The Construction-based Approach

The examples of linearization HPSG presented so far all focus on phenomena that can be analyzed with global LP constraints – constraints that are taken to apply universally within a given grammar (Gazdar and Pullum 1981). Linguists have also worked with languages that appear to demand construction-specific word order constraints. One example of an analysis for such a language is Donohue and Sag's (1999) analysis of Warlpiri, an Australian language. Donohue and Sag make the following generalizations about Warlpiri:

1. An auxiliary element (AUX) must appear in every Warlpiri sentence containing a verb. It generally occupies second position, but it may be sentence-initial in connected speech (as opposed to isolated utterances).
2. Auxiliaries with a monosyllabic base encliticize to the preceding word.
3. NPs (which consist of a noun and any number of modifiers) may be realized in two ways:
   - a construction in which case marking appears only on the right-most element. This construction can appear in the pre-AUX position; it cannot be interrupted by material from other constituents.
   - a construction in which case marking appears on each element. This construction cannot appear in the pre-AUX position (although any of its elements alone can) and can be realized discontinuously.

The generalizations regarding noun phrases are illustrated[2] in (32) – (34).

(32)	kurdu-jarra-rlu	ka-pala	maliki	wajili-pi-nyi	wita-jarra-rlu.
	child.DUAL.ERG	PRES.3DUSUB	dog.ABS	chase.NPAST	small.DUAL.ERG

‘Two small children are chasing the dog.’

(33)	kurdu	wita-jarra-rlu	ka-pala	maliki	wajili-pi-nyi.
	child	small.DUAL.ERG	PRES.3DUSUB	dog.ABS	chase.NPAST

‘The two small children are chasing the dog.’

(34) *	kurdu	ka-jana	yalumpu-rlu	maliki-patu	jiti-rni.
	child	PRES.3PLOBJ	that.DUAL.ERG	dog.PL	tease.NPAST

‘Those two children are teasing the dogs.’

In (32), the noun phrase *kurdu-jarra-rlu wita-jarra-rlu* appears discontinuously; each element bears case marking, and it is acceptable for the first element to occupy initial position. In contrast, the noun phrase *kurdu wita-jarra-rlu* in (33) exhibits case marking only on the second element, and the entire noun phrase occupies initial position; sentence (34) demonstrates that such a noun phrase cannot appear discontinuously.

[2]All examples in this section are quoted from (Donohue and Sag 1999); in the glosses, periods separate the grammatical information carried by each morpheme.

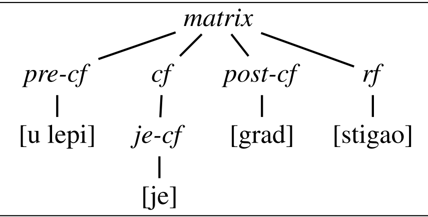


Figure 2.11: Topological Fields for Serbo-Croatian Clitics

Integrating Reape's notion of a DOM object into the constructional variant of HPSG advocated in (Sag 1999), Donohue and Sag create constructional analogues of domain union (35) and domain insertion (36).

(35) $\textit{liberating-cx} \Rightarrow \begin{bmatrix} \textit{hd-cx} & \\ \text{MOTHER} & \begin{bmatrix}\text{DOM} & \delta_0 \bigcirc \cdots \bigcirc \delta_n\end{bmatrix} \\ \text{HD-DTR} & \begin{bmatrix}\text{DOM} & \delta_0\end{bmatrix} \\ \text{NON-HD-DTRS} & \left\langle \begin{bmatrix}\text{DOM} & \delta_1\end{bmatrix}, \ldots, \begin{bmatrix}\text{DOM} & \delta_n\end{bmatrix} \right\rangle \end{bmatrix}$

(36) $\textit{compacting-cx} \Rightarrow \begin{bmatrix} \text{MOTHER} & \begin{bmatrix} \text{SYNSEM} & \boxed{1} \\ \text{DOM} & \left\langle \begin{bmatrix}\text{SYNSEM} & \boxed{1}\end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix}$

A liberating construction is one in which the mother's domain object is formed by shuffling the domain objects of its daughters, which may therefore freely intermingle with each other; that is, a construction which liberates all of its daughters. A compacting construction, however, must correspond to a single domain object, the details of which are left to subtypes of *compacting-cx* to determine.

One such subtype of *compacting-cx* is given in (37).

(37) $\textit{compac-mod-nom-cx} \Rightarrow \begin{bmatrix} \text{MOTHER} & \begin{bmatrix}\text{DOM} & \left\langle \begin{bmatrix}\text{DOM} & \left\langle \boxed{1}, \boxed{2} \right\rangle\end{bmatrix} \right\rangle\end{bmatrix} \\ \text{NON-HD-DTRS} & \left\langle \boxed{1}\begin{bmatrix}\text{CASE} & \textit{none}\end{bmatrix} \right\rangle \\ \text{HD-DTR} & \boxed{2} \end{bmatrix}$

This construction consists of a caseless modifier immediately preceding a noun. As a subtype of *compacting-cx*, the construction's mother category must contain a single domain object on its DOM list; this construction specifies that that element must consist solely of the single non-head daughter followed by the head daughter. As a result, no other material may intervene between the two daughters.

The corresponding construction in (32) is analyzed as a subtype of *liberating-cx*, given in (38).

(38) $\textit{liber-mod-nom-cx} \Rightarrow \begin{bmatrix} \text{NON-HD-DTRS} & \left\langle \begin{bmatrix}\text{CASE} & \boxed{1}\end{bmatrix} \right\rangle \\ \text{HD-DTR} & \begin{bmatrix}\text{CASE} & \boxed{1}\end{bmatrix} \end{bmatrix}$

As the relationship between the mother's DOM list and the daughters' DOM lists is already specified by the constraints on *liberating-cx*, this construction need only impose a requirement for case identity.

Turning to the requirement that no more than one element precede the auxiliary, Donohue and Sag posit a second subtype of *compacting-cx* as well as two linear precedence rules; these are given in (39) – (41).

(39) $$ip\text{-}cl \Rightarrow \begin{bmatrix} \text{MOTHER} & \left[\text{DOM} \left\langle \left[\text{DOM } \texttt{order}(\boxed{0},\ldots,\boxed{n})\right]\right\rangle\right] \\ \text{NON-HD-DTRS} & \left\langle \boxed{1}\left[\text{SS } \boxed{s_1}\right],\ldots,\boxed{n}\left[\text{SS } \boxed{s_n}\right]\right\rangle \\ \text{HD-DTR} & \boxed{0}\begin{bmatrix} \textit{word} & \\ \text{HEAD} & \textit{aux} \\ \text{COMPS} & \left\langle \boxed{s_1},\ldots,\boxed{s_n}\right\rangle \end{bmatrix}\end{bmatrix}$$

(40) $\text{AUX} \prec \left[\text{FOC } -\right]$

(41) $\left[\text{FOC } +\right] \prec [\;]$

Donohue and Sag do not discuss the `order` relation; it presumably yields a permutation of its arguments that does not violate any LP constraints. Constraint (40) requires that the auxiliary precede all unfocused elements, and (41) requires that a focused element precede all others.

Put together, these constraints account for the data given. In (32), each word of the noun phrase *kurdu-jarra-rlu wita-jarra-rlu* bears case marking, and so the phrase itself is licensed as a *liber-mod-nom-cx*. The two words are shuffled together with the rest of the arguments of the auxiliary, and *kurdu-jarra-rlu* may appear before the auxiliary as long as it is focused. In contrast, the noun phrase *kurdu wita-jarra-rlu* exhibits case marking only on the second element, and can only be licensed by *compac-mod-nom-cx*, forming a single domain object. If this domain object is focused, as in in (33), the entire noun phrase may occupy initial position. Finally, as no construction will license a domain object containing only a single caseless modifier, sentence (34) is ungrammatical.

2.3 Constituency and Locality

Once constituency has been rejected as the sole determinant of word order, a question arises as to its linguistic significance. Indeed, frameworks like categorial grammar and dependency grammar either relegate to an epiphenomenon or entirely reject the notion of constituency as linguistically meaningful. Yet independent motivation for the notion of constituency can be found in the principle of locality: the concept that the syntactic properties at one level of structure are determined solely by the grammatical properties of local levels of structure. This general principle can take many forms: for instance, locality can be described as requiring a mother's syntactic properties to be determined by the its daughters' properties and not its daughters' daughters' properties; or that a category's properties are determined by its dependents' properties and not its dependents' dependents' properties. However it is phrased, locality is meaningless in the absence of a metric for 'distance' that can differentiate 'local' from 'non-local', and constituency is one way of providing this metric.

In a critique of Reape's assumptions regarding German clause structure (as presented in section 2.2.1), Kathol (2000) observes that Reape's structure is incompatible with this constituency-based notion of locality. He then shows that an approach referred to as *argument attraction*, introduced into HPSG by Hinrichs and Nakazawa (1990)[3], successfully integrates this notion of locality with discontinuous constituency. Under an argument attraction analysis, the constituent structure assigned to (18a), repeated here as (42), is markedly different: the verbs first combine to form a verbal cluster that subcategorizes for a combination of each component's subcategorization requirements. The nature of this combination depends on the relationship between the verbs; in this example it is set union.

[3] As Pollard et al. (1994) point out, this approach is not original with HPSG; it echoes the notions of function composition in combinatory categorial grammar, clause union and ascension from relational grammar, and earlier work in GPSG by Nerbonne (1986).

(42)

daß	das Buch$_1$	dem Mann$_2$	die Frau$_3$	zu lesen$_1$	versprochen$_2$	hat$_3$
that	the book	the man	the woman	to read	promised	has

'… that the woman promised the man to read the book.'

This is illustrated in Figure 2.12. Here, *zu lesen* is an ordinary content verb subcategorizing for a subject and an object. As a subject control verb, *versprochen* subcategorizes for a subject, a dative object, and a verbal complement whose subject is coindexed with *versprochen*'s subject; here, the verbal complement is selected via the VCOMP feature (Chung 1993; Rentier 1994; Müller 1997). Thus the verbal complex *zu lesen versprochen* subcategorizes for three NP arguments. At the next level, *hat*, as a raising verb, simply promotes its verbal argument's subcategorization requirements to be its own without adding any extra arguments, and thus the highest V node is expecting three arguments, which it will then combine with. There is no consensus on whether the arguments combine all at once, or one at a time; Müller (2004) discusses the arguments for and against each analysis.

Now constructions exist in German in which matrix verbs require access to various properties of, respectively, the subject, the direct object, and the indirect object of embedded clauses. By treating the verbal complex (in the last example, *zu lesen versprochen hat*) as a consituent (which Reape's analysis does not do), it is possible to construct accurate analyses of these phenomena which preserve locality.

First consider German raising verbs, which agree with their subject when one is present and take third-person singular inflection otherwise. This is shown in (43) – (45).

(43)

Heute	scheint	gearbeitet	zu	werden.
today	seems.3SG	worked	to	be

'There seems to be work going on today.'

(44)

Den	Mädchen	scheint/*scheinen	schlecht	zu	werden
the	girls.DAT	seem.3SG/seem.3PL	ill	to	become

'The girls seem to be getting ill to the stomach.'

(45)

Du	scheinst/*scheint	nichts	zu	verstehen.
You	seem.2SG/seem.3SG	nothing	to	understand

'You don't seem to understand anything.'

All three sentences have the raising verb *scheinen* as their main verb. Sentences (43) and (44) are subjectless constructions, in which the 3SG form *scheint* is required. The embedded verb in (45), in contrast, has a subject, and so the verb must agree with this 2SG subject.

However in Reape's approach, where a raising verb combines with a full sentence, the information on the presence and nature of the embedded subject is no longer locally available: the embedded verb has combined with its arguments and now has empty valence features. This is illustrated in Figure 2.13, where the object representing *nichts zu verstehen* is a sentence: it has an empty SUBCAT list. Thus the raising verb *scheinen* cannot locally select for any properties of the embedded subject.

As a result, Reape would have to assume an alternative account of the selectional data. With argument attraction, however, raising verbs combine with their embedded verbs before the subject is realized, and they may therefore straightforwardly make use of the embedded verb's valence features.

This is not the only alternative, of course; one could assume that SUBJ is a head feature (see, e.g. Kiss 1995; Meurers 1999). But the same arguments apply in the case of matrix access to embedded direct and indirect objects, in which case percolation of SUBJ will not provide the necessary information. Again, one could argue that ARG-ST should also be

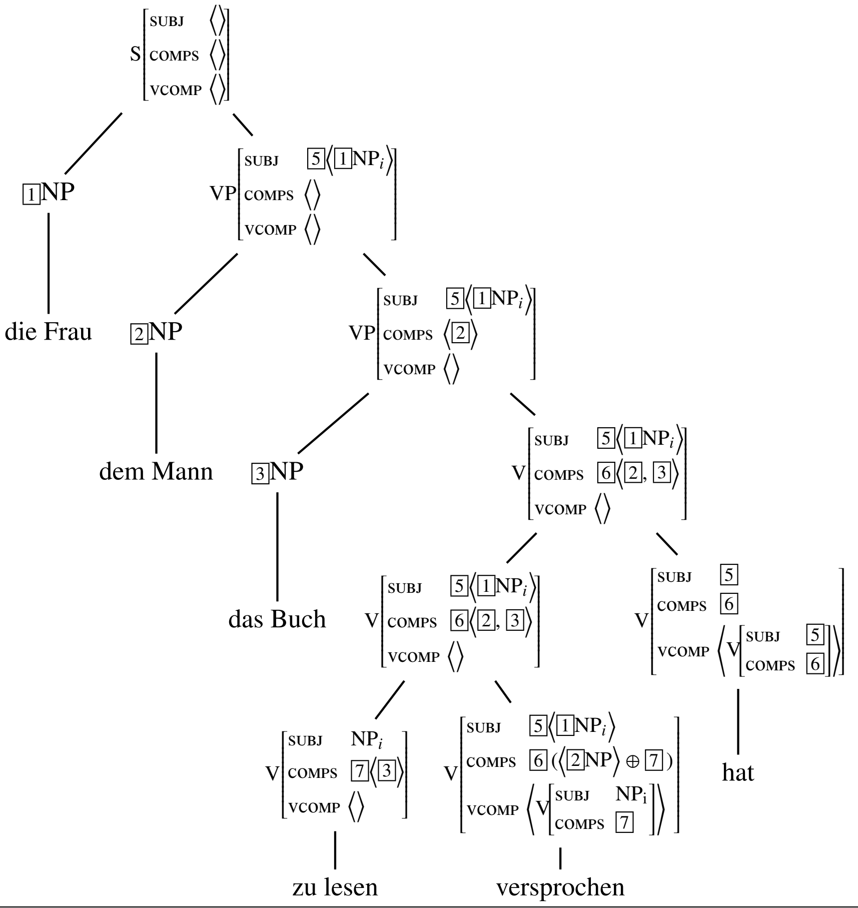


Figure 2.12: Argument Attraction

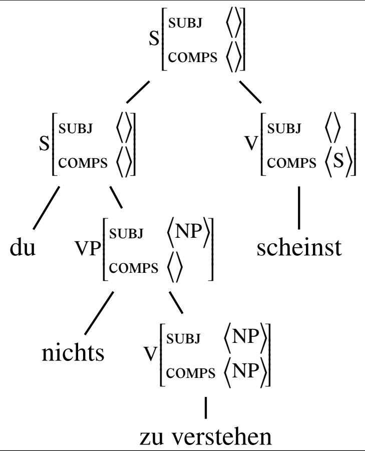


Figure 2.13: Invisibility of Subject Properties

percolated as a head feature (following, e.g. Przepiórkowski 2001), but this goes against virtually any notion of locality.

Kathol illustrates the need for matrix access to properties of embedded direct and indirect objects with the sentences in (46) – (48). Sentence (46) illustrates the phenomenon known as ‘distant’ or ‘long’ passive (Höhle 1978), in which the noun phrase *der Wagen* is case-marked as the grammatical subject of the passive predicate *wurde versucht* while thematically serving as the patient of the embedded predicate *zu reparieren*. Sentence (47) is an instance of the ‘recipient’ or ‘dative’ passive, in which the indirect object *Junge* is case-marked as the subject of *bekommt* while thematically serving as the recipient of the embedded predicate *schenkt* (in contrast to (48), where *Junge* is case-marked as an indirect object).

(46)	Der	Wagen	wurde	zu	reparieren	versucht.
	the	car.NOM	was	to	repair	tried

‘Someone tried to repair the car.’

(47)	Der	Junge	bekommt	(von	mir)	ein	Buch	geschenkt.
	the	boy.NOM	receives	by	me	a	book.ACC	presented

‘The boy is presented with a book (by me).’

(48)	Ich	schenke	dem	Jungen	ein	Buch.
	I	present	the	boy.DAT	a	book.ACC

‘I present the boy with a book.’

As with the subject data above, these phenomena cannot be accounted for if verbs like *versuchen* and *bekommen* combine with verb-phrases, as they do under Reape’s approach: the direct object of *reparieren* and the indirect object of *schenken* would not be visible at the VP level and thus unable to be affected by the passivization of *versuchen* or *schenken*. Under the argument attraction approach, in constrast, *der Wagen* becomes a direct object of the verbal complex *reparieren versuchen*, which is then passivized as a unit, assigning passive morphology to *versucht* and realizing *der Wagen* as a grammatical subject rather than as an object. Kathol concludes that Reape’s structural assumptions are unable to provide local analyses for these kinds of phenomena as well, while argument attraction can.

2.4 Conclusion

From the material presented in this chapter, it is clear that many linguists have found discontinuous constituency a necessary component of the analysis of language, and it is therefore worthwhile to have a parsing mechanism designed for linearization-HPSG grammars. Any such parsing mechanism should be designed to enable the linguist to develop grammars that model the underlying linguistic theory as closely as possible, while still allowing efficient processing. The next chapter will survey existing parsing mechanisms in the literature capable of handling discontinuous constituents, evaluating them with regard to these desiderata – in particular, the capability to independently specify ID rules and LP constraints and to explicitly specify the domains of LP constraints for both fixed and free word-order languages.

Chapter 3

Existing Formalisms for Parsing With Discontinuous Constituents

Any formalism intended to serve as a backbone for linearization-HPSG grammar processing needs to have at least two characteristics: (1) the capability to independently specify ID rules and LP constraints and (2) the ability to explicitly specify the domains of LP constraints smaller or larger than the local tree. A secondary consideration is that a parsing algorithm is available that uses these LP constraints and word order domains as early as possible in the parsing process, so as to avoid an inefficient "generate and test" pattern.

This chapter surveys the literature on formalisms for parsing with discontinuous consituents with regard to these considerations, ultimately concluding that no existing formalism satisfies these criteria. This chapter thereby establishes the need for the GIDLP formalism and parsing algorithm that will be presented in the remaining chapters.

3.1 Early Work

Johnson (1985), working with definite clause grammars (DCGs), provided some of the initial work on parsing discontinuous constituents. Where DCGs by default use concatenation to build phrases from their constituents, Johnson replaces this with a `combines` predicate that succeeds if two conditions are met: (1) the first argument (representing the left-hand category's coverage) is equal to the union of all other arguments (representing the right-hand category coverages); and (2) none of the right-hand category coverages overlap. For instance, Johnson gives the rule in (49) as part of his grammar for Guugu Yimidhirr.[1] This rule states that an S may consist of a V and two NPs, as long as their coverages respect the `combines` predicate.

(49) $\mathrm{S}(\ell) \leftarrow \mathrm{V}(\ell_1)$ & $\mathrm{NP}(\ell_2, \mathrm{Erg}, 0)$ & $\mathrm{NP}(\ell_3, \mathrm{Abs}, 0)$ & `combines`$(\ell, \ell_1, \ell_2, \ell_3)$

A similar approach is taken by van Noord (1991), who presents a modified version of a left-corner parser (anticipating his later head-corner parser). As with Johnson's parser, discontinuity is a function of the combinatory predicate attached to each rule. For example, the four combinatory schemes used in van Noord's grammar of Dutch are given in Figure 3.1; these schemes interact as shown in Figure 3.2 while parsing the sentence given in (50). In the latter diagram, each node is represented as the mode of combination that created the node (if any), following by the string represented by the node. The head of each node appears in small caps.

[1]The second and third arguments in the NP descriptions represent morphological information.

(50) Ziet Jan Piet Marie kussen?
sees Jan Piet Marie kiss
'Does Jan see Piet kiss Mary?'

Here, the modes `left` and `right` are leftward and rightward concatenation, respectively. The `vr` mode wraps the argument around the functor's head: here, the argument's left context *marie* is placed between the functor's left context *piet* and head *ziet*, and the argument's head and right context *kussen* are placed between the functor's head *ziet* and right context (here, empty). Finally, in the `v2` mode, the functor is empty; the mother is formed by inverting the argument's head *ziet* and left context *jan piet marie*.

Reape (1991a) generalizes these methods, presenting versions of several algorithms that have been adapted to allow for discontinuous constituents: top-down, left-corner, head-corner, shift-reduce, and CYK. The input format remains (roughly) the same for each: immediate dominance rules and a `lexicon` predicate whose clauses map words to categories.

All three of these approaches support free word order (in van Noord's approach, one would simply write a non-deterministic `cb` predicate). Yet there are no facilities for constraining word order outside the local tree. In addition, the combinatory predicates attached to the rules are not used to guide parsing – they only act as post-hoc filters. None therefore allow for direct modelling of linearization-HPSG grammars.

3.2 Dependency Grammar

The term dependency grammar refers to a wide array of research, generally considered to have been brought into the modern era by Tesnière (1959). Through all these strands runs a common theme: the concept of 'phrase' is deemed unnecessary; instead, linguistic structure is said to arise through the dependencies words have on another.

Per Robinson's (1970) formalization, dependency is an acyclic order on the lexical items in a sentence in which (1) there is only a single maximal element; and (2) all other elements depend on exactly one element. A dependency grammar is then specified as a set of dependency rules, each of the form 'category X may have category Y as a dependent'; a sample grammar is given in (51). (Some versions of dependency grammar also include valence requirements with the inverse force: 'category X must have category Y as a dependent'.)

(51) a) V ← N
b) N ← ADJ
c) N ← DET

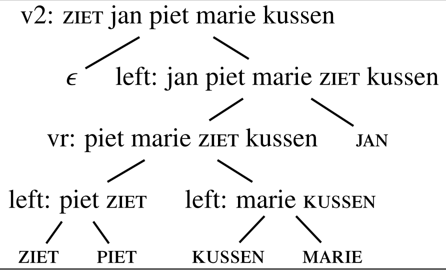
The difference between dependency grammar and phrase structure grammar is illustrated by the two trees in Figure 3.3. The tree on the left is the familiar phrase structure analysis, while the tree on the right gives the dependency analysis (here, solid lines represent dependency, while the dotted lines show the projection of each lexical item). The major contrast is that words (like *big* and *dog*) combine to form phrases (like *big dog*) in the phrase structure tree, while in the dependency grammar tree, words depend on each other (as *big* depends on *dog*).

A significant debate in the dependency grammar community centers around an additional axiom known as *projectivity*. This axiom effectively states that dependency lines may not cross projection lines in a dependency graph. As such, analyses like that in Figure 3.4 (Covington 1990) would be rejected.

When projectivity is adopted, dependency analyses can be stated with trees rather than graphs, and dependency grammars become weakly equivalent to context-free grammars (as

```
% cb(Identifier, FunctorDaughter, ArgDaughter, Mother)
% Categories are represented by p(Left, Head, Right)
% X-Y is interpreted as "from position X to position Y"
cb(left, p(L4-L, H, R), p(L1-L2, L2-L3, L3-L4), p(L1-L, H, R)).
cb(right, p(L, H, R1-R2), p(R2-R3, R3-R4, R4-R), p(L, H, R1-R)).
cb(vr, p(L1-L2, H, R3-R), p(L2-L, R1-R2, R2-R3), p(L1-L, H, R1-R)).
cb(v2, p(A-A, B-B, C-C), p(R1-R2, H, R2-R), p(A-A, H, R1-R)).
```

Figure 3.1: Three modes of combination



v2: ZIET jan piet marie kussen
ϵ
left: jan piet marie ZIET kussen
vr: piet marie ZIET kussen
JAN
left: piet ZIET
left: marie KUSSEN
ZIET
PIET
KUSSEN
MARIE

Figure 3.2: Modes of combination as used in parsing

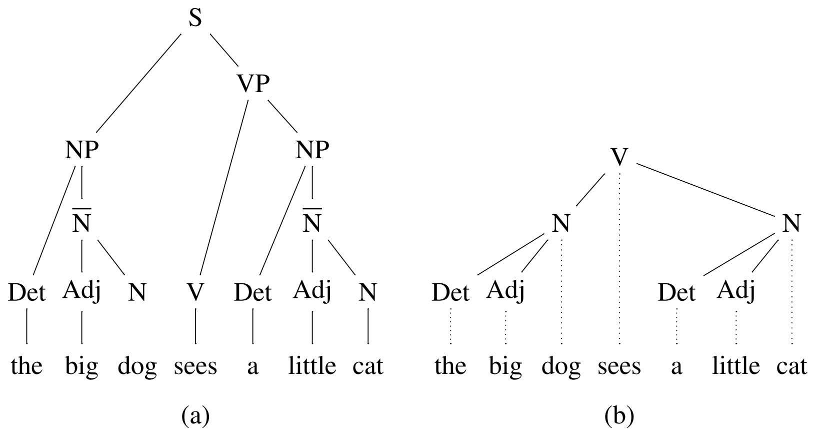


Figure 3.3: Phrase Structure and Dependency Grammar Trees

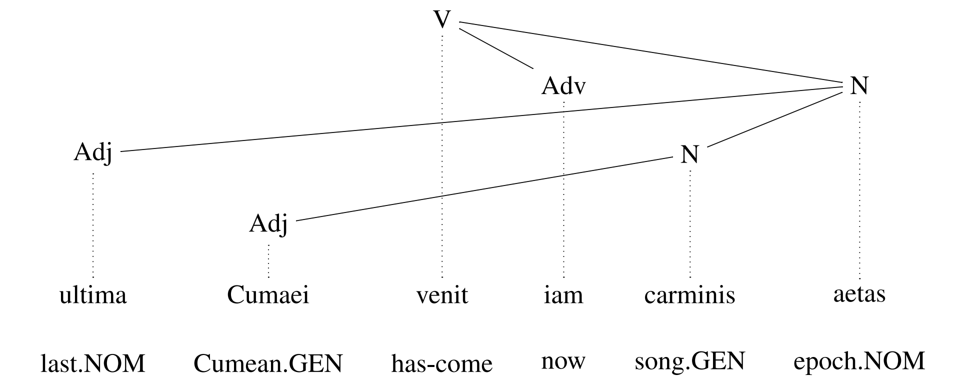


Figure 3.4: Non-Projective Dependency Tree

shown by Hays (1964)); as a result, the literature on parsing with context-free grammars is directly applicable to parsing projective dependency grammars.

But projectivity did not feature in Tesnière's (1959) original formulation of dependency grammar, and as a result, a body of work has arisen around non-projective dependency parsing. One of the earliest such parsers is described by Covington (1990, 1992), who accepts grammars similar to that shown in (51), with two modifications: categories are feature-structures, and each dependency rule is characterized as head-final or head-initial.

Covington informally presents his algorithm as follows: for each word, check for previously-seen words on which the current word could depend. If none are found, mark the word as a potential head; if one is found, record the dependency; and if multiple ones are found, record the dependency with the closest word, keeping track of the alternatives. Then search the list of potential heads for words that could depend on the current word. For each one found, establish the dependency and remove it from the list of potential heads. At the end of the parse, this list will only contain the head of the sentence.

Covington notes that because of the parser's preference to establish dependencies with the closest words, the parser prefers continuous constituents where possible, adding what he sees as a degree of psychological realism to the algorithm. He also describes a number of potential avenues for enhancement of this parser; for example, one could limit the depth to which the lists of previous words and potential heads were searched, implementing a maximum distance between head and dependent.

As a backbone for linearization-HPSG, however, the parser falls short – as Covington (1994) points out, an order constraint between two dependents of a given head (for example, that English indirect objects precede direct objects) cannot be directly stated and must be encoded on an ad hoc basis in the verb's lexical entry: by stating the verb's dependents as a list rather than as a set, the verb can be made to combine with one dependent before the other.

In a similar vein, Koch (1993) observes that Covington's formalism treats all dependencies as optional and thus has no way to encode the requirement that, for example, English verbs must be associated with a subject or that transitive verbs must be associated with an object. Koch addressses this problem, but only through a generate-and-test approach, adding constraints to the grammar on the numbers of each type of dependency per category; these constraints are then used to discard otherwise-successful parses.

3.3 DPSG

A different approach to discontinuous constituency is taken in a series of papers describing a framework called Discontinuous Phrase Structure Grammar (van der Sloot 1989; Bunt 1991; Vogel and Erjavec 1994; Bunt and van der Sloot 1996). As a variant of phrase structure grammar, its key feature is that discontinuity is modeled by phrase structure rules that explicitly label some constituents as 'internal context'. As an example, van der Sloot provides the grammar in (52), which generates the parse tree in Figure 3.5 for the sentence 'John will not come'.

(52) a) PN $\rightarrow$ NP
b) S $\rightarrow$ NP + V
c) S $\rightarrow$ S + ADV
d) V $\rightarrow$ AUX + [ADV] + V

Here the first three rules are standard context-free rules, while the last rule indicates that a V may consist of an AUX and a V, with ADV (the bracketed element) interrupting the continuity of the V. Here, ADV is referred to as the *internal context* of the rule, or as a *context daughter*.

Bunt also discusses a version of the formalism modelled after ID/LP rules. In this version, the ID/LP variant of (52b) is (53).

(53) V $\rightarrow$ AUX, V ; ADV ; AUX < ADV, ADV < V

Here, the context daughters appear in a separate field of the rule; as a result, one does not have to specify the order of the context daughters. Just as one traditional ID/LP rule can abbreviate several context-free rules, so can one DPSG ID/LP rule abbreviate several DPSG rules without changing the power of the formalism.

DPSG's advantage, as Vogel and Erjavec (1994) have shown, is that it is only mildly-context sensitive (yielding a complexity class no worse than $O(n^7)$) as long as tree branches are only allowed to cross to the left; this restriction does not limit the expressivity of the formalism.

The parser described by van der Sloot (1989) acts much like a normal chart parser; internal context daughters are predicted and completed just like regular daughters, so that the 'dot' can only move over a context daughter when such a category in the proper location has been licensedq. When a passive edge containing a context daughter is completed, however, the context daughter is re-scanned and allowed to complete with descendants of the edge. Each edge contains a construction list that marks, for each of its daughters, whether that daughter is a context daughter or not; as a result, the left index of any given edge's successor can be computed at any time. A node with a successor position corresponding to the end of the string is considered to indicate a successor parse; this ensures that all context daughters will eventually be incorporated as actual daughters at some level of the parse tree.

For example, in Figure 3.5, the categories AUX, ADV, and V from the fourth grammar rule will be predicted and completed in turn, forming a V node spanning positions 2 through 5, with successor position 3. This V node is then completed with the NP to form an S node spanning positions 1 through 5, with successor position 3. This is completed with the S rule, leading to a prediction of ADV (via rule c) at position 3. The context daughter is re-scanned at this point and completes the S, forming an edge spanning positions 1 through 5 with successor position 5. Since this S node has a successor position corresponding to the end of the string, it is considered to indicate a successful parse.

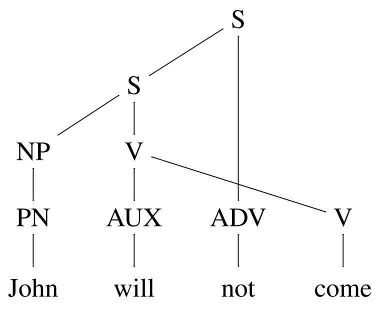


Figure 3.5: Discontinuous Phrase Structure Grammar Tree

Thus while DPSG can be efficiently processed, the requirement that internal contexts be explicitly stated makes it suitable only for languages with few discontinuities. For a language like Latin or Sanskrit with almost completely-free word order, an arbitrary sequence of elements can interrupt almost any other constituent – in effect, the internal context should be X^*, where X is a variable over categories. Since the notation X^* is not part of the formalism, the grammar effectively has to be written many times over. Initially, a set of categories is licensed. Then each of those categories, and each possible sequence of those categories, has to be licensed as a possible internal context of each other category.

3.4 Topological Parsing

Given the popularity in descriptive linguistics of the topological field model (presented in section 2.2.3), it seems natural that it would eventually be chosen as the basis for a parsing formalism. This is an active field of research, and many direct topological field parsing models exist. This section will take two such parsers as exemplars to support the claim that the topological field model is too restrictive to serve as a general formalism for linearization-HPSG. While it is necessary to have a formalism which can support a topological field model, it is overly confining to force **all** linearization-HPSG grammars to fit a topological field model.

3.4.1 Topological Dependency Parsing

Gerdes and Kahane (2001) present a formalism called Topological Dependency Grammar (TDG). Under this formalism, a parser receives a dependency tree as input and determines the corresponding topological hierarchy.

The formalism is quite complex and best presented in the form of an example. Consider first the sentence in (54), with the dependency tree given in Figure 3.6. Note that, in an extension to the version of dependency grammar described in section 3.2, Gerdes and Kahane assume that each dependency relation is labelled with a syntactic relation.

(54)

Zu	lesen	hat	diesem	Mann	das	Buch	niemand	versprochen.
to	read	has	this	man	the	book	nobody	promised

‘Nobody promised this man that they would read the book.’

The topological grammar that Gerdes and Kahane present consists of eight components. Note that for purposes of illustration, Gerdes and Kahane assume that noun phrases are atomic units; this presentation also omits any elements of Gerdes and Kahane’s grammar that are not used while parsing (54).

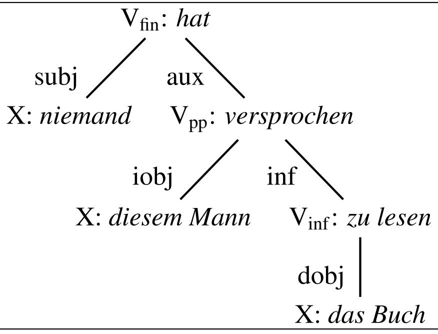


Figure 3.6: Dependency Tree

1. A set of syntactic categories.

   $\{V_{\text{fin}}, V_{\text{inf}}, V_{\text{pp}}, X\}$

   Here, Gerdes and Kahane use the category X to abstract over structures unimportant to their grammar; generally noun phrases or other arguments. The formalism additionally allows for abbreviations of disjunctions of categories: Y abbreviates "any category", and the formalism also allows for other abbreviations of the form V or $V_{\neg\text{fin}}$.
2. A set of lexical items, each associated with a category.

$X \rightarrow$ *das Buch*	$V_{\text{fin}} \rightarrow$ *hat*
$X \rightarrow$ *diesem Mann*	$V_{\text{pp}} \rightarrow$ *versprochen*
$X \rightarrow$ *niemand*	$V_{\text{inf}} \rightarrow$ *zu lesen*

3. A set of syntactic relations.

   {aux, dobj, inf, iobj, subj}

   R is an abbreviation for "any relation", akin to the category Y.
4. A set of fields, each with a corresponding descriptor.

   {i!, vf!, [!, mf*,]?, nf*, h!, o?, u?, −!, f}

   There are four field descriptors, each detailing how many boxes the field can contain: ! indicates exactly one, ? at most one, + at least one, and * zero or more. One of the fields (here, *i*) must be designated as initial. Field disjunction abbreviations are again allowed; here, f stands for any of vf, mf, or nf. The field − is special in that it may only be filled by a lexical item, rather than a box.
5. A set of boxes (word order domains), each with a corresponding description rule.

md → vf [mf] nf	vc → o h u
ed → mf] nf	v → −

   (Note that "[", "]", and "−" are actual field names, not punctuation.) Each box consists of a series of fields (defined below) that occur in exactly the order described.
6. The box permeability order, a ranking on boxes.

   vc < ed < md

   This order is interpreted in the context of the correspondence rules described in the next item.

7. Correspondence rules, which relate the topological structure to the dependency structure.

   C1) (R, Y, V,], –) C3) (R, V, Y, f, ed)[2]
   C2) (R, V, $V_{\neg\text{fin}}$, o, vc)

   These rules have the form (rel, c1, c2, f2, b): if a word w2 of category c2 has a dependency of type rel on a word w1 of category c1, then it may be targeted for a field f2 in any box containing w1, as long as that box is not separated from w1 by any borders ranked higher than b in the permeability order. As such, it is interesting to notice that a compacted domain in this approach is simply one in which no correspondence rule uses that domain or a domain that outranks it in permeability as a boundary.
8. Box creation rules.

   B1) (V_{fin}, i, md, [) B3) (V,]|u, vc, h)
   B2) ($V_{\neg\text{fin}}$, f, ed,]) B4) (V, [|h, v, –)

   These rules detail the hierarchical structure of the boxes. A rule of the form (c, f, b, f2) indicates that a word of category c targeted for field f instead creates a box b in which the word is now targeted for field f2. These rules generally apply recursively until a lexical box is created.

Note that the relationship between boxes and fields is mutually recursive: boxes contain fields, and fields contain boxes. Such a topological structure can therefore be depicted in the form of a tree in which alternate ranks contain fields (circular nodes) and boxes (rectangular nodes). The parse of (54) proceeds as follows:

Step 1 (Figure 3.7): the root node of the tree (here, a V_{inf}) is first targeted for the initial field i. Box creation rules then apply until a lexical box is created. In this case, only B1 applies to the pair (V_{inf}, i). An md box is created, and the V_{inf} is now targeted at the [field within that box. The rule that applies to this new pair (V_{inf}, [) is B4, ending this step.

Step 2 (Figure 3.8): The first dependency considered is that of V_{pp} on V_{fin}. (Since none of the correspondence rules in this example are conditioned on dependency type, the types can be ignored.) Rule C1 applies, and so the V_{pp} will be targeted for the] field in the md box containing the V_{fin} (since there is only one, permeability does not apply). The pair (V_{pp},]) matches B3, creating a vc box within which the h field is targeted. The pair (V_{pp}, h) matches B4, ending this step.

Step 3 (Figure 3.9): The next dependency is that of V_{inf} on V_{pp}. Rule C3 can be applied, targeting the V_{inf} for any major field. The V_{pp} is contained in two boxes: vc and md. Since the final component of rule C3 is ed, which outranks vc in permeability, the vf field of the md box can be chosen as a target. (If a less permeable field boundary had been specified, then only the other fields within the lower vc box could have been targeted.) Now the box creation rules apply. The pair (V_{inf}, vf) creates an ed box in accordance with B2; (V_{inf},]) creates a vc box by B3; and (V_{inf}, h) is lexical.

Step 4 (Figure 3.10): The final three dependencies are parsed in the same manner. All invoke correspondence rule C3, which allows the Xs to permeate their various lower domains and be placed in the mf. At this point there are no dependencies left, and each field's descriptor has been satisfied, so the parse concludes.

As can be seen, there are no LP constraints per se in this formalism – the relative order of words is determined solely by the ordering of fields within boxes. In other words, if

[2]Gerdes and Kahane erroneously give this rule as (R, V, X, f, ed); it is clear from their sample parse that the third element should be Y.

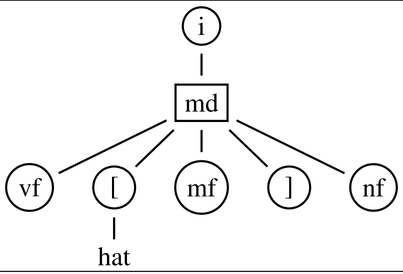


Figure 3.7: Topological Dependency Parse: Step 1

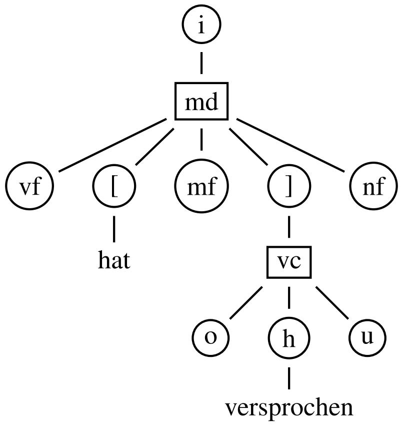


Figure 3.8: Topological Dependency Parse: Step 2

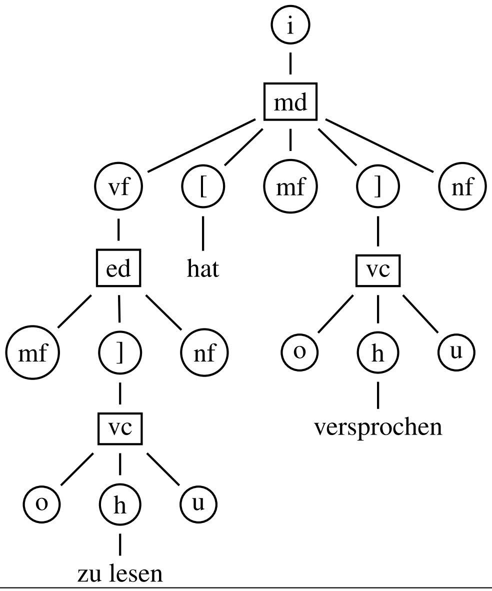


Figure 3.9: Topological Dependency Parse: Step 3

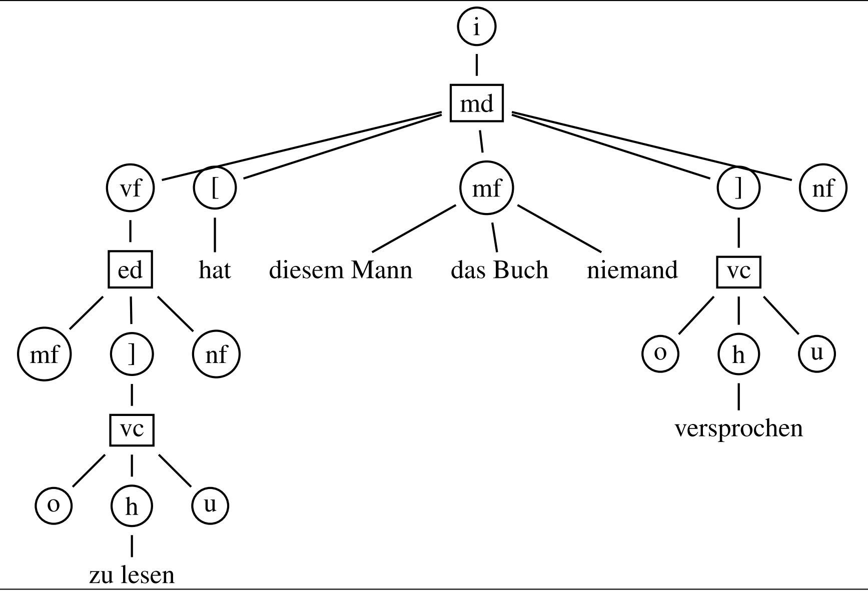


Figure 3.10: Topological Dependency Parse: Step 4

a grammar writer wanted to model some of the constraints on word order in the German middle field, they would have to add additional boxes and fields to the grammar that could be created within the middle field. In a fixed word-order language like English, a grammar writer would end up needing to create a near isomorphism between categories and fields, obscuring the intent of the grammar. Given this inflexibility, the formalism cannot serve as a backbone for naturally-described linearization-HPSG grammars.

3.4.2 Penn's Topological Parser

Penn and Haji-Abdolhosseini (2003) describe a parsing formalism superficially similar to that of the TDG parser. Unlike that parser, however, this parser simultaneously builds up tectogrammatic and phenogrammatic structure. Those aspects of the input language dealing with topological structure in Gerdes and Kahane's TDG parser are the same in Penn and Haji-Abdolhosseini's formalism: a set of fields with field descriptors and a set of boxes (now called regions) with the fields they contain. Penn and Haji-Abdolhosseini also assume a set of categories, lexical entries, and phrase structure rules (annotated with constraints as described below); the formalism as described can accomodate atomic categories or more complex categories.

In the TDG parser, the recursive relationships between fields and boxes were mediated by the box creation rules; given a category and the field it was targeted for, the existence of a new box within that field was licensed. Penn and Haji-Abdolhosseini's parser replaces these rules with a set of linking constraints that make no reference to tectogrammatic categories. Linking constraints can be of the form $r \twoheadleftarrow f$ or $f \twoheadrightarrow R$. The former have the interpretation "all regions r are found in a field f"; the latter, "all fields f contain one of the regions in R". Note that Penn and Haji-Abdolhosseini interpret the field descriptors in a slightly different way than Gerdes and Kahane. For Penn and Haji-Abdolhosseini, a field may contain at most one region, though a region may contain several instances of a field; whereas for Gerdes and Kahane, a field only appears once in a box but may contain several boxes. For example, in the topological structure produced by Gerdes and Kahane's parser (Figure 3.10), there is a single *mf* field with three boxes, one each for *diesem Mann*, *das*

Buch, and *niemand*. For Penn and Haji-Abdolhosseini, this would be described as a region which contains several *mf* fields – one for each of the three arguments.

Penn and Haji-Abdolhosseini also choose to constrain, rather than specify, the relationship between the tectogrammatic and phenogrammatic structure. There are three such types of constraints. The first all deal with correspondences between categories, fields, regions, and RHS elements of tectogrammatic rules (here, ϕ is a category, f is a field, r is a region, and i is a token):

ϕ `covers` f	i `covers` f	f `covered_by` ϕ	
ϕ `matches` f	i `matches` f	f `matched_by` ϕ	f `matched_by` i
ϕ `covers` r	i `covers` r	r `covered_by` ϕ	
ϕ `matches` r	i `matches` r	r `matched_by` ϕ	r `matched_by` i

As with the linking rules, the LHS of each constraint is universally quantified, and the RHS is existentially quantifed (thus the need for both x and x`ed_by` versions). One element is said to cover another if the phonological yield of the first is a superset of that of the second; they match if the two yields are equal. Constraints stated in terms of a token must be stated on individual rules; all other constraints have global effect.

In addition, both categories and rule tokens can be specified as compacted (that is, no discontinuities are present), and precedence constraints (both weak and immediate) can be imposed on rule tokens.

Penn and Haji-Abdolhosseini do not provide an example grammar, and it is unclear in general when a grammar contains enough constraints to actually parse with. The authors describe a parsing strategy that alternates between top-down tectogrammatic parsing and bottom-up phenogrammatic parsing. The parse starts by retrieving the categories for each lexical item. Matching/covering constraints are then used to get a set of, in effect, 'preterminal' fields. These fields are then parsed into regions, which may correspond to higher tectogrammatic categories, and so on.

Thus while superficially quite different from the TDG parser, this parser suffers from the same core problem – that word order constraints referring to elements in different (tecto) local trees can only be expressed through field ordering, preventing the LP constraints in a linearization-HPSG analysis from having any direct analogue – and is similarly unsuitable as a linearization-HPSG backbone.

3.5 Suhre's Linear Specification Language

The work most closely related to that presented in this thesis on parsing discontinuous constituents has been done by Suhre (1999). Noting that a traditional context-free rule can be seen as imposing a total order on its RHS, Suhre proposes a linear specification language (LSL) based on rules where the RHS is only partially ordered. Specifically, the RHS of an LSL rule consists of a directed acyclic graph whose vertices are categories and whose edges may either be weak (denoting weak precedence between the relevant categories) or strong (for immediate precedence); it must be possible to topologically sort the graph in order for the rule to be valid. Some nodes may also be marked as isolated, in the sense that their phonological yields will consist of an uninterrupted subsequence of the input string. It is also possible to mark the LHS of a rule as isolated. An example LSL grammar is given in (55).

(55) a) $A \rightarrow \textcircled{B} \quad \textcircled{C}$

b) $$\begin{tikzcd} D \arrow[r] & \textcircled{E} \arrow[r, two heads] & \textcircled{F} \end{tikzcd}$$

c) $G \rightarrow \textcircled{H} \rightarrow \textcircled{I}$

In this grammar, (55)a has no precedence constraints, (55)b requires E to immediately precede F, and (55)c requires H to weakly precede I. (A double circle around a category – not seen in this example – would indicate that that category forms a word order domain, while a double circle around the LHS would indicate that the local tree itself is a domain.)

Suhre's parser is a variant of Earley's algorithm (as will be described in section 5.1) and will be discussed in greater detail in chapter 5. As a model for linearization-based HPSG, Suhre's LSL falls short in two respects. First, there is no way to directly describe partial compaction phenomena (as described in section 2.2.2), as the formalism only allows individual RHS elements, or the entire RHS, to form domains.

Second, there is no notion of a domain-specific constraint: all constraints are attached to rules. As a result, word order constraints cannot be imposed on constituents outside their local tree. This latter drawback was recognized by Fouvry and Meurers (2000), who describe an implementation of Suhre's LSL grammars with the addition of global LP statements that are taken to hold within each word order domain.

3.6 Conclusion

This chapter has shown that no existing formalism in the literature allows both the capability to independently specify ID rules and LP constraints and the ability to explicitly specify the domains of LP constraints. As a result, a new formalism and parsing algorithm must be developed to efficiently work with linearization-HPSG grammars. The next chapter will define such a formalism.

CHAPTER 4

The GIDLP Formalism

As discussed in chapter 2, the central aspects of linearization-based HPSG grammars are the word order domains and the linear precedence (LP) constraints that work within them. Any formalism intended to serve as a backbone for such grammars must include avenues for defining domains (including those domains that result from the partial compaction of a local tree) as well as for stating LP constraints that can constrain the order of elements originating in different local trees just as easily as of those originating in the same tree. The survey in chapter 3 of existing frameworks for parsing with discontinuous constituents shows that no current grammar formalism provides these facilities; as a result, a new formalism must be defined.

This chapter[1] presents such a formalism: Generalized ID/LP (GIDLP) grammar. Section 4.1 contains a discussion of the formalism, presented in an informal notation; section 4.2 gives some example grammars in the formalism; and section 4.3 presents a model of the formalism as a rewriting system.

4.1 Generalizing ID/LP Grammar

4.1.1 Overview

Following the linear specification language developed by Suhre (1999), the GIDLP formalism is based on the syntax of ID/LP rules, augmented with a means for specifying the relationship between the mother and daughters' domain objects. As such, a GIDLP grammar consists of three main parts (along with a set of categories and terminals): a set of lexical entries, a set of grammar rules, and a start declaration.

GIDLP grammars also reference two additional types of statements: LP constraints and domain declarations. LP constraints can be expressed in three contexts: in individual rules (as *rule-level* constraints), in domain declarations (as *domain-level* constraints), and over the entire grammar (as *global* constraints). Rule-level LP constraints are distinguished from domain-level LP constraints in terms of their scope, as will be seen in section 4.1.5; in contrast, global LP constraints are syntactic sugar – they simply allow a constraint which applies to every domain in the grammar to be stated once. In contrast, global domain declarations differ in notation and interpretation from rule-level domain declarations, as will be described in section 4.1.6.

It should be noted that the GIDLP formalism takes syntactic categories to be the objects of manipulation both in phenogrammar and tectogrammar, in the sense of (Curry 1961).

[1]Descriptions of the GIDLP formalism have already been published (Daniels and Meurers 2002, 2004a,b); the formalism presented in this chapter is an extended version of those presentations.

Throughout this section, syntactic categories will be taken to be represented by feature structure descriptions.

4.1.2 Lexical Entries

Lexical entries have the form $A \rightarrow t$, stating that the terminal t may be an instance of category A. For consistency, each lexical entry is taken to correspond to its own domain object.

4.1.3 Grammar Rules

Grammar rules are effectively annotated ID rules; they have the form $A \rightarrow \alpha; C; D$, where α is a set of categories (the RHS of the rule), C is a set of linear precedence constraints, and D is a set of domain declarations.

If the sets of LP constraints and domain declarations are empty, we obtain the simplest type of rule, exemplified in (56).

(56) S → NP, VP

This rule says that an S may immediately dominate an NP and a VP, with no constraints on the relative ordering of NP and VP. One may precede the other, the strings they cover may be interleaved, and material dominated by a node dominating S can equally be interleaved.

When linear precedence constraints or domain declarations are present, they will impose further limitations on the relative ordering of the daughters, as described in sections 4.1.5 and 4.1.6 below.

4.1.4 Start Declaration

The start declaration has the form $start(S) : L$, where S is a category and L a set of LP constraints. Its main role is to state that S is the *start symbol* of the grammar (thus playing the same role as the start symbol in a context free grammar). In addition, since the entire input string is implicitly an order domain, any constraints unique to that domain must be stated here.

4.1.5 LP Constraints

Intuitively, LP constraints enforce the idea that any instance of the LHS of the constraint must precede any other instance of the RHS within the same context (either the RHS of a grammar rule, or a word order domain). Specifically, consider all pairs of elements in the context such that the phonological realization of the first completely precedes that of the second. If any of these pairs jointly satisfies the pair description $\langle B, A \rangle$, the constraint is violated.[2] It is important to base this definition on satisfiability rather than unifiability – a constraint cannot be said to apply until the categories involved are at least as specific at the constraint's descriptions; see section 6.2 for further discussion.

LP constraints may optionally require that there be no intervening material between the two elements: this is referred to as immediate precedence. This operation in effect allows for constituents to be bound together without affecting the word order domains.

LP constraints are notated as follows:

[2]This definition is due to (Kasper et al. 1995) and is intended to deal with cases where the nature of the match between the first element and A will influence whether or not the second element matches B, and vice versa. This point is further discussed in section 6.2.

- **Weak precedence:** A < B.
- **Immediate precedence:** A $\ll$ B.

The symbols *A* and *B* may be categories, or pointers to specific elements of a rule's right-hand side (which are referred to as *tokens*). Constraints involving categories apply within specified domains (either the domain on which a constraint is specified, or all domains, in the case of global constraints). Constraints involving tokens, on the other hand, only have meaning with respect to a specific rule (the tokens are invisible outside that rule) and can only occur on individual rules. In this thesis, tokens are represented by numbers referring to the subscripted indices on the RHS categories.

Note that one consequence of the pair-based definition given above is that a constraint of the form A < A is satisfiable and has the effect of allowing at most a single instance of the category named in the relevant word order domain. The constraint A $\ll$ A has the same meaning.

An example of a rule-level LP constraint is given as part of (57).

(57) $\mathrm{A} \rightarrow \mathrm{NP}_1, \mathrm{V}_2, \mathrm{NP}_3; 3 < \mathrm{V}$

This constraint specifies that the token 3 in the rule's RHS (the second **NP**) must precede any constituents described as **V** occurring in the same domain.

One can view the RHS of a context-free rule as a description of a totally-ordered set, just as the RHS of an ID rule is a description of an unordered set. In this light, the effect of those rule-level constraints involving tokens is to partially order the RHS of the GIDLP rule. Category descriptions, on the other hand, are needed in order to have constraints that operate on categories that, while part of the same order domain, were introduced in different rules. Taken together with the option of specifying different constraints in different domains (see section 4.1.6), these two ways of referring to categories allow the grammar writer to develop grammars in which different environments require different relative orderings between a common set of categories.

It should also be noted at this point that by default, all properties of a category are assumed to be LP-accessible, as was the case in Reape's work, where domain objects were lists of signs. Other authors, like Kathol (2000), have argued that domain objects should store only a certain set of properties. Section 4.1.6 will describe functionality for accomplishing this.

4.1.6 Domain Declarations

As the GIDLP framework is intended to serve as a backbone for linearization-HPSG grammars, the analogues to Reape's domain objects are not stated explicitly. Instead, domain declarations are used to describe the desired domain objects.

A rule-level domain declaration has the form $\langle\{\alpha\}, A, L\rangle$, where α is a list of tokens, *A* is a category, and *L* is a set of LP constraints. Such a statement specifies that the constituents referenced in α form a word order domain with category *A*, inside of which the constraints in *L* hold.

Recall that partial compaction (see section 2.2.2) creates domain objects that are not constituents; to accomodate this, the second component *A* in a compaction statement needs to represent the linearization properties of the resulting domain object. In the case of total compaction, the linearization properties of the domain object are generally but need not be identical to those of the LHS category of the containing rule. In particular, in a system where only certain properties of a category are LP-accessible, *A* will represent the result of removing all other properties from the LHS of the containing rule. For example, a system where non-local properties are not LP-accessible might include a rule like that in (58).

(58) $\begin{bmatrix} a & \\ \text{LOCAL} & \boxed{1} \\ \text{NON-LOCAL} & \boxed{2} \end{bmatrix} \rightarrow B_1\ C_2;; \langle\{1, 2\}, \begin{bmatrix} a & \\ \text{LOCAL} & \boxed{1} \end{bmatrix}, []\rangle$

When this domain object is subsequently evaluated with respect to LP constraints in higher phrases, the non-local properties will not be able to influence the results. (One could get the same effect, of course, by simply not writing LP constraints that reference non-local properties. But this system makes the grammar writer's intent clear.)

The domain declarations on a rule determine the relationship between the mother's word order domain and the daughters' word order domains. If a given daughter is not referenced in a domain declaration, then it is unioned into its mother's word order domain; if it is the sole category in a domain declaration, it will be inserted into its mother's word order domain as a compacted domain; and if it occurs along with other categories in a domain declaration (as in (58)), these categories, taken together, will be inserted into their mother's word order domain as a single compacted domain.

For instance, in (59), each of the daughter S categories forms its own domain; they are domain-inserted into the word order domain of the mother S. In (60), the righthand side of the rule is partially compacted: the V and the first NP form a domain with category VP that does not include the second NP.

(59) $S \rightarrow S_1, \text{Conj}_2, S_3;\ 1 \ll 2, 2 \ll 3, \langle\{1\}, S, []\rangle, \langle\{3\}, S, []\rangle$

(60) $VP \rightarrow V_1, NP_2, NP_3;\ \langle\{1, 2\}, VP, []\rangle$

One will often form a domain from only a single category without adding domain-specific LP constraints, so an abbreviatory notation is introduced of writing such a compacted category in square brackets. In this way (59) can be written as (61).

(61) $S \rightarrow [S_1], \text{Conj}_2, [S_3];\ 1 \ll 2, 2 \ll 3$

A final abbreviatory device is useful when the entire RHS of a rule forms a single domain, which Suhre (1999) refers to as "left isolation". This is denoted by using the token 0 in the compaction statement if linear precedence constraints are attached, or by enclosing the LHS category in square brackets, otherwise. Thus (62), (63), and (64) are equivalent:

(62) $S \rightarrow V_1, \text{Nom}_2, \text{Acc}_3;\ 2 \ll 1, 1 \ll 3\ ;\ \langle\{1,2,3\}, S, []\rangle$

(63) $S \rightarrow V_1, \text{Nom}_2, \text{Acc}_3;\ 2 \ll 1, 1 \ll 3\ ;\ \langle\{0\}, S, []\rangle$

(64) $[S] \rightarrow V_1, \text{Nom}_2, \text{Acc}_3;\ 2 \ll 1, 1 \ll 3$

The formalism also supports *global compaction statements*. A global compaction statement has the form $\langle A, L\rangle$, where A is a category, and L is a list of domain-level precedence constraints. The interpretation is that whenever the designated category appears on the RHS of a rule, it always forms a compacted domain in which the given constraints hold. As an example, one could model Blevins's observation (see section 2.1.3) that only maximal projections form word order domains with the constraint in (65).

(65) $\langle\begin{bmatrix}\text{SUBCAT} & \textit{elist}\end{bmatrix}, []\rangle$

4.2 Examples

4.2.1 Mimicking a CFG

We start with an example illustrating how a CFG rule is encoded in GIDLP format. A CFG rule encodes the fact that each element of the RHS immediately precedes the next, and that the mother category dominates a contiguous string. The context-free rule in (66) is therefore equivalent to the GIDLP rule shown in (67).

(66) $S \rightarrow$ Nom V Acc

(67) $[S] \rightarrow V_1$, Nom$_2$, Acc$_3$; $2 \ll 1, 1 \ll 3$

4.2.2 Illustrating Domain Formation

In (68) we see a more interesting example of a GIDLP grammar. (To make the grammar as compact as possible while still illustrating the same features, artificial categories have been used.)

(68)
a) start(A) : []
b) $A \rightarrow B_1, C_2, [D_3]$; $2 < 3$
c) $B \rightarrow F_1, G_2, E_3$
d) $C \rightarrow E_1, D_2, I_3$; $\langle\{1,2\}, H, []\rangle$
e) $D \rightarrow J_1, K_2$
f) Lexical entries: $E \rightarrow$ *e*, ...
g) $E < F$

(68a) is the start declaration, stating that an input string must parse as an A; the empty list shows that no LP constraints are specifically declared for the domain. (68b) is a grammar rule stating that an A may immediately dominate a B, a C, and a D; it further states that the second constituent must precede the third (more precisely, following the definition in section 4.1.5, it must not be the case that the third constituent completely precedes the second constituent) and that the third is a compacted domain. (68c) gives a rule for B: it dominates an F, a G, and an E, in no particular order. (68d) is the rule for C, illustrating partial compaction: its first two constituents jointly form a compacted domain, which is assigned category H. (68e) gives the rule for D and (68f) specifies the lexical entries (here, the preterminals just rewrite to the respective lowercase terminal). Finally, (68g) introduces a global LP constraint requiring an E to precede an F whenever both elements occur in the same domain.

Now consider licensing the string *efjekgikj* with the above grammar. The parse tree, recording which rules are applied, is shown in figure 4.1. Given that the domains in which word order is determined can be larger than the local trees, we see crossing branches where discontinuous constituents are licensed.

To obtain a representation in which the order domains are represented as local trees, we can draw a tree with the compacted domains forming the nodes, as shown in figure 4.2. This representation highlights the fact that not all nodes in the tectogrammatic structure correspond to nodes in the phenogrammatic structure. The B node, for instance, does not correspond to a domain, but instead to a set of domains {E, F}.

There are three non-lexical compacted domains in the tree in figure 4.1: the root A, the compacted D, and the partial compaction of D and E forming the domain H within C. In each domain, the global LP constraint $E < F$ must be obeyed. Note that the string is licensed by this grammar even though the second occurrence of E does not precede the F. This E is inside a compacted domain and therefore is not in the same domain as the F, so that the LP constraint does not apply to those two elements. This illustrates the property of LP locality: domain compaction acts as a 'barrier' to LP application.

The second aspect of domain compaction, contiguity, is also illustrated by the example in connection with the difference between total and partial compaction. The compaction of D specified in (68b) requires that the material it dominates be a contiguous segment of

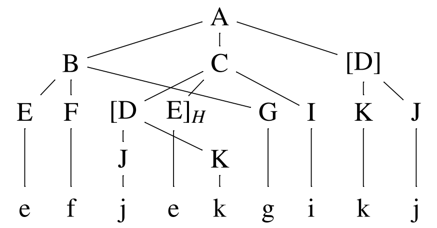


Figure 4.1: Parse Tree Showing Tectogrammatic Structure

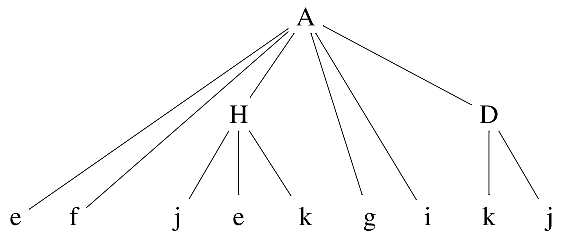


Figure 4.2: Parse Tree Showing Phenogrammatic Structure

the input. In contrast, the partial compaction of the first two RHS categories in rule (68d) requires that the material dominated by D and E, taken together, be a continuous segment. This allows the E to occur between the two categories dominated by D.

4.3 Formal Model

What has been presented so far is an informal description of the GIDLP grammar formalism, complete with convenience notations and syntactic sugar. The following formal model of a GIDLP grammar assumes that none of these convenience notations are being used: specifically, all compaction statements are given in full, and global LP constraints and domain declarations are not expressed as such, but rather appear on each relevant domain and rule, respectively.

The formal model of a GIDLP grammar arises from the intuition behind the formal definition of a context-free grammar:

> A context-free grammar G is a quadruple (V, Σ, R, S), where V is an alphabet, Σ (the set of terminals) is a subset of V, R (the set of rules) is a finite subset of $(V - \Sigma) \times V^*$, and S (the start symbol) is an element of $V - \Sigma$. The members of $V - \Sigma$ are called nonterminals. For any $A \in V - \Sigma$ and $u \in V^*$, we write $A \rightarrow_G u$ whenever $(A, u) \in R$. For any strings $u, v \in V^*$, we write $u \Rightarrow_G v$ if and only if there are strings $x, y \in V^*$ and $A \in V - \Sigma$ such that $u = xAy$, $v = xv'y$, and $A \rightarrow_G v'$. The relation $\Rightarrow_G^*$ is the reflexive, transitive closure of $\Rightarrow_G$. Finally, $L(G)$, the language generated by G, is $\{w \in \Sigma^* : S \Rightarrow_G^* w\}$; we also say that G generates each string in $L(G)$. (Lewis and Papadimitriou 1998)

In particular, just as a context-free derivation is expressed as a series of strings over the corresponding alphabet, a GIDLP derivation will consist of a series of domain objects, each of which is rewritten into the next.

A GIDLP grammar and its components can now be formally defined. Note that the following definitions are presented in a top-down fashion; each will make use of subsequent definitions.

A *GIDLP grammar* is a quintuple $\langle T, N, R, L, G\rangle$ where T is a set of lexical items (terminals), N is a set of categories (nonterminals), R is a domain object (the start symbol), L a relation from $T \to N$ (the lexicon), and G a set of grammar rules (the grammar).

A *domain object* is a triple $\langle A, \alpha, C\rangle$ where $A \in N$ is the result category, α is a string of categories and/or domain objects, and C is a set of LP constraints. A domain object is the equivalent of a domain declaration in the informal description.

A *grammar rule* is a triple $\langle A, \alpha, C\rangle$ where $A \in N$ is a category (the LHS of the rule), α is a string of category-token pairs $\langle a, b\rangle$, where $a \in N$ and $b \in \mathbb{N}$, and/or domain objects (the RHS of the rule), and C is a set of LP constraints. For all distinct $\langle a, b\rangle, \langle c, d\rangle \in \alpha$, it must be the case that $b \neq d$ (in other words, tokens may not be duplicated within a rule's RHS). Here, the main difference from the informal presentation is that there is no set of compaction statements on the rule; instead, domain objects are allowed to intermingle with categories on the rule's RHS.

An *LP constraint* is a triple $\langle a, b, t\rangle$ where $a, b \in (N \cup \mathbb{N})$ and $t \in \{w, i\}$ (representing weak and immediate precedence, respectively). Such an LP constraint is *satisfied* by a domain object when, for all pairs of distinct domain elements x, y such that x precedes y, x, y does not match the pair description b, a. In addition, if $t = i$, then for all pairs of distinct domain elements x, y such that x does not immediately precede y, it must be the case that x, y does not match the pair description a, b.

Now let A be the domain object $\langle a, \alpha A\beta, C_d\rangle$ and A' the domain object $A' = \langle a, \gamma, C_d\rangle$. If there is a rule $r \in G$ such that $r = \langle A, \delta, C_r\rangle$ and γ is a permutation of $\alpha \cdot \delta \cdot \beta$ such that, for all $c \in (C_d \cup C_r)$, γ satisfies c, then we say that $A \Rightarrow A'$ (read *A derives* A' *in one step*). In effect, γ represents a valid insertion of δ into $\alpha\beta$.

The transitive closure of $\Rightarrow$ is denoted $\Rightarrow^*$; when $A \Rightarrow^* A'$, we say *A derives* A', and A' is derived from A. Finally, let a *preterminal string s* of a terminal string t with length n is a string of length n such that for all n, $\langle t_n, s_n\rangle \in L$. Then a string of terminals is *recognized* by a grammar if there exists a corresponding preterminal string that can be derived from the start symbol of the grammar.

As an example, the grammar in (68) is formally described in figure 4.3 (for clarity, rule RHSs are given in terms of categories only instead of category-token pairs); the derivation of the string *efjekgikj* is given in figure 4.4.

Note that, aside from the fact that the compaction statements appear 'inside' the rules, (68) and figure 4.3 only differ in the absence of global order statements; as will be discussed in section 5.2.2.1, these are merely an abbreviatory device for grammar writers.

4.4 Conclusion

The goal of this section has been to present a formalism in which the grammar writer may define domains (including those domains that result from the partial compaction of a local tree) and state LP constraints that can constrain the order of elements originating in different local trees just as easily as of those originating in the same tree.

With the formalism presented in full, attention can now turn to the parsing algorithm, which will be presented in the next chapter.

$$
\begin{aligned}
T &= \{e, f, g, i, j, k\}\\
N &= \{A, B, C, D, E, F, G, H, I, J, K\}\\
R &= \langle A, [A], \{\langle E, F, w\rangle\}\rangle\\
L &= \{\langle e, E\rangle, \langle f, F\rangle, \ldots\}\\
G &= \{\langle A, [B, C, \langle D, [D], \{\langle E, F, w\rangle\}\rangle], \{\langle 2, 3, w\rangle\}\rangle,\\
&\qquad \langle B, [F, G, E], \emptyset\rangle,\\
&\qquad \langle C, [\langle H, [D, E], \{\langle E, F, w\rangle\}\rangle, I], \emptyset\rangle,\\
&\qquad \langle D, [J, K], \emptyset\rangle\}
\end{aligned}
$$

Figure 4.3: Formal Presentation of a GIDLP Grammar

$$
\begin{aligned}
&\langle A, [A]\rangle\\
&\Rightarrow \langle A, [B, C, \langle D, [D]\rangle]\rangle\\
&\Rightarrow \langle A, [B, C, \langle D, [D]\rangle]\rangle\\
&\Rightarrow \langle A, [E, F, G, C, \langle D, [D]\rangle\rangle\\
&\Rightarrow \langle A, [E, F, \langle H, [D, E]\rangle, G, I, \langle D, [D]\rangle\rangle\\
&\Rightarrow \langle A, [E, F, \langle H, [J, E, K]\rangle, G, I, \langle D, [D]\rangle\rangle\\
&\Rightarrow \langle A, [E, F, \langle H, [J, E, K]\rangle, G, I, \langle D, [K, J]\rangle\rangle
\end{aligned}
$$

Figure 4.4: Formal Derivation with respect to a GIDLP Grammar

CHAPTER 5

Parsing GIDLP Grammars With Atomic Categories

This chapter[1] presents an efficient algorithm for parsing grammars written in the GIDLP formalism described in chapter 4. The chapter begins in section 5.1 by describing the inspiration for the GIDLP parser. Section 5.2.1 describes the central concepts involved in creating a GIDLP parser from a context-free parser, and then section 5.2 presents the actual algorithm for GIDLP parsing over atomic categories. Section 5.3 concludes with some long-form examples of the parsing process.

5.1 Earley's Algorithm

Just as the GIDLP grammar format was developed as an extension of ID/LP grammar, we turn to Earley's algorithm (Earley 1970) – a memoizing algorithm for context-free parsing – as a starting point for the GIDLP parser. Note that the notion of a chart parser, introduced in (Kay 1980), postdates Earley's algorithm, as does the now-standard terminology used in discussions of chart parsing. This chapter will refer to a *chart* as a set of *edges*, rather than Earley's original terminology – a set of *statesets*, each of which is a set of *states*. In addition, as will be discussed in section 5.1.1, this presentation will include several commonly-made changes to the organization of the algorithm that will bring it closer in form to the GIDLP parsing algorithm, thereby making the relationship between the two algorithms clearer.

As with any parsing algorithm, Earley's algorithm takes a grammar and a string of terminals as input and determines whether the grammar in question licenses the input string; in other words, whether a series of context-free rewriting steps could transform the start symbol of the grammar into the input string, and then output a representation of that derivation.

The operation of the algorithm, given in figure 5.1, centers around two operations. In the first, hypotheses are formed describing potential expansions of a current target category; this process is referred to as *prediction*. The second combines information describing an already-located subconstituent into a prediction, producing either a more specific hypothesis or a well-formed substring; this is referred to as *completion*.

In chart parsing terms, the hypotheses are called *active edges*; they describe the potential for a category to be found covering a span that includes a given location and list the additional evidence that will be needed to confirm this hypothesis. Thus the prediction step creates one active edge based on another; the latter is said to be *responsible* for the former.

[1]Portions of the material in this chapter were originally presented in Daniels and Meurers (2002, 2004a); this chapter reflects a revised and updated version of that material.

```
PROCEDURE earley:
  INITIALIZE the chart with lexical edges
  ENQUEUE prediction of the start category
  WHILE the action queue is non-empty
    DEQUEUE the top action and perform it
  EXAMINE the chart for successful parses

PROCEDURE predict($_i[A \to \alpha \bullet_j \ B\beta]$):
  FOREACH grammar rule B $\to \gamma$:
    ADD an edge $_j[B \to \ \bullet_j \ \gamma]$ to the chart
    ENQUEUE prediction and completion with this edge

PROCEDURE complete($_i[A \to \alpha \bullet_j \ \beta]$):
  IF $\beta$ is empty:
    FOREACH active edge $_k[B \to \gamma \bullet_i \ A\delta]$
      ADD an edge $_k[B \to \gamma A \bullet_j \ \delta]$ to the chart
      ENQUEUE prediction and completion with this edge
  ELSE let $\beta = E\epsilon$:
    FOREACH passive edge $_j[E \to \gamma \bullet_k \ ]$
      ADD an edge $_i[A \to \alpha E \bullet_j \ \epsilon]$ to the chart
      ENQUEUE prediction and completion with this edge
```

Figure 5.1: Earley's Algorithm

Since the chart is a set of edges, any operation that attempts to add a redundant edge to the chart will have no effect; in such cases, the redundant edge is said to have been *blocked* from the chart.

The RHS of an active edge will grow smaller with each completion step, and once completion yields an edge with an empty RHS (a *final completion* of that edge), the edge is referred to as a *passive edge*. A passive edge encodes the information that a category has definitely been found at a certain location in the input; the passive edge is said to *provide* that category. Not all passive edges arise from completions, however; the term *lexical edges* is used to refer to those passive edges that map lexical items in the input string to preterminal categories. Completion can therefore be seen as creating a string of active edges, starting with an edge formed by prediction (the *ancestor* of the edges in the string), linked by other passive edges, and possibly terminating with a passive edge.

Formally, edges have the form $_i[A \to \alpha \bullet_j \ \beta]$, indicating a parse state in which the string from positions i to j matches α and an A will have been found if β is found immediately following j. In an active edge, j is known as the *active position*, and the initial category B of β is the *active element* of the edge; in a passive edge, β will be empty.

Early work on parsing (Younger 1967; Kasami and Torii 1969) made use of the notion of a well-formed substring table: a data structure containing the passive edges that had been found to date. Earley's algorithm extends this concept by storing both passive and active edges in a data structure known as an *active chart*.

As explained in figure 5.1, the parse begins by predicting the start symbol of the grammar. Each time an edge is added to the chart, the parser responds by triggering further prediction and completion steps. Once the chart has finished responding to the initial prediction and its consequences, every passive edge in the chart spanning the entire input string that provides the start symbol indicates a successful recognition.

As a result, the edges in the chart at the end of the parse can be arranged into a tree, with each edge having the active edge that created it as its parent. Each leaf in this tree represents an edge for which prediction and completion yielded no new edges. Thus all optimizations will take two forms: reducing the amount of effort needed to compute a newly-predicted or completed edge, and finding ways to make the search tree smaller. If an edge can be identified for which no children will ever contribute to a successful parse, preventing that edge from being added to the chart will prune that entire branch of the tree, leading to potentially considerable savings.

5.1.1 Modifying Earley's Algorithm

In the years since it was initially published, researchers have put substantial work into the analysis and implementation of Earley's algorithm and potential improvements. The version of Earley's algorithm presented in the previous section differs from Earley's original presentation by the addition of lexical seeding and active completion.

In Earley's original algorithm, a third operation – *scanning* – is used to insert lexical edges into the chart when the active category is a terminal. The parser described here seeds the chart with all of the lexical edges before the parse begins (so that only those pre-terminal rules are predicted whose lexical items appear in the input string). This is done to strengthen the bottom-up component, which is important considering the overall goal of parsing linearization-based HPSG grammars, where much of the information guiding parsing originates in the lexicon.

Eliminating scanning also requires modifying the triggering of prediction and completion steps, however. In Earley's algorithm, only a single step is triggered each time an edge is added to the chart: if the active category is a non-terminal, prediction triggers; if a terminal, scanning triggers; and if empty, completion triggers. But when lexical seeding is performed, and pre-terminal rules are no longer predicted, completion must occur whenever an edge is added to the chart, regardless of whether the entered edge is active or passive.

Yet even if lexical seeding were to be abandoned, this change would still need to be present in the GIDLP parsing algorithm. Because context-free parsing proceeds from left to right, Earley can assume that all possible active edges that will need a given category will have already been created when the passive edge is added to the chart. GIDLP parsing cannot proceed from left to right, however, since the grammar writer is able to specify the order in which categories are to be located (see section 5.2.1.2).

5.2 The GIDLP Parsing Algorithm – Atomic Categories

5.2.1 Design Considerations

It is useful to discuss several concepts that manifest themselves throughout the parsing algorithm before the actual steps of the algorithm are presented and discussed.

5.2.1.1 Edge Coverage

The single interval formed by i and j stored on each edge in Earley's algorithm is not sufficient to model edge coverage in a grammar that licenses discontinuous constituents, as each edge may potentially be covered by a discontinuous subset of the string. Johnson (1985) showed that this issue can be addressed by generalizing from single intervals to *lists of intervals*: for example, [1-3, 5] represents an edge that covers the first, second, third, and fifth words of the input.

As Johnson pointed out, such lists of intervals are conceptually equivalent to *bitvectors* – vectors of bits, each of which directly corresponds to a string position. When the bit is active, its position is considered to be included in the coverage of the vector; when the bit is inactive, it is not. Reape (1991a) uses prolog lists to encode bitvectors, so that each position may be either `true`, `false`, or a prolog variable. As is known from other applications, bitvectors can also be encoded as integers by representing them as binary numbers,[2] with the least-significant bit of the number corresponding to the leftmost word in the input string. A position set to 1 is referred to as *active* (versus *inactive* for 0 bits). (Note that this representation can be confusing at first when bitvectors are presented as binary numbers, since the leftmost word in the string is represented by the rightmost digit in the number.)

The merits of each style of representation can best be determined by considering a set of relevant bitvector operations necessary for GIDLP parsing; these are given in figure 5.2. Here, x and y are bitvectors, p is a 0-based position index, and AND, OR, XOR, and NOT are the ordinary bitwise operators. In the examples, the bitvectors are assumed to be five digits long.

The primary advantage of lists of intervals is constant-time checking for isolation: if the cardinality of the list of intervals is one, the edge is isolated. On the other hand, all of the other operations needed to retrieve and use edges in parsing, such as OVERLAP and COMBINE, are linear in the length of the lists of intervals (which, in the worst case, only differs from the length of the input string by a constant multiple). Reape cites the flexibility of the list-based representation – the possibility to mark a position as unknown by using an anonymous variable – as an advantage of that method of representation, but the list representation incurs the same efficiency penalty as the lists of intervals representation.

With an integer representation, in constrast, all necessary bitvector operations can be computed as numeric expressions that require time proportional to a logarithm of the length of the input string (where the base of the logarithm is the word size of the executing processor; the exact details depend on the interaction between the processor and the language support for large integers, sometimes referred to as ‘bignums’). In the linearization parsing literature, Ramsay (1999) seems to have been the first to explore this possibility, giving definitions of OVERLAP and COMBINE and an alternative way of computing ISOLATED.

5.2.1.2 The Dot

It should be observed at this point that the GIDLP parser retains the notion of a single active element per edge, even though any of the RHS categories might occur leftmost in the input string; this is in contrast to Suhre (1999), who essentially follows the direct ID/LP parsing tradition (see (Volk 1996) and references cited therein) by predicting all graph-minimal categories – those categories which no constraint on the rule requires anything to precede – in every rule. For example, in the grammar presented in (69) (originally presented as (55)), there would initially be a dot before the categories B, C, E, and H.

(69)

$$\begin{tikzcd}
A \arrow[r] & B & C \\
D \arrow[r] & E \arrow[r, two heads] & F \\
G \arrow[r] & H \arrow[r] & I
\end{tikzcd}$$

[2]For example, Davis (2002) mentions the use of integer representations of bitvectors in the context of machine translation, and some of the inspiration for the bitvector computations below stems from bitboard-based computer chess discussions on `rec.games.chess`.

SINGLETON(p): In which bitvector is p the only active position?
Computed as 2^p
Example: singleton(1) is 00010.

OVERLAP(x, y): Is there a position active in both x and y?
Computed as AND(x, y) $\neq 0$
Example: 10111 and 01010 have a bitwise-AND of 00010, and therefore overlap.

COMBINE(x, y): What is the union of x and y?
Computed as OR(x, y)
Example: 10110 and 01010 combine to form 11110.

RBOUND(x): What is the most-significant active bit in x?
Computed as $\lfloor \log_2(x) \rfloor$
Example: the right bound of 01010 is 3, which is $\lfloor 3.32 \rfloor$.

LBOUND(x): What is the least-significant active bit in x?
Computed as RBOUND(XOR($x, x - 1$))
Example: the left bound of 01010 is 1, which is the right bound of 00011 = XOR(01010, 01001).

PREFIX(p): What bitvector covers all positions $\leq p$?
Computed as $2^{p+1} - 1$
Example: PREFIX(3) is 01111 (15).

SUFFIX(p): What bitvector covers all positions $\geq p$?
Computed as NOT($2^x - 1$)
Example: SUFFIX(3) is NOT(00111) = 11000.

PRECEDE(x, y): Does x completely precede y (where x and y are assumed not to overlap)?
Equivalent to $x < y$
Example: 00011 (3) precedes 01100 (12).

IPRECEDE(x, y): Does x immediately precede y (where x and y are assumed not to overlap)?
Equivalent to RBOUND(x) = LBOUND(y) $- 1$
Example: the right bound of 00011 is 1, and the left bound of 01100 is 2, so 00011 immediately precedes 01100.

ISOLATED(x): Does x form a continuous unit?
Equivalent to $x =$ AND(PREFIX(RBOUND(x)), SUFFIX(LBOUND(x)))
Example: With the vector 01101, AND(01111, 11111) is 01111 $\neq$ 01101, so 01101 is not isolated.

Figure 5.2: Common Operations on Bitvectors

In Earley's original parser, the dot can be seen to serve two purposes: (1) it indicates the portion of the *string* that has already been incorporated into chart edges; and (2) it distinguishes the *categories* that have been found from those that are left, highlighting the active category as the category to next be located.

In the GIDLP parsing algorithm, the first purpose is served by the coverage vector (as discussed in the previous section); thus the dot only has the second purpose. Since every element on the RHS of a rule has to be found at some point for that rule to be successfully completed. And as was stated in section 4.1, the RHS of a GIDLP grammar is conceptually a set of categories – not an ordered list. As a result, the use of multiple dots is unnecessary.

When only a single dot is used, a GIDLP parser is free to use the RHS order for other purposes. The parsing algorithm described in this chapter uses the RHS order to determine the order in which the RHS categories are predicted. As a result, the grammar writer can use the order to specify those daughters to be searched first which are most likely to cause an early failure. For example, a rule introducing a conjunction of sentences can be specified as (70).

(70) $\mathrm{S} \rightarrow \mathrm{Conj}_1, \mathrm{S}_2, \mathrm{S}_3\ ;\ 2 \ll 1,\ 1 \ll 3\ ;\ \langle\{2\}, [], \mathrm{S}\rangle, \langle\{3\}, [], \mathrm{S}\rangle$

This causes the parser to look for the easy-to-identify conjunction before it tries to find the potentially-complex conjunct sentences.

One might object that it is unreasonable to expect a grammar writer to take processing considerations into account. It must be observed, however, that the RHS order has no impact on the correctness of the resulting parse. Much of the work of finding the optimal ordering for the categories in a rule could be accomplished either at compile time by heuristics of varying complexity, by profiling the grammar on a suitably-exhaustive test-suite (akin to the quick check computation described in section 6.1.2), or by hand as the grammar is written. For the sample grammars presented in this chapter, two heuristics were followed: categories should be ordered by the number of times each is mentioned in an order constraint in that rule, and exclusively pre-terminal categories should be ordered before other categories. Both of these considerations, for instance, lead to ordering the **Conj** element first in (70). In section 7.3.2, a similar heuristic based on head status will be seen.

5.2.1.3 LP Constraint State Representation

One of the key criticisms made in chapter 3 of many algorithms for parsing with discontinuous constituents was that the LP constraints, if present at all, were only used in a generate-and-test fashion. In order for the GIDLP parsing algorithm to not fall prey to this same criticism, it must be possible for the prediction step to intelligently take word order constraints into account.

In principle, once a daughter of an active edge has been found, the other daughters should only be predicted to occur in string positions which are compatible with the word order constraints of the active edge. For example, consider the edge in (71).[3]

(71) $\mathrm{A} \rightarrow \mathrm{B}_1 \bullet \mathrm{C}_2\ ;\ 1 < 2$

This notation represents the point in the parse during which the application of a rule $A \rightarrow B_1 C_2; 1 < 2$ has been predicted, and a B has already been located. For the sake of example, assume that B has been found to cover the third position of a five-word string. Two facts

[3]Many of the examples in this chapter and the next will be given in terms of arbitrary a-b-c categories. This allows the examples to be presented with as little scaffolding as possible.

are now known: from the order constraint $1 < 2$, *C* cannot precede *B*, and from the general principle that the RHS of a rule forms a (perhaps non-contiguous) partition of its LHS, *C* cannot overlap *B*. Thus *C* cannot cover positions one, two, or three.

The core of the GIDLP parsing algorithm, and one of the central insights of this thesis, is that the same data structure used to describe the coverage of an edge can encode these kinds of restrictions on the parser's search. This is accomplished through two additional uses of bitvectors: as *negative masks* (n-masks) and as *positive masks* (p-masks).

The n-mask on an edge constrains the set of possible coverage vectors for the edge. The 1-positions in an n-mask represent the positions that are masked out: the positions that cannot be filled when completing this edge. The 0-positions in the negative mask represent positions that may potentially be part of the edge's coverage. For the edge in (71), the coverage vector for the edge would be 00100, since only the third word *B* has been found so far. Assuming no additional restrictions from a higher rule in the same domain, the n-mask for any edge predicted to provide *C* would be 00111, encoding the fact that the final coverage vector of the edge for *C* must be either 01000, 10000, or 11000 (that is, *C* must occupy position four, position five, or both of these positions). In essence, then, the negative mask encodes information on where the active category cannot be found.

In contrast, the p-mask encodes information about the positions the active category **must** occupy. This knowledge often arises from immediate precedence constraints. For example, consider the edge in (72).

(72) $\mathrm{D} \rightarrow \mathrm{E}_1 \bullet \mathrm{F}_2$; $1 \ll 2$

If an *E* occupies position one, then any *F* must at least occupy position two; this would be represented by a p-mask of 00010.

Taken together, the n-mask and p-mask encode all information available to the parser when the edge is constructed about the edge's potential coverage.

5.2.1.4 Domains in Earley's Algorithm

One of the primary advantages of a chart parser like Earley's algorithm is the fact that passive edges need only be constructed once; if a given passive edge doesn't immediately trigger a completion, it remains in the chart to be picked up during future completions. As a result, it won't always be the case that the parser knows what domain a given passive edge will be used in. Consider the grammar in (73):

(73)
a) root(A, [])
b) $\mathrm{A} \rightarrow \mathrm{B}_1, \mathrm{C}_2$; $1 < 2$
c) $\mathrm{B} \rightarrow \mathrm{D}_1$
d) $\mathrm{C} \rightarrow \mathrm{D}_1$; ; $\langle\{1\}, [\mathrm{E} < \mathrm{F}], \mathrm{D}\rangle$
e) $\mathrm{D} \rightarrow \mathrm{E}_1, \mathrm{F}_2$

This grammar accepts the strings *EFEF* and *FEEF* and rejects the strings *EFFE* and *FEFE*. Consider the process of parsing the string *FEFE*. Given rule (73e), the last two symbols of the string constitute a *D*. By rule (73c), this *D* is also a *B*. It is not, however, a *C*: rule (73d) states that *C*s dominate contiguous sections of the input within which all *E*s precede all *F*s. As presented so far, a passive edge stores relatively little information: a category label and a bitvector representing the edge's coverage. In the example above, the passive edge would simply say that a *D* has a coverage of 1100. It is impossible to tell from this alone whether such a *D* would be an acceptable *C* or not.

The parser could simply store each edge's parse tree (leaving out daughters of compacted constituents) and examine that tree each time a new domain constraint becomes relevant (as in the example above). This is inefficient in general, though – the parse tree of a given edge might turn out to be arbitrarily deep. In addition, under this solution even a recognizer would have to maintain candidate parse trees, thus negating any practical difference between a recognizer and a parser for the GIDLP formalism.

Instead, the parser uses the notion of *dormant* (contrasted with *active*) order constraints. When the grammar is compiled (see section 5.2.2.1) all domain-local LP constraints (that is, those not introduced on individual rules) are added as dormant constraints in every other domain. A dormant constraint is still tracked and updated (as will be described in section 5.2.2.9) as normal, with the exception that a constraint violation does not prevent an edge from being created (as a violation in an active constraint does). Instead, it merely reduces the number of domains that the edge could be completed into in the future. When a domain-introducing rule is predicted from a mother edge, if any of the dormant constraints for that domain had already been violated on that edge, that prediction will not create any new edges. In essence, the set of dormant constraints can be seen as a 'distilled' version of the edge's parse tree: the minimal amount of information needed to determine future domain memberships.

Thus in the example above – attempting to parse the ungrammatical string *FEFE* – the last two symbols would still parse as a *D*, albeit one that violates the dormant constraint E < F. This *D* would then parse as a *B* via the second rule without a problem, but correctly fail to parse as a *C* via the third rule, as the dormant constraint would become active at that point.

5.2.1.5 Edge Subsumption and Ambiguity Packing

The primary intuition behind a chart parser is that any given parsing hypothesis is only examined once; as such, chart parsing falls into the general class of *memoizing* algorithms, which record intermediate results to avoid redundant computation. In context-free parsing, where edges are small and contain relatively little information – two pointers to string positions, a pointer to the governing rule, and a pointer to the dot's position within the rule – four pointer-equality tests are enough to detect whether a proposed edge would be redundant and should be blocked from the chart.

An edge for a GIDLP parser contains much more information, however – a coverage bitvector replaces the start and end pointers, and instead of just storing a pointer to the rule, the parser must keep track of the rule's constraints as they've been updated (as will be described in section 5.2.2.9. In addition, a binary equal–nonequal distinction is no longer sufficient when comparing fields from different edges: instead, one field of an edge may be more specific, identical, or less specific than the other. These notions generalize to the edge as a whole: one edge is at least as general as another as long as all of its fields are at least as general as the corresponding fields on the other edge.

The notion that an edge would be redundant if a more general edge is already present in the chart is a straightforward generalization of the nature of the chart; Oepen and Carroll (2000) refer to this process as *proactive ambiguity packing*. Less clear, however, is the related situation in which an edge proposed for addition to the chart is more general than an edge in the chart. There are two options: a parser can add the new edge to the chart, in which case some derivations will end up being performed multiple times (as any consequence of the existing edge will also be a consequence of the new edge), or the edge already in the chart can be removed and replaced with the new edge; this process is referred to as *retroactive ambiguity packing*. On the basis of some practical experiments, Oepen and Carroll (2000) conclude that a parser using both of these forms of ambiguity packing has a clear empirical advantage over one that does not. They do not, however, provide any results on the relative merits of proactive and retroactive ambiguity packing.

It is clear that the implementation of retroactive ambiguity packing in the GIDLP parsing algorithm would be expensive – the parser would need to walk through the agenda and replace references to the old edge (in the parsing agenda and in the backpointer classes discussed in the next section) with the new edge, and recompute the consequences of the old edge. Thus in the absence of evidence that retroactive ambiguity packing leads to an overall increase in efficiency, the GIDLP parsing algorithm does not currently include retroactive ambiguity packing.

5.2.1.6 Backpointer Classes

When a passive edge is created through completion, the parser needs to find edges to complete the new edge with. In Earley's algorithm, each edge stores a pointer to the initial position of its ancestor edge; when the edge needs to be completed, only those edges with the given initial position are considered.

With the GIDLP parsing algorithm, there are no longer 'initial positions' to be used in such a manner, and a replacement must be found. The easy way out is to attempt completion of the new passive edge with any active edge seeking the category that the passive edge provides. An optimization (somewhat analogous to what is done in some unification-based parsers; see section 6.1.1.2) is to only complete an edge with the ancestor edge that spawned it in the first place. The idea seems simple to implement in practice – store a pointer to the responsible edge when an edge is predicted, and propagate that pointer through all edges completed from that edge.

This is complicated by ambiguity packing, however. In particular, if a predicted edge P1 is not added to the chart because another identical edge P2 is already there, then the edge responsible for P1 needs to also be considered responsible for P2. For example, assume a grammar with rules S → NP, VP; NP → PP, NP; and PP → P, NP (among others). Assume that a parse starts with edge (74). The category NP is predicted, and edge (75) is created, storing a backpointer to its responsible edge. Next, PP is predicted, leading to (76) with a backpointer to its responsible edge. Assume the parser then completes (77) with a P, yielding edge (77); the backpointer is unchanged. Predicting the NP leads to edge (78), which won't be entered into the chart as it duplicates edge (75). Then when edge (75) is completed (via other rules), its backpointer only points to (74), and so the parser would never try to complete that edge with (77), and the parse would not succeed.

(74) S → • NP VP; from ...

(75) NP → • PP NP; from (74)

(76) PP → • P NP; from (75)

(77) PP → P • NP; from (75)

(78) NP → • PP NP; from (77)

It might be argued that it is incorrect to let an edge be blocked from addition to the chart in the presence of differing backpointers; after all, the concept of a chart is that only exact duplicates are blocked. Following that argument, however, would lead the parser to become susceptible to the same problem that Earley's algorithm was designed to avoid: nontermination in the presence of left recursion. Consider a grammar with one rule A → A A (and one lexical item with category A). The parser initially predicts edge (79). Prediction from (79) yields (80); prediction from (80) yields (81), and so on. Each edge has a different backpointer than the previous edge, and the parser will therefore never terminate.

(79) A → • A A; from ...

(80) A $\rightarrow \bullet$ A A; from (79)

(81) A $\rightarrow \bullet$ A A; from (80)

This problem can be resolved by maintaining equivalence classes of backpointers, which will be referred to as *backpointer classes*, rather than backpointers themselves. Each backpointer class is a group of pointers to parents (active edges) and children (passive edges), and each edge is marked with its backpointer class number. When an edge formed by prediction is added to the chart, it is assigned a backpointer class containing only its parent. When an active edge formed by completion is added to the chart, it is assigned to the same backpointer class as its parent active edge. (Note that these steps are unchanged from the strategy described above.) New to this approach is the idea that when an edge formed by prediction is blocked from being added to the chart, its parent is added to the parent set of the backpointer class of the edge that blocked addition into the chart.

The child set of a backpointer class then contains all passive edges having the parent edges as their ancestors: when a passive edge formed by completion is added to the chart, it is added to its backpointer class as a child. When such an edge is blocked from the chart, indicating a structural ambiguity (see section 6.1.1.1), the parser must make that edge available to its ancestors for completion without adding a redundancy to the chart. This is done by merging the backpointer class of the blocked edge with the backpointer class of the blocking edge – in effect declaring them to have been the same backpointer class.

Thus, in the situation described in the example above, when (78) was blocked from the chart, its responsible edge (77) would be added to the parent set of (75)'s backpointer class; thus any passive edges resulting from (75) will be available for completion with (77), and the parser will now give the correct results.

5.2.2 The Algorithm Itself

Now that the building blocks have been presented, the algorithm for parsing GIDLP grammars can now be given in figure 5.3. The following sections will explain each of the steps in the algorithm; section 5.3 will provide two example parses to serve as macro-level examples.

5.2.2.1 Grammar Compilation

Grammars are compiled into an internal representation prior to parsing; this is necessary for two main reasons. First, the grammar format contains a number of abbreviatory devices: syntactic sugar for some domain declaration cases and the chance to use global order constraints instead of stating a constraint every time a domain is established. These abbreviations need to be normalized so that the parsing algorithm can consider them in a singular fashion. Additionally, mixed LP constraints (ones written in terms of a rule token and a category description) must be stated in rules, so that the parser knows which rule's tokens are indicated, yet must be kept track of as domain-level constraints, in case the described category is located first.

The processes described so far are mechanical, but three steps deserve further attention. Following (Haji-Abdolhosseini and Penn 2003), the minimum and maximum category yields of each rule are calculated during grammar compilation for use during the parse (see section 5.2.2.6). Finally, the integrity of the grammar may be checked by calculating the *reachability* of each domain (the set of categories that may occur within the domain). In so doing, it is possible to locate domain-level constraints mentioning unreachable categories (that is, those that will never affect the parse). An implementation can then point

```
PROCEDURE earley:
  COMPILE the grammar (section 5.2.2.1)
  INITIALIZE the chart and the agenda (section 5.2.2.5)
  WHILE the action queue is non-empty
    DEQUEUE the top action and perform it
  EXAMINE the chart for successful parses

PROCEDURE predict(A): (section 5.2.2.6)
  FOREACH grammar rule providing A's active category
    COMPUTE the masks for the new edge (section 5.2.2.7)
    EVALUATE the masks for edge viability
    attempt to ADD the edge to the chart
    ENQUEUE prediction and completion with this edge

PROCEDURE complete(P, A): (section 5.2.2.8)
  CHECK that A and P do not overlap
  UPDATE the LP constraints and domain declarations on the edge (section 5.2.2.9)
  attempt to ADD the new edge to the chart
  ENQUEUE prediction and completion with this edge
```

Figure 5.3: GIDLP Parsing Algorithm

these out to the user for corrective action. Each of these steps will now be described in turn.

5.2.2.2 Minimum Yield

The minimum yield of each rule is defined as the minimum number of lexical items that the LHS of the rule could dominate. This calculation can be seen as an instance of bottom-up grammar flow analysis (GFA): a family of related problems that can be solved by the same generic procedure (Möncke and Wilhelm 1982). In their terminology, the *flow information space* – the space of possible answers to the question being asked – is the whole numbers. The *information propagation function* (which computes the value for a rule given the values for each RHS element) is addition. The *information combination function* (which computes the value for a non-terminal based on the values for each rule generating that non-terminal) is minimum.

A variety of algorithms (Jourdan and Parigot 1990) are available to solve GFA problems; at an abstract level, the value for each rule and non-terminal are computed in turn until a fixed-point is reached. Optimization techniques can analyze a given grammar to find the ideal order in which values should be computed; this is, of course, only merited when the individual computations are more time-consuming than the optimization analysis.

To give an example of the minimum yield calculation, consider the grammar[4] in figure 5.4. Here, the minimum yield of the first rule is (via the information propagation function) the minimum yield of an *np* plus the minimum yield of a *vp*; the minimum yield of an *np* is (via the information combination function) the minimum of the minimum yield of rules (b) and (d). Applying a GFA algorithm gives minimum yields of 5, 2, 3, 5, 6, and 3 for rules (a) – (f), respectively, as illustrated by the sample constituents.

[4]Since LP constraints are irrelevant for this example, they have been omitted.

a)	s → np, vp	*the dog met the cat*
b)	np → det, n	*the dog*
c)	vp → v, np	*met the cat*
d)	np → np, pp	*the cat on the lawn*
e)	vp → vp, pp	*met the cat on the lawn*
f)	pp → p, np	*on the lawn*

Figure 5.4: Minimum Yield Example

5.2.2.3 Maximum Yield

In principle, the task of calculating the maximum yield of each rule is an almost-identical instance of GFA; the only change is that the information combination function must now be maximum instead of minimum. With this change, however, the GFA algorithm is no longer guaranteed to terminate; any recursion in the grammar, whether direct or indirect, leads to an unbounded maximum yield. Thus the actual maximum yield problem is to first determine whether a rule has an infinite or finite maximum yield, and then to compute the finite maximum yields in the grammar. For instance, in the grammar of figure 5.4, only rule (b) has a finite maximum yield (namely, 2).

The detection of those rules involved in a recursion is straightforward, as any such recursion must correspond to a cycle in the corresponding grammar graph (Möncke and Wilhelm 1982). A grammar graph has nodes corresponding to each rule and nonterminal in the grammar; each rule node then has in-edges from each nonterminal on its RHS and an out-edge to its LHS nonterminal. For example, the grammar graph of the grammar in figure 5.4 is given in figure 5.5. In this figure, the square nodes represent rules, while the circular nodes represent non-terminals.

Each cycle in this graph (for instance, $NP \rightarrow 6 \rightarrow PP \rightarrow 4 \rightarrow NP$) is a recursion; every node along such a cycle will have infinite maximum yield. Now recall from graph theory the notion of a *strongly connected* set of nodes: a path exists from every node in the set to every other node in the set. A *strongly connected component* (SCC) of a graph is then a strongly connected set of nodes such that the addition of any other node would cause the set to no longer be strongly connected. A graph may be partitioned into its SCCs; those SCCs containing only a single node (*singleton SCCs*) correspond exactly to those nodes not involved in any cycles. Finally, note that for any directed graph G with nodes $g_1 \ldots g_n$, a graph G' can be constructed whose nodes $g'_1 \ldots g'_n$ are the SCCs of G and which contains an edge (g'_i, g'_j) whenever an edge (g_a, g_b) exists in G and it is the case that $g'_i = SCC(g_a)$ and $g'_j = SCC(g_b)$; this is the *SCC graph* of G. For example, the SCC graph of figure 5.5 is given as figure 5.6.

Once the cycles have been detected, the SCC graph can be topologically sorted to provide an order in which the maximum yield of each node can be calculated in a single pass. A topological order on a graph is an order on the nodes of a graph such that, for any pair $x < y$ in the order, there is no path from y to x in the graph. For example, one topological order in figure 5.6 is Det, N, 2, P, V, SCC, 1, S.

The algorithm for calculating maximum yields is therefore as follows:

1. Construct the grammar graph G and the corresponding SCC graph G'.
2. Assign a maximum yield of ∞ to all nodes in non-singleton SCCs.
3. Calculate the maximum yield of each SCC node in topological order, according to the SCC graph.

Here, being (exclusively) preterminals, Det, N, P, and V all have maximum yield 1; r2 has maximum yield 2; and the remaining nodes have infinite maximum yield.

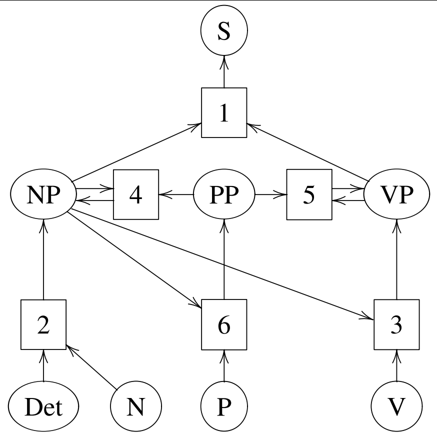


Figure 5.5: Grammar Graph

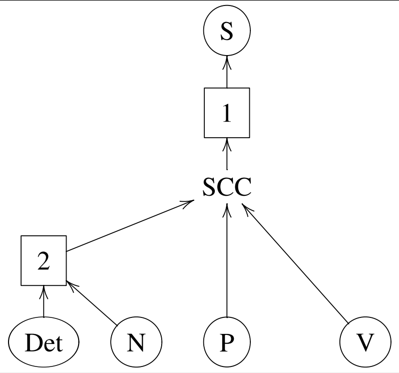


Figure 5.6: Strongly Connected Component Graph

5.2.2.4 Reachability

A constraint can be described as *relevant* to a given domain whenever both of its categories are simultaneously reachable in that domain. Thus the final task in grammar compilation is to determine the *reachability* of each domain: the set of categories that may occur inside that domain. Closely related is the notion of category reachability: the set of categories that can occur in the same domain as a given category. Note that reachability must be represented as a set of sets, as categories may have disjunctive expansions: in figure 5.7, for example, the category E may expand as to either a J and a G, or an H and an I.

The compiler first calculates the immediate reachability of each category: for each rule generating that category, it takes the set of RHS categories, minus those categories mentioned on domain declarations, plus the result categories of any domain declarations (collectively, the *visible categories* of that rule). Each such set is a member of the immediate reachability of that category. For example, the immediate reachability of each category in figure 5.7 is shown on the left side of figure 5.8.

The GFA algorithm is then used to calculate the reachability of each category. Here, the value space is sets of sets of categories. The information propagation function returns the union of the visible categories on each rule with the reachability of those categories that are not the result category of a domain declaration. The information combination function $\times$ takes two sets of sets and forms a "product" of the two sets: given S_1 and S_2,

a) start(A) ; B < C
b) A → B, [C]
c) C → D, E, F ; ⟨{1,3}, [D < F, E ≪ F], J⟩
d) E → F, G ; 1 < 2
e) E → H, I ; 1 < D
f) [F] → F, C ; E ≪ 1
g) E < B, F < I, F < G

Figure 5.7: Example Grammar for Reachability

	Initial	Final
A	{{B, C}}	{{B, C}}
C	{{J, E}}	{{E, F, G, J}, {E, H, I, J}}
E	{{F, G}, {H, I}}	{{F, G}, {H, I}}
F	{{F}}	{{F}}

Figure 5.8: Category Reachability

it returns $\{x | a \in S_1, b \in S_2, x = a \cup b\}$. For example, $\{\{a, b\}\} \times \{\{c, d\}\} = \{\{a, b, c, d\}\}$ and $\{\{a, b\}, \{c, d\}\} \times \{\{e, f\}, \{g, h\}\} = \{\{a, b, e, f\}, \{a, b, g, h\}, \{c, d, e, f\}, \{c, d, g, h\}\}$. The final reachability of each category is given on the right side of figure 5.8.

At this point, the reachability of each domain can be calculated. Each domain reaches the categories it contains as well as the categories they reach. Since at this point the full reachability of each category is known, the domain reachabilities can be computed in a single pass. Figure 5.9 gives the reachability for each domain in figure 5.7.

Now the information about each domain's reachability can be used to optimize the placement of the domain-level order constraints. For a constraint to be relevant to a given domain, both of its components must be reachable in that domain. In Figure 5.7, the domain declaration in rule (c) contains the constraint E ≪ F, yet E and F are never simultaneously reachable in that domain; as a result, that constraint is flagged as erroneous. At the same time, global constraints and mixed token–description constraints are moved to the active constraint list on each relevant domain and to the dormant list on each other domain. The final compiled grammar, showing only active constraints in each domain, is given in figure 5.10.

5.2.2.5 Parser Agenda

The parser is controlled by an agenda: a queue of parsing tasks (for instance, predict from edge X, or complete edges Y and Z). Each prediction/completion step can add any number of tasks to the agenda as follows:

- When an active edge is added to the chart, it is scheduled for prediction and for completion with each passive edge providing its active category.
- When an active edge formed by prediction is blocked from the chart, its parent is scheduled for completion with all the children in the backpointer class of the edge that blocked it.
- When a passive edge formed by completion is added to the chart, it is scheduled to complete with each parent in its backpointer class.

a)	$\{\{A\}\} \times \text{reach}(A)$	$= \{\{A, B, C\}\}$
b)	$\{\{C\}\} \times \text{reach}(C)$	$= \{\{C, E, F, G, J\}, \{C, E, H, I, J\}\}$
c)	$\{\{D, F\}\} \times \text{reach}(D) \times \text{reach}(F)$	$= \{\{D, F\}\}$
f)	$\{\{F, C\}\} \times \text{reach}(C) \times \text{reach}(F)$	$= \{\{C, E, F, G, J\}, \{C, E, F, H, I, J\}\}$

Figure 5.9: Domain Reachability

a) start(A) ; $B < C$
b) $A \rightarrow B, C$; $\langle\{2\}, [F < G], C\rangle$
c) $C \rightarrow D, E, F$; $\langle\{1, 3\}, [D < F], J\rangle$
d) $E \rightarrow F, G$; $1 < 2$
e) $E \rightarrow H, I$
f) $F \rightarrow F, C$; $E \ll 1$, $\langle\{1, 2\}, [F < I, F < G], F\rangle$

Figure 5.10: Compiled Grammar

- When an active or passive edge formed by completion is blocked from the chart, triggering a class merger (see section 5.2.1.6), the new edges on each side are scheduled for completion.

Initially, the chart is seeded with passive edges corresponding to each word in the input. The parser then jumpstarts the agenda by predicting an isolated instance of the start symbol covering the entire input. Once the agenda is empty, the parser will examine the chart for successful edges.

5.2.2.6 Prediction

Recall that prediction takes an active edge seeking its active category and produces a new edge that will provide that category if successfully completed. The parser considers each rule in the grammar that provides the symbol being predicted, and for each rule, it generates the masks for the new edge, taking both rule-based and domain-based order constraints into account (as described in the following section).

Once the new masks have been calculated, the parser ensures that an edge is only entered into the chart if the n-mask has enough space for the rule being considered: that is, the number of inactive positions in the n-mask cannot be smaller than the minimum yield of the rule (as calculated during grammar compilation).

If this check passes, the parser will attempt to add the newly-predicted edge to the chart with an empty coverage vector. Whether this attempt succeeds or fails will determine the parsing tasks added to the agenda, as described in section 5.2.2.5.

5.2.2.7 Mask Computation

The new edge's n-mask is initially set to the n-mask of the triggering edge. If the triggering edge has a singleton RHS (indicating that the edge being predicted must lead to a final completion of the trigger), then the new edge's p-mask is set to the p-mask of the trigger, minus the trigger's coverage. That is, any position that the trigger must cover but has not yet done so must be covered by this new edge. For example, if an active edge with a singleton RHS has a p-mask of 00111 and covers 00110, the edge being predicted must at least cover 00001. If the trigger has a non-singleton RHS, then the p-mask is initialized to zero.

The parser then examines each LP constraint in turn for references to the category currently being predicted. Each reference provides a component of the final mask, as indicated in figure 5.11.

Here, Cat refers to the category being predicted (in the form of either a description or a token), and Loc refers to the location of an already-located category. (Thus constraints that do not mention the category being predicted and constraints that do not refer to any already-located categories are irrelevant for prediction.)

In addition to these LP constraint patterns, the parser can also generate mask components based on domain declarations that only need one more category to be complete (**singleton** domain declarations). Since a domain declaration must dominate a contiguous sequence of positions in the input, the last-located category must fill any gaps in the current coverage of such a statement. These gaps are therefore activated in the corresponding p-mask component. For instance, if a domain declaration ⟨{3}, , J⟩ covering 01010 is on an edge from which the category with token 3 is being predicted, the component 00100 should be added to the p-mask.

Each constraint in turn is checked to see which (if any) of these patterns it matches, and the relevant mask components are calculated. At the end, all n-mask components are COMBINED with the initial n-mask, and all p-mask components with the initial p-mask: these are the new masks. As long as the masks do not overlap (reflecting contradictory requirements), the edge will be proposed for addition to the chart.

For example, if the parser is predicting *verb* as token 2, it would respond to some sample constraints as shown in figure 5.12. For the first constraint, the parser sees that categories matching the description *verb* must follow the position 00010, generating the negative mask component 00011. The second constraint doesn't contribute anything, as it does not refer to any as-yet determined locations. The third constraint requires category number 2 to immediately precede the position 01000. This generates both a negative mask component (11000), which encodes the fact that the category cannot follow the position given, and a positive mask component (00100), which encodes the fact that the category must include the third position.

5.2.2.8 Completion

Recall that completion is the process of combining a compatible pair of active and passive edges to produce a new edge. The parser considers each edge compatible with the edge that triggered completion. For each such edge, the parser constructs the LP statements for the new edge (as described in the following section), the new n-mask, and the new coverage vector. If this was a final completion (the resulting edge is passive), the parser must additionally check to see that the p-mask was respected. Edges are indexed by their active element, so the parser can efficiently retrieve only those edges which are likely to yield successful completions.

5.2.2.9 Order Constraint Updating

For each edge retrieved, the parser must update the word order constraints of the active edge with the coverage of the passive edge. As edges are initially constructed from grammar rules, all order constraints are initially expressed in terms of either categories or tokens. (In this chapter, constraints stated entirely in terms of tokens will be referred to as *token-based*, while all others will be referred to as *description-based*.) As the parse proceeds, these constraints are updated in terms of the actual locations where matching constituents have been found. For example, a constraint like $1 < 2$ (where 1 and 2 are tokens) can be updated with the information that the constituent corresponding to token 1 has been found as the first word, i.e. as position 00001.

	n-mask component	p-mask component
Cat < Loc	SUFFIX(LBOUND(C))	
Loc < Cat	PREFIX(RBOUND(C))	
Cat ≪ Loc	SUFFIX(LBOUND(C))	SINGLETON(LBOUND(C) - 1)
Loc ≪ Cat	PREFIX(RBOUND(C))	SINGLETON(RBOUND(C) + 1)

Figure 5.11: Mask Computation

	n-mask component	p-mask component
00010 < verb	00011	
3 < verb		
2 ≪ 01000	11000	00100
result	11011	00100

Figure 5.12: Mask Computation Example

5.2.2.10 Activating dormant constraints

Before this can be done, however, the parser must first check to see if any dormant constraints on the passive edge need to be activated, based on the active edge's domain. The first step of this process is to find the domain that the passive edge is being completed into. If the active element of the active edge is referred to in a domain declaration, then that domain is the relevant one. Otherwise, the passive edge is assumed to be completed into the active edge's own domain. Once the relevant domain has been determined, the constraints for that domain will be either active or dormant on the passive edge. If dormant, they must be merged with the active edge's active constraints; all other dormant constraints on the passive edge are merged with the active edge's dormant constraints. The procedure for merging constraints will be discussed below.

5.2.2.11 Updating token-based constraints

Each update step will take one of the following forms:

- The first time one of the categories mentioned in a precedence constraint has been found, the constraint is updated as above and tested to see whether there is enough space for the other category. For example, if, given the constraint $1 \ll 2$, constituent 2 is found as the first word of the string, the constraint will be impossible to fulfill.
- When the remaining category of a precedence constraint is found, the parser checks that the constraint actually holds; if it does, then that constraint will not appear as part of the word order constraint set of the resulting edge. If the constraint is discovered to have been violated, the completion step aborts and no edge is added to the chart.

5.2.2.12 Updating and merging description-based constraints

With description-based constraints, weak and immediate precedence constraints must be handled separately.

In a weak precedence constraint, the parser need only keep track of the most extreme matching cases (the *left* and *right frontiers*), if any. For example, the constraint NP < VP needs to keep track of the rightmost **NP** and the leftmost **VP** seen in the domain so far. As long as the rightmost **NP** remains to the left of the leftmost **VP**, the constraint will never cause completion to fail. Similarly, when merging two weak precedence constraints, the most extreme version of each frontier is kept.

Immediate precedence constraints, on the other hand, need only keep track of whether zero, one, or many of each part of the constraint has been seen. The possibilities are illustrated in figure 5.13. Here, the rows represent the status of the lefthand side of the constraint, and the columns represent the status of the righthand side; the cells then encode whether that particular status indicates a constraint violation. For instance, if the parser knows that NP $\ll$ VP in this domain, it is acceptable for there to be several **VP**s, as long as no **NP**s are present (since, by definition, a category cannot immediately precede multiple locations).

To summarize, immediate precedence constraint violations can be detected by keeping count of the occurences of each side of the constraint; the one exception is the situation represented by the center cell, where each side has been observed once. In this case, the parser must check whether the locations are properly adjacent.

The corresponding chart for merging two count values, given in figure 5.14, is straightforward.

A complication arises when a category matches both halves of a precedence constraint – if both halves are updated, the constraint will appear to have been violated, since no position precedes itself. Thus weak precedence constraints[5] must actually maintain a third frontier in addition to the left and right frontiers to be updated in such circumstances. Any subsequent update to any of the three frontiers is considered a constraint violation when the third frontier is non-empty.

5.2.2.13 Completion – Final Steps

Once the word order constraints have been successfully updated, the rest of the new edge is easy to compute: the category of the edge is the category of the active edge, the missing righthand side is the tail of the active edge's righthand side, and the coverage vector is the bitwise OR of the two edges' coverage vectors. Finally, if this was a final completion (that is, the edge being created is passive), the parser checks to see if the p-mask was respected: in other words, every active bit in the p-mask must be active in the new coverage vector. The resulting edge is then added to the chart and itself triggers another round of completion and prediction.

5.3 Sample Parses

Having described the parsing algorithm in general, the paper will now present some concrete examples illustrating the parser's actions on a specific grammar and input sentence.

5.3.1 Relatively-free Word Order

This sample parse uses the toy Sanskrit-inspired grammar in (82) and the sentence in (83).[6]

(82)
a) start(s).
b) s $\rightarrow$ conj, [s], [s] ; 2 $\ll$ 1, 1 $\ll$ 3
c) s $\rightarrow$ verb, nom ; 2 < 1
d) s $\rightarrow$ verb, nom, acc ; 2 < 1, 3 < 1
e) acc $\rightarrow$ adj, acc
f) nom $\rightarrow$ नलस् 'Nala' (a proper name)
g) acc $\rightarrow$ नगरम् 'city'

[5]Since, as mentioned in section 4.1.5 there is no semantic difference between A < A and A $\ll$ A, the grammar compiler converts all instances of the latter into the former.
[6]The example has been tokenized from रुचिरं नलो नगरमगच्छश्च नलो ऽवदत्.

	$A \ll B$	B		
		Zero	One	Many
	Zero	OK	OK	OK
A	One	OK	OK if precedence is respected	Failure
	Many	OK	Failure	Failure

Figure 5.13: Immediate Precedence Frontier Updating

	$A + B$	B		
		Zero	One	Many
	Zero	Zero	One	Many
A	One	One	Many	Many
	Many	Many	Many	Many

Figure 5.14: Immediate Precedence Frontier Merging

h) verb → अगच्छत् ‘went’
i) conj → च ‘and’
j) verb → अवदत् ‘spoke’
k) adj → रुचिरम् ‘shining’

(83)

रुचिरम्	नलस्	नगरम्	अगच्छत्	च	नलस्	अवदत्
shining	Nala	city	went	and	Nala	spoke

‘Nala went to the shining city and Nala spoke’

The grammar can be summarized as follows: A sentence may consist of a verb and either one or two arguments preceding it. A sentence may also consist of a conjunction immediately between two (conjunct) sentences, each of which forms an isolated domain. Finally, accusatives may be modified by an adjective which may occur anywhere in a sentence, before or after the accusative it modifies. Note also that the example sentence contains the discontinuous constituent रुचिरम् नगरम् ‘shining city’.

In these parse traces, the first row contains the line number of the parsing step and a brief description of the parser action. The second row then contains either the details of the resulting edge (edge number, coverage vector, dotted rule, word order statements) or the reason that no edge was created. Any inactive bit in the coverage vector that is active in the n-mask is shown struck-through: ~~0~~. Inactive bits in the coverage vector that are active in the p-mask are underlined: $\underline{0}$. Other edge components not relevant to the example are not shown.

Before parsing, the parser seeds the passive chart with the lexical entries, each covering a singleton vector.

Line 1:	SCAN रुचिरम्	⇒ p0: adj at 0000001
Line 2:	SCAN नलस्	⇒ p1: nom at 0000010
Line 3:	SCAN नगरम्	⇒ p2: acc at 0000100
Line 4:	SCAN अगच्छत्	⇒ p3: verb at 0001000
Line 5:	SCAN च	⇒ p4: conj at 0010000
Line 6:	SCAN नलस्	⇒ p5: nom at 0100000
Line 7:	SCAN अवदत्	⇒ p6: verb at 1000000

For instance, line 1 was created by scanning the input word रुचिरम्. The resulting edge, number p1, provides the category *adj* and covers the leftmost word of the input (0000001).

The active chart is seeded with an initial edge as well.

Edge a0: $\$ \to$ s at $\underline{0000000}$
[1]:0000000 $\to$ s

Here, *s* is the start category of the grammar. The agenda is then jumpstarted by scheduling prediction from this edge.

In this prediction step, each of the rules that can generate this symbol are considered in turn. There are three: rules (82b), (82c), and (82d). Since a0 has active bits in its p-mask, the prediction step must make sure that each rule has a maximum yield no less than the number of active bits in the p-mask (or else the categories generated by that rule could not hope to satisfy the p-mask). Here, rule (82c), which has a maximum yield of two, fails that test; the other two rules generate a1 and a2, respectively.

Line 8: PRED S in a0
Rule (82c) would generate too few categories.
Edge a1: s $\to$ verb$_2$, nom$_3$, acc$_4$ at $\underline{0000000}$
c3 < c2, c4 < c2
Edge a2: s $\to$ conj$_2$, s$_3$, s$_4$ at $\underline{0000000}$
c3 $\ll$ c2, c2 $\ll$ c4, [3]:0000000 $\to$ s, [4]:0000000 $\to$ s

All edges generated by prediction have an empty (all zeroes) coverage vector. Since the trigger a0 has only one category on its RHS, its p-mask propagates to a1 and a2. As each edge is entered into the chart, the parser schedules the prediction of that rule as well as its completion with the edges that generate its active category. Here, a1 has the active category *verb*, so it is scheduled to complete with p6 and p3. (Recall that the search for compatible passive edges is done at this point so as to ensure that the edges found are older than the current edge – subsequently-created passive edges will search for active edges to complete with. If the parser waited until dequeueing this action from the agenda to search for compatible edges, it might pick up edges that had already been scheduled to complete with the active edge.) Edge a2 has active category *conj* and is scheduled to complete with p4.

Prediction from a1 is vacuous, since no grammar rules provide *verb*. Thus the next task dequeued from the parser's agenda is to complete a1 with p6.

Line 9: COMP a1 with p6
Edge a3: s $\to$ nom$_3$, acc$_4$ at $1\underline{000000}$
c3 < 1000000, c4 < 1000000

The differences between this edge and a1 illustrate the process of completion: *verb*$_1$ has been removed from the list of daughters and the constraint c3 < c2 has been updated to c2 < 1000000 (representing the fact that category 2 has been found at position 1000000). The p-mask is unchanged from a1; completion never changes the p-mask. As before, this edge is scheduled for prediction, and completion with p5 and p1, the edges that currently provide *nom*.

The parser now completes a1 with p4, the other passive edge providing *verb*.

Line 10: COMP a1 with p3
Edge a4: s $\to$ nom$_3$, acc$_4$ at $\underline{000}1\underline{000}$
c3 < 0001000, c4 < 0001000

The only difference here from a3 is in the position of the located constituent; this edge too will be scheduled for prediction and completion with p5 and p1.

Prediction from a2 is also vacuous; the parser moves on to its next task: completing a2 with p4.

Line 11: COMP a2 with p4

Edge a5: $s \rightarrow s_3, s_4$ at $\underline{00}1\underline{0000}$

c3 $\ll$ 0010000, 0010000 $\ll$ c4, [3]:0000000 $\rightarrow$ s, [4]:0000000 $\rightarrow$ s

As no passive edges currently provide *s*, this edge is only scheduled for prediction.

The next substantive tasks are the completion of a3 with p5 and p1.

Line 12: COMP a3 with p5

Edge a6: $s \rightarrow acc_4$ at $11\underline{00000}$

c4 < 1000000

Line 13: COMP a3 with p1

Edge a7: $s \rightarrow acc_4$ at $1\underline{0000}1\underline{0}$

c4 < 1000000

As before, the located element is removed from the edge's RHS. In addition, as the constraint c3 < 1000000 is updated (to 0100000 < 1000000 and 0000010 < 1000000), the parser notes that the constraint is now complete (in the sense that no further information can be received that would update it) and inviolate (as 0100000/0000010 really do precede 1000000). Each edge is then scheduled for prediction and completion with p2.

This process is then repeated with respect to a4.

Line 14: COMP a4 with p5

Passive coverage 0100000 does not precede 0001000

Line 15: COMP a4 with p1

Edge a8: $s \rightarrow acc_4$ at $\underline{000}1\underline{0}1\underline{0}$

c4 < 0001000

According to our grammar, verbs are sentence-final. This prevents the *nom* in sixth position from being a part of a sentence containing the verb in fourth position. The *nom* in second position is acceptable, though, and edge a8 is formed. Like a6 and a7, it is scheduled for prediction and completion with p2.

The parser has now reached the scheduled prediction of *s* from a5.

Line 16: PRED S in a0

Edge a9: $s \rightarrow verb_2, nom_3$ at ~~000~~$\underline{0}$000

c3 < c2

Edge a10: $s \rightarrow verb_2, nom_3, acc_4$ at ~~000~~$\underline{0}$000

c3 < c2, c4 < c2

Rule (82b) would generate too many categories.

Here, the parser is predicting the left-hand *s* generated by the conjunction rule. Since the parser has already located the conjunction in fifth position, the fifth through seventh positions are masked out. This prevents prediction with rule (82b), which has a minimum yield of five, more than could fit in the remaining space. The immediate precedence constraint involving the left-hand *s* in a5 requires the fourth position to be covered.

The parser now turns its attention to the tasks generated by the creation of a6. First, the parser predicts *acc* in a6.

Line 17: PRED acc in a6

Edge a11: $acc \rightarrow adj_2, acc_3$ at ~~00~~$\underline{00000}$

$\emptyset$

This edge inherits its n-mask from a6's coverage; it also gets its p-mask from the noncovered portions of a6's pmask.

The next task is completion with p2.

Line 18: comp a6 with p2

Proposed passive edge does not obey p-mask.

This completion would produce an edge with coverage 1100010, which does not obey a p-mask of 1111111.

The tasks generated by the creation of a7 and a8 are handled in much the same way.

Line 19: pred acc in a7

Edge a12: acc → adj_2, acc_3 at $\cancel{0}\underline{00000}\cancel{0}\underline{0}$

∅

Line 20: comp a7 with p2

Proposed passive edge does not obey p-mask.

Line 21: pred acc in a8

Negative mask 1111010 overlaps positive mask 1110101.

Line 22: comp a8 with p2

Proposed passive edge does not obey p-mask.

Line 21 illustrates another aspect of the mask creation process. Sometimes, the requirements on an edge about to be created by prediction conflict. In this case, the edge's p-mask must include all non-covered bits of a8's p-mask (since a8 has a singleton RHS), leading to a p-mask of 1110101; in addition, the category in question is constrained to precede position four and cannot overlap with the positions of the already-located constituents, which leads to its n-mask of 1111010. For the edge to be successfully completed, then, it would have to simultaneously occupy and not occupy positions five, six, and seven; the edge is therefore not created.

The parser continues in this fashion, and I will only comment on those parsing steps exemplifying new aspects of the algorithm.

Line 23: comp a9 with p6

Passive coverage 1000000 overlaps with active n-mask 1110000

Line 24: comp a9 with p3

Edge a13: s → nom_3 at $\cancel{0}\cancel{0}\cancel{0}1000$

$c3 < 0001000$

Line 25: comp a10 with p6

Passive coverage 1000000 overlaps with active n-mask 1110000

Line 26: comp a10 with p3

Edge a14: s → nom_3, acc_4 at $\cancel{0}\cancel{0}\cancel{0}1000$

$c3 < 0001000$, $c4 < 0001000$

Line 27: comp a11 with p0

Edge a15: acc → acc_3 at $\cancel{0}\cancel{0}\underline{0000}1$

∅

Line 28: comp a12 with p0

Edge a16: acc → acc_3 at $\cancel{0}\underline{0000}\cancel{0}1$

∅

Line 29: comp a13 with p5

Passive coverage 0010000 overlaps with active n-mask 1111000

Line 30: comp a13 with p1 ⇒ p7: s at 0001010

At this point, a new passive edge is created. It is scheduled to complete with each active edge in its backpointer class; here, that is only a5.

Line 31: comp a14 with p5

Passive coverage 0010000 overlaps with active n-mask 1111000

Line 32: COMP a14 with p1
Edge a17: s → acc$_4$ at ~~000~~1010
c4 < 0001000
Line 33: PRED acc in a15
Edge a18: acc → adj$_2$, acc$_3$ at ~~00~~$\underline{0000}$~~0~~
∅
Line 34: COMP a15 with p2
Proposed passive edge does not obey p-mask.
Line 35: PRED acc in a16
Edge a19: acc → adj$_2$, acc$_3$ at ~~0~~$\underline{0000}$~~00~~
∅
Line 36: COMP a16 with p2
Proposed passive edge does not obey p-mask.
Line 37: COMP a5 with p7
Coverage vector 0001010 is not contiguous.

Edge a5 has a domain declaration for each of its *s* daughters; as a result, p7 is unsuitable as a completion candidate for that edge.

Line 38: PRED acc in a17
Edge a20: acc → adj$_2$, acc$_3$ at ~~000000~~0
∅
Line 39: COMP a17 with p2 ⇒ p8: s at 0001110
Line 40: COMP a18 with p0
Passive coverage 0000001 overlaps with active n-mask 1100001.
Line 41: COMP a19 with p0
Passive coverage 0000001 overlaps with active n-mask 1000011.
Line 42: COMP a20 with p0
Edge a21: acc → acc$_3$ at ~~000000~~1
∅
Line 43: COMP a5 with p8
Edge a22: s → s$_4$ at $\underline{00}$11111$\underline{0}$
0010000 ≪ c4, [4]:0000000 → s
Line 44: PRED acc in a21
Rule (82e) would generate too many categories.
Line 45: COMP a21 with p2 ⇒ p9: acc at 0000101
Line 46: PRED s in a22
Negative mask 0011111 overlaps positive mask 1100001.
Negative mask 0011111 overlaps positive mask 1100001.
Negative mask 0011111 overlaps positive mask 1100001.
Line 47: COMP a22 with p8
Passive coverage 0001110 overlaps with active n-mask 0011110.
Line 48: COMP a22 with p7
Passive coverage 0001010 overlaps with active n-mask 0011110.
Line 49: COMP a17 with p9 ⇒ p10: s at 0001111
Line 50: COMP a5 with p10
Edge a23: s → s$_4$ at $\underline{00}$11111
0010000 ≪ c4, [4]:0000000 → s
Line 51: PRED s in a23

Edge a24: s → verb$_2$, nom$_3$ at $\underline{00}$~~00000~~
c3 < c2
Rule (82b) would generate too many categories.
Rule (82d) would generate too many categories.
Line 52: comp a23 with p8
Passive coverage 0001110 overlaps with active n-mask 0011111.
Line 53: comp a23 with p10
Passive coverage 0001111 overlaps with active n-mask 0011111.
Line 54: comp a23 with p7
Passive coverage 0001010 overlaps with active n-mask 0011111.
Line 55: comp a24 with p6
Edge a25: s → nom$_3$ at 1$\underline{0}$~~00000~~
c3 < 1000000
Line 56: comp a24 with p3
Passive coverage 0000100 overlaps with active n-mask 0011111.
Line 57: comp a25 with p5 ⇒ p11: s at 1100000
Line 58: comp a25 with p1
Passive coverage 0000010 overlaps with active n-mask 1011111.
Line 59: comp a23 with p11 ⇒ p12: s at 1111111
Line 60: comp a0 with p12 ⇒ p13: $ at 1111111

At this point, the agenda is empty. The parser looks for passive edges providing the $ category whose coverage extends over the entire string, and finds this in edge p13. As a result, the parse is deemed successful.

5.3.2 Domains and Dormant Constraints

This sample parse uses the grammar in (84) and the input sentence **fegefhi**.

(84) a) root(a, [g ≪ c, d < x]).
b) a → b$_1$, c$_2$, g$_3$; 1 < 2
c) b → d$_1$
d) c → h$_1$, d$_2$, i$_3$; ; ⟨{2}, [e < f], d⟩, ⟨{1, 3}, [h < i], x⟩
e) d → e$_1$, f$_2$

This grammar has been written[7] to exercise the domain-level constraint handling abilities of the parser; it is a more complicated version of (73). Most importantly, the partial compaction effected by the domain declaration in (84d) will require a constraint merger. Here, domain constraints will be listed on a separate line; active domain constraints are displayed in **bold**.

Line 1: scan f
⇒ p0: f at 0000001 **g** ≪ **c**, **d** < **x**, e < f, h < i
Line 2: scan e
⇒ p1: e at 0000010 **g** ≪ **c**, **d** < **x**, e < f, h < i
Line 3: scan g
⇒ p2: g at 0000100 **g** ≪ **c**, **d** < **x**, e < f, h < i
Line 4: scan e

[7]It is not possible to construct a compact linguistically-reminiscent grammar that exercises the same features of the parser in the same amount of space, and thus a non-linguistic grammar is used in this section.

⇒ p3: e at 0001000 $\mathbf{g} \ll \mathbf{c}$, $\mathbf{d} < \mathbf{x}$, e < f, h < i

Line 5: SCAN f

⇒ p4: f at 0010000 $\mathbf{g} \ll \mathbf{c}$, $\mathbf{d} < \mathbf{x}$, e < f, h < i

Line 6: SCAN h

⇒ p5: h at 0100000 $\mathbf{g} \ll \mathbf{c}$, $\mathbf{d} < \mathbf{x}$, e < f, h < i

Line 7: SCAN i

⇒ p6: i at 1000000 $\mathbf{g} \ll \mathbf{c}$, $\mathbf{d} < \mathbf{x}$, e < f, h < i

All edges must contain some set of domain constraints, but as lexical items never trigger prediction steps, the choice of domain is irrelevant. As a result, lexical items are arbitrarily assumed to be in the root domain.

The initial edge is as follows:

Edge a0: $\$ \rightarrow a_1$ at $\underline{0000000}$
[1]:0000000 → a : $\mathbf{g} \ll \mathbf{c}$, $\mathbf{d} < \mathbf{x}$, e < f, h < i
$\mathbf{g} \ll \mathbf{c}$, $\mathbf{d} < \mathbf{x}$, e < f, h < i

Here, the third line displays the current state of the constraints within the root domain, while the second line indicates the current state of the constraints with the subordinate domain that will encompass the daughter category of this rule. (Recall that this separation is how the principle of domain locality is enforced: as elements are found, only the constraints in that domain within which the element is visible will be updated.)

The initial predictions are generated in much the same way as in the previous example.

Line 8: PRED a in a0
Edge a1: $a \rightarrow b_2, c_3, g_4$ at $\underline{0000000}$
c2 < c3
$\mathbf{g} \ll \mathbf{c}$, $\mathbf{d} < \mathbf{x}$, e < f, h < i

Line 9: PRED b in a1
Edge a2: $b \rightarrow d_2$ at 0000000
$\emptyset$
$\mathbf{g} \ll \mathbf{c}$, $\mathbf{d} < \mathbf{x}$, e < f, h < i

Line 10: PRED d in a2
Edge a3: $d \rightarrow e_2, f_3$ at 0000000
$\emptyset$
$\mathbf{g} \ll \mathbf{c}$, $\mathbf{d} < \mathbf{x}$, e < f, h < i

Initially, a1 gets its domain constraints from the initial state of the domain declaration on a0. Thereafter, since each prediction remains in the same domain, the constraints are passed on.

Line 11: COMP a3 with p3
Edge a4: $d \rightarrow f_3$ at 0001000
$\emptyset$
$\mathbf{g} \ll \mathbf{c}$, $\mathbf{d} < \mathbf{x}$, e:0001000 < f, h < i

Line 12: COMP a3 with p1
Edge a5: $d \rightarrow f_3$ at 0000010
$\emptyset$
$\mathbf{g} \ll \mathbf{c}$, $\mathbf{d} < \mathbf{x}$, e:0000010 < f, h < i

Having located an *e*, the relevant dormant constraints on a4 and a5 are updated. This will ensure that, if this edge or one of its descendants is later completed into the domain where that constraint is relevant, the constraint can be properly evaluated.

Line 13: comp a4 with p4
⇒ p7: d at 0011000
g ≪ **c**, **d** < **x**, e:0001000 < f:0010000, h < i
Line 14: comp a4 with p0
⇒ p8: d at 0001001
g ≪ **c**, **d** < **x**, void, h < i

The contrast between p7 and p8 illustrates the impact of dormant constraints. On p7, the *f* at a position that follows 0001000, and the constraint is represented. (In contrast to token-based constraints, description-based constraints can never be removed from an edge, as another category matching the description might yet be encountered.) On p8, however, the *f* is found preceding 0001000, and the constraint is violated. This is not fatal, however, since it is only a dormant constraint. (As the parse continues, notice that the void constraint is propagated to descendent edge p12 and a7.)

Line 15: comp a5 with p4
⇒ p9: d at 0010010
g ≪ **c**, **d** < **x**, e:0000010 < f:0010000, h < i
Line 16: comp a5 with p0
⇒ p10: d at 0000011
g ≪ **c**, **d** < **x**, void, h < i
Line 17: comp a2 with p7
⇒ p11: b at 0011000
g ≪ **c**, **d:0011000** < **x**, e:0001000 < f:0010000, h < i
Line 18: comp a2 with p8
⇒ p12: b at 0001001
g ≪ **c**, **d:0001001** < **x**, void, h < i
Line 19: comp a2 with p9
⇒ p13: b at 0010010
g ≪ **c**, **d:0010010** < **x**, e:0000010 < f:0010000, h < i
Line 20: comp a2 with p10
⇒ p14: b at 0000011
g ≪ **c**, **d:0000011** < **x**, void, h < i
Line 21: comp a1 with p11
Edge a6: a → c_3, g_4 at $\underline{00}11\underline{000}$
0011000 < c3
g ≪ **c**, **d:0011000** < **x**, e:0001000 < f:0010000, h < i
Line 22: comp a1 with p12
Edge a7: a → c_3, g_4 at $\underline{000}1\underline{00}1$
0001001 < c3
g ≪ **c**, **d:0001001** < **x**, void, h < i
Line 23: comp a1 with p13
Edge a8: a → c_3, g_4 at $\underline{00}1\underline{00}1\underline{0}$
0010010 < c3
g ≪ **c**, **d:0010010** < **x**, e:0000010 < f:0010000, h < i
Line 24: comp a1 with p14
Edge a9: a → c_3, g_4 at $\underline{0000}11$
0000011 < c3
g ≪ **c**, **d:0000011** < **x**, void, h < i
Line 25: pred c in a6

Rule (84d) would generate too many categories.

Line 26: PRED c in a7

Rule (84d) would generate too many categories.

Line 27: PRED c in a8

Rule (84d) would generate too many categories.

Line 28: PRED c in a9

Edge a10: c → h_2, d_3, i_4 at $\underline{00000}11$

[3]:0000000 → d : g ≪ c, d < x, **e** < **f**, h < i,

[2, 4]:0000000 → x : g ≪ c, d < x, e < f, **h** < **i**

g ≪ **c**, **d:0000011** < **x**, void, h < i

Note that line 28 makes reference to all three domains in this grammar simultaneously. In particular, note that the second domain will be recognized partially before and after the first, since the parser is told to look for the daughters in the order *h*, *d*, and then *i*.

Line 29: COMP a10 with p5

Edge a11: c → d_3, i_4 at 01000~~00~~

[3]:0000000 → d : g ≪ c, d < x, **e** < **f**, h < i,

[4]:0100000 → x : g ≪ c, d < x, e < f, **h:0100000** < **i**

g ≪ **c**, **d:0000011** < **x**, void, h < i

Here, since the *h* is included in the span of the second domain declaration, only the constraints in that domain are updated with the fact that an *h* was found at 0100000 – the other two domains do not receive this information. The parser also updates the coverage of the second domain declaration; this bitvector will have to have a contiguous set of active bits once all categories in the statement's span have been found.

Line 30: PRED d in a11

Edge a12: d → e_2, f_3 at 0~~0~~000~~00~~

∅

g ≪ c, d < x, **e** < **f**, h < i

Line 31: COMP a11 with p8

Passive coverage 0001001 overlaps with active n-mask 0100011.

Line 32: COMP a11 with p9

Passive coverage 0010010 overlaps with active n-mask 0100011.

Line 33: COMP a11 with p10

Passive coverage 0000011 overlaps with active n-mask 0100011.

Line 34: COMP a11 with p7

Edge a13: c → i_4 at 01110~~00~~

[4]:0100000 → x : g ≪ c, d < x, e < f, **h:0100000** < **i**

g ≪ **c**, **d:0011000** < **x**, void, h < i

In line 34, a *d* is found, completing one of the domains on this edge. The domain declaration itself is removed from the edge, and the edge's domain constraints are updated with the fact that a *d* (the domain declaration's result category) was found at location 0011000. Since this is further to the right than the existing frontier of 0000011, the constraint is updated accordingly.

Line 35: COMP a12 with p3

Edge a14: d → f_3 at 0~~0~~010~~00~~

∅

g ≪ c, d < x, **e:0001000** < **f**, h < i

Line 36: COMP a12 with p1
 Passive coverage 0000010 overlaps with active n-mask 0100011.
Line 37: COMP a13 with p6
 ⇒ p15: c at 1111000
  g $\ll$ **c**, **d:0011000** < **x:1100000**, void, h < i
Line 38: COMP a14 with p4
 ⇒ p16: d at 0011000
  g $\ll$ c, d < x, **e:0001000** < **f:0010000**, h < i
Line 39: COMP a14 with p0
 Passive coverage 0000001 overlaps with active n-mask 0101011.
Line 40: COMP a9 with p15
 Edge a15: a → g_4 at $1111\underline{0}11$
  $\emptyset$
  g $\ll$ **c:1111000**, **d:0011000** < **x:1100000**, void, h < i
Line 41: COMP a11 with p16
 Result blocked by edge a13.
Line 42: COMP a15 with p2
 ⇒ p17: a at 1111111
  g:0000100 $\ll$ **c:1111000**, **d:0011000** < **x:1100000**, void, h < i
Line 43: COMP a0 with p17
 ⇒ p18: $ at 1111111
  g $\ll$ **c**, **d** < **x**, e < f, h < i

Here, edge p18 is the marker of success.

5.4 Conclusion

The parsing algorithm that has been presented in this section applies to GIDLP grammars written to use atomic categories. When compared to a standard context-free Earley parser, its key features are as follows:

- edge coverage representation is set-based, rather than interval-based
- the same data structure represents edge coverage as well as LP constraint state
- a single dot suffices even in the presence of discontinuous constituency
- dormant constraints can serve to model the effect of word order domains

The next chapter will turn to the issues brought up by adapting this algorithm to deal with feature structure categories.

CHAPTER 6

Parsing GIDLP Grammars With Feature Structure Categories

Having seen the operation of the atomic GIDLP parsing algorithm, this chapter turns to the issues involved in moving to feature structure categories, which is needed to truly support linearization-HPSG grammars. Fortunately, many researchers have worked on parsing with feature structures in the context of unification-based parsing; this work will be discussed in section 6.1. Similar work exists for the treatment of feature structure-based LP constraints and non-local information flow, which has been studied in connection with direct ID/LP parsing; this is discussed in section 6.2. Section 6.3 presents the algorithm that has been developed for parsing GIDLP categories with feature structures, discussing the integration of these existing technologies into the previous algorithm.

6.1 Unification-Based Parsing

It is initially helpful to review the basic framework that underlay much of the original work on parsing with feature structure categories. Consider first the sample grammar (inspired by Shieber (1985)) defined over untyped feature structures given in Figure 6.1.

Such a grammar has three parts:

1. A set of feature structures (here, *Rule 1* and *Rule 2*) that form the rules of the grammar, describing how the mother node of a local tree can be related to its daughters. In particular, the X_0 attribute's value is the parent feature structure, corresponding to the left-hand side of a CFG rule, and the remaining attributes' values are the feature structures for the daughters, corresponding in order to the right-side of a CFG rule.
2. A set of feature structures (here, *Lexeme 1*) representing lexical entries. These typically have one attribute whose value can be matched against the input (that is, some sort of phonological or orthographical representation) and other attributes that describe the lexical item itself, depending on the intended use of the grammar and the underlying linguistic theory.
3. A feature structure providing the start category (here, *Start*). This plays an analagous role to the start symbol in a context-free grammar; it is a constraint on the root of the parse tree.

Consider the tree given in Figure 6.2. Since the tree's root is subsumed by the 'Start' feature structure, each local tree (seen as a single feature structure) is subsumed by a rule in the grammar, and each leaf is subsumed by a lexical feature structure, the parse tree is a valid parse of *aaa*.

$$\text{Rule 1}\ \begin{bmatrix} \textsc{x}_0 & \begin{bmatrix} \textsc{cat} & t \\ \textsc{f} & \boxed{1} \end{bmatrix} \\ \textsc{x}_1 & \begin{bmatrix} \textsc{cat} & t \\ \textsc{f}\,|\,\textsc{f} & \boxed{1} \end{bmatrix} \\ \textsc{x}_2 & \begin{bmatrix} \textsc{cat} & a \end{bmatrix} \end{bmatrix} \quad \text{Rule 2}\ \begin{bmatrix} \textsc{x}_0 & \begin{bmatrix} \textsc{cat} & t \end{bmatrix} \\ \textsc{x}_1 & \begin{bmatrix} \textsc{cat} & a \end{bmatrix} \end{bmatrix}$$

$$\text{Lexeme 1}\ \begin{bmatrix} \textsc{cat} & a \\ \textsc{phon} & \text{‘}a\text{’} \end{bmatrix} \quad \text{Start}\ \begin{bmatrix} \textsc{cat} & t \\ \textsc{f} & a \end{bmatrix}$$

Figure 6.1: Example feature structure grammar

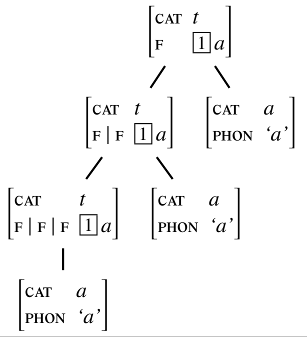


Figure 6.2: Parse tree for string *aaa*

Just as with ID/LP parsing (as discussed in section 2.1.3), one can talk about direct or indirect unification-based parsing. Indirect unification-based parsing involves rewriting or expanding the feature-structure grammar into another type of grammar (for instance, a CFG), and then applying well-known parsing methods to that grammar. Direct unification-based parsing, on the other hand, involves algorithms that work directly with feature structures.

Since a feature-structure grammar like that illustrated in Figure 6.1 can easily be seen as an augmentation of a context-free grammar, much of the early work on direct feature structure parsing focused on methods of converting context-free parsing algorithms into feature structure parsing algorithms. One of the most obvious aspects of such a conversion is the need to move from matching via symbol equality to matching via unifiability. The predictor now looks for rules in the grammar whose left-hand symbol can unify with the active category in the edge being predicted; and the completer looks for passive edges whose left-hand categories can unify with the active category of the current edge.

6.1.1 Restriction

6.1.1.1 Restriction To Ensure Termination

In a standard context-free chart parser, the predictor uses an identity test to avoid adding redundant edges to the chart. When an identical edge already exists in the chart, it does not add that edge to the chart again. This is necessary to prevent loops in the prediction process, which would otherwise cause the algorithm not to terminate on left-recursive grammars. Following Pereira and Warren (1983), a feature-structure chart parser must test for *subsumption* instead: a feature-structure predictor should not add an edge to the chart when a subsuming feature-structure is already present in the chart.

It has also been assumed that a subsumption check is also needed during completion to prevent redundant edges from being added to the chart. Erbach (1997) points out that this is only the case if the grammar is capable of generating identical edges from distinct completion steps. In order for this to happen, the grammar must contain structural ambiguities, or else it will be impossible for the completion step to yield a redundant edge. Additionally, the grammar must not incorporate the parse tree into the general linguistic data structure (as, for instance, HPSG's DTRS feature does), or else the edges corresponding to each possible structure of an ambiguous string will be distinct. If either of these conditions fails to hold, the subsumption check can be dispensed with during completion.

Shieber (1985) shows, however, that a subsumption check during prediction is not enough to guarantee termination when parsing with an infinite set of nonterminals (as can arise with grammars involving first order terms or feature structures). Consider how the parse tree in 6.2 could have been constructed. The most straightforward adaptation of Earley's algorithm to feature structure parsing involves a prediction step as follows: Given an edge with active category index i such that the value of x_n is A, and a rule whose x_0 value is B, if the unification $C = [\mathrm{x}_n A] \sqcup B$ exists, add C as a new edge to the chart. Thus, given a start symbol of $\begin{bmatrix}\text{CAT} & t\end{bmatrix}$, the initial prediction step yields (85).

(85)
$$\begin{bmatrix} \mathrm{x}_0 & \begin{bmatrix}\text{CAT} & t\\ \text{F} & \boxed{1}\end{bmatrix}\\ \mathrm{x}_1 & \begin{bmatrix}\text{CAT} & t\\ \text{F}\,|\,\text{F} & \boxed{1}\end{bmatrix}\\ \mathrm{x}_2 & \begin{bmatrix}\text{CAT} & a\end{bmatrix}\end{bmatrix}$$

Prediction (which unifies x_1 of the triggering edge with x_0 of the rule) now applies to this edge to yield (86); and in turn (87).

(86)
$$\begin{bmatrix} \mathrm{x}_0 & \begin{bmatrix}\text{CAT} & t\\ \text{F}\,|\,\text{F} & \boxed{1}\end{bmatrix}\\ \mathrm{x}_1 & \begin{bmatrix}\text{CAT} & t\\ \text{F}\,|\,\text{F}\,|\,\text{F} & \boxed{1}\end{bmatrix}\\ \mathrm{x}_2 & \begin{bmatrix}\text{CAT} & a\end{bmatrix}\end{bmatrix}$$

(87)
$$\begin{bmatrix} \mathrm{x}_0 & \begin{bmatrix}\text{CAT} & t\\ \text{F}\,|\,\text{F}\,|\,\text{F} & \boxed{1}\end{bmatrix}\\ \mathrm{x}_1 & \begin{bmatrix}\text{CAT} & t\\ \text{F}\,|\,\text{F}\,|\,\text{F}\,|\,\text{F} & \boxed{1}\end{bmatrix}\\ \mathrm{x}_2 & \begin{bmatrix}\text{CAT} & a\end{bmatrix}\end{bmatrix}$$

There is no point at which any of (85), (86), or (87) subsumes a previous edge, and it is clear that this pattern will never terminate. To restore the termination property, Shieber proposed that the prediction step be modified to use *restriction*.

Restriction involves a function (a *restrictor*) over feature structures. While a restrictor could be stated generally – a limit on embedding, for instance, a restrictor is usually represented by a set of *paths* – series of features in the feature structure, written as a list of attribute labels separated with vertical bars. This kind of restrictor, which must necessarily be grammar-specific, takes an arbitrary feature structure as input and returns a *restricted feature structure* (RFS) consisting of the restrictor's paths and their values. In effect, the RFS expresses a subset of constraints expressed by the original feature structure. Following Harrison and Ellison (1992), such a restrictor is termed a *positive* restrictor, since it specifies which paths to keep in the RFS. The set of paths could also be interpreted as enumerating those paths to remove in creating the RFS; this interpretation yields a *negative* restrictor. For example, applying the restrictor in (88) to (89) in the positive sense yields (90) and in the negative sense yields (91).

(88) $\langle \text{A}|\text{B},\ \text{D}|\text{E}|\text{F},\ \text{D}|\text{I}|\text{J}|\text{F}\rangle$

(89)
$$\begin{bmatrix} \text{A} & \begin{bmatrix}\text{B} & c\end{bmatrix}\\ \text{D} & \begin{bmatrix}\text{E} & \boxed{1}\begin{bmatrix}\text{F} & \begin{bmatrix}\text{G} & h\end{bmatrix}\end{bmatrix}\\ \text{I} & \begin{bmatrix}\text{J} & \boxed{1}\end{bmatrix}\\ \text{K} & l\end{bmatrix}\end{bmatrix}$$

(90)
$$\begin{bmatrix} \text{A} & \begin{bmatrix}\text{B} & c\end{bmatrix}\\ \text{D} & \begin{bmatrix}\text{E} & \boxed{1}\begin{bmatrix}\text{F} & g\end{bmatrix}\\ \text{I} & \begin{bmatrix}\text{J} & \boxed{1}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

(91) $\begin{bmatrix} \text{A} & b \\ \text{D} & \begin{bmatrix} \text{E} & f \\ \text{I} & \begin{bmatrix} \text{J} & f \end{bmatrix} \\ \text{K} & l \end{bmatrix} \end{bmatrix}$

Shieber then modifies prediction to use restriction as follows: As before, we are given an edge with active category index i such that the value of x_n is A and a rule whose x_0 value is B. Let $A' = A \upharpoonright R$, where R is a restrictor. Then, if the unification $C = [\mathrm{x}_n A'] \sqcup B$ exists, add C as a new edge to the chart. Shieber observes that since there are a finite number of RFSs, restriction has the effect of dividing the infinite feature structure space into a finite number of equivalence classes. As a result, the prediction step will never loop indefinitely. In the grammar given in Figure 6.1, for instance, the restrictor can only contain a finite number of paths, and eventually the parse will reach a point where prediction yields an edge already in the chart.

As Shieber points out, it should be kept in mind that the point of the prediction step is to provide top-down guidance to the parse: restriction deletes some of that information in order to guarantee termination. This never affects the correctness of the parse, since the RFS is more general than the original feature structure. The key is then to find the combination of paths for a particular grammar that maximize the quality of top-down guidance given while still ensuring termination.

6.1.1.2 Restriction as Optimization

Gerdemann (1991) expands the role of restriction in Earley-inspired feature structure parsing by pointing out several additional aspects of the algorithm where restriction can aid efficiency. In particular, it has often been observed (see, for example, Tomabechi 1995) that more time is spent unifying feature structures than performing any other part of the parse. Unification is, in a sense, used as both a boolean predicate (could these two feature structures be descriptions of the same object?) and as a function (what is the most general feature structure that describes such an object?); the latter's domain is those pairs of feature structures for whom the answer to the former was 'yes'. Gerdemann's insight was that the first question can often be answered by considering a restriction of each feature structure: if the two RFSs are incompatible, then there's no way that the full feature structures could have been compatible.

As a result, Gerdemann modifies Shieber's predictor to work as follows: a *predictors list* is kept of all RFSs used at a given position for prediction. Then during prediction, the active category's RFS is checked to see if it is subsumed by anything on the list. If so, the prediction step can immediately terminate – any edge generated by the subsumed edge would necessarily be subsumed by the edges generated by the earlier prediction and hence fail to be entered into the chart. For example, if the RFS $\begin{bmatrix} \text{CAT} & np \end{bmatrix}$ has been used for prediction, there is no need to try to predict with the RFS $\begin{bmatrix} \text{CAT} & np \\ \text{NUM} & sing \end{bmatrix}$.

A second use of restriction to optimize the parsing process occurs as a part of the completion step. In Shieber's parsing algorithm, completion involves attempting to unify the value of the x_0 feature on the passive edge with the value of the active category feature of each appropriate active edge. For Shieber, 'appropriate' means that the two edges are adjacent. In practice, however, Gerdemann observes that completion of a given passive edge only need be attempted with the active edges that are ancestors (in the sense of the parser's search space) of the passive edge, **or** those that could have been, were it not for the subsumption test that prevents entry of redundant edges.

Gerdemann therefore adds a forward pointer and a backpointer to each edge: during each prediction step, the RFS that triggered prediction is noted both on the newly-created edge as the backpointer and on the triggering edge as the forward pointer. When an edge is blocked from triggering prediction (as explained earlier in this section), the RFS from the predictors list that was responsible for the blocking is stored as the forward pointer. Then during completion, unification attempts can be further limited to those active edges whose forward pointers match the backpointer of the passive edge.

6.1.1.3 Dangers of Improper Restriction

Harrison and Ellison (1992) show that with certain grammars, a poorly-chosen Shieber-style restrictor will cause the parser to fail to terminate. They illustrate this with the grammar in Figure 6.3 (by convention, the LHS of Rule 1 is the start symbol; note that *Rule 3* is an epsilon rule), which can recognize either of the strings *a* or *b*; the choice determines the depth of the *t* recursion.

This is illustrated in Figure 6.4. Consider the case where *a* is being parsed. Rule 1 is predicted, and the resulting edge is completed with Lexeme 1. Predicting the *t* from the result of the completion will generate the edge (92) – in which the information about the (finite) length of the G path has been lost, due to the effects of the restrictor.

(92) $$\begin{bmatrix} \text{X}_0 & \begin{bmatrix} \textsc{cat} & t \\ \textsc{g} \mid \textsc{g} & \boxed{1} \end{bmatrix} \\ \text{X}_1 & \begin{bmatrix} \textsc{cat} & t \\ \textsc{g} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

This edge then completes with the empty category admitted by Rule 3 to yield the edge (93).

(93) $$\begin{bmatrix} \text{X}_0 & \begin{bmatrix} \textsc{cat} & t \\ \textsc{g} \mid \textsc{g} & \boxed{1} \end{bmatrix} \\ \text{X}_1 & \begin{bmatrix} \textsc{cat} & t \\ \textsc{g} & \boxed{1}\, e \end{bmatrix} \end{bmatrix}$$

This edge is passive, and can again be completed with (92) to yield (94); this process repeats indefinitely.

(94) $$\begin{bmatrix} \text{X}_0 & \begin{bmatrix} \textsc{cat} & t \\ \textsc{g} \mid \textsc{g} & \boxed{1} \end{bmatrix} \\ \text{X}_1 & \begin{bmatrix} \textsc{cat} & t \\ \textsc{g} & \boxed{1} \begin{bmatrix} \textsc{g} & e \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Harrison and Ellison observe that this is not a fatal flaw with restriction in general and merely indicates a need for caution in developing the restrictor for a given grammar. Here, just changing the restrictor to include $\langle \textsc{g}|\textsc{g}|\textsc{g} \rangle$ will allow parses with this grammar to terminate.

6.1.2 Quick Check Vectors

An idea similar to restriction is proposed by Malouf et al. (2000), who observe that no formal requirements give the order in which the subunits of a feature structure should be processed during unification. As a result, it makes sense to first check those arcs which are statistically most likely to fail first, so that an unsuccessful unification can be detected as quickly as possible. (This is the same insight that the GIDLP parsing algorithm applies to the RHS order of an ID rule; see section 7.1.2.) Malouf et al. further point out that it is possible to extract this statistical information from a suitably-instrumented parser. If one is

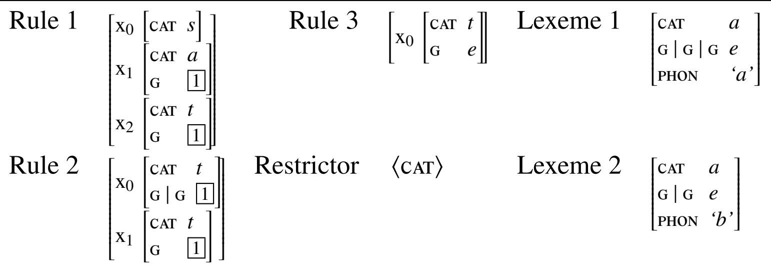


Figure 6.3: Harrison and Ellison's grammar

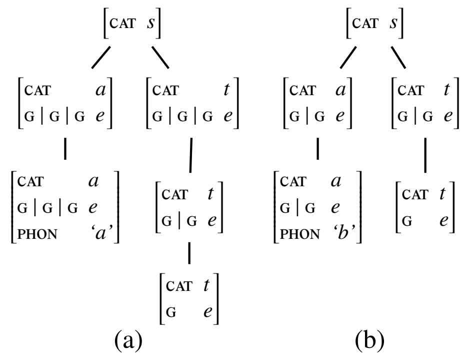


Figure 6.4: Parse trees for strings *a* and *b*

in possession of a suitably-broad testsuite for a grammar, one can have a parser record the path at which unification failed during every step of the parse.

At the end of this process, the ideal unification order can be determined and potentially built into the feature structure representation. As a less invasive alternative, the *n* most common failing paths can be used to create a restrictor that only makes note of the types of the nodes at the end of each of its paths (rather than the entire substructure at those points, as a Shieber-style restrictor does). The resulting *quick check vector* can then be used as a pre-unification filter just as a Shieber-style restrictor can.

6.2 Non-Local Information Flow

The initial work on combining feature structure-based formalisms with linear precedence constraints was done by Seiffert (1987, 1991), who assumes a feature structure-based ID/LP formalism. Seiffert identifies two main issues with the advent of linear precedence constraints over feature structures. The first is straightforward: if LP constraints are to be checked as local trees are built, care must be taken to ensure that either the restrictor does not strip away the information needed to test the LP constraints, or that LP constraints are evaluated with respect to the unrestricted feature structures.

The second issue, dealing with the potential for non-local LP information flow, is best illustrated with an example grammar given in Seiffert (1987), reproduced here in Figure 6.5. When parsing the string *edfg*, passive edges corresponding to the local trees in Figures 6.6a and 6.6b will be created for categories *b* and *c*, respectively. Neither of these trees violates either LP constraint. When we complete these trees with the initial prediction, the resulting top-most local tree similarly violates no LP constraint. Yet the entire edge (shown in Figure 6.7) violates LP Constraint 1: in the local tree with mother category *b*, a daughter with $\begin{bmatrix}\textsc{f1} & \textit{one}\end{bmatrix}$ follows a sister with $\begin{bmatrix}\textsc{f2} & \textit{two}\end{bmatrix}$. From this, Seiffert concludes that in addition to checking LP constraints as each local tree is completed, each candidate parse tree must be checked for LP acceptability after the chart is completed. Since this check involves testing every LP constraint at every internal node of every possible parse tree, it is quite expensive.

Morawietz (1995), working with typed (as opposed to Seiffert's untyped) unification grammars, presents a refinement of this algorithm. He notes that the relationship between a pair of categories and an LP constraint should be thought of as tripartite: if the category pair is subsumed by the constraint, the constraint **strongly applies**. Otherwise, if the category pair subsumes the constraint, the constraint **weakly applies**, and if not, the LP constraint does not apply.

Thus at the point when a local tree like 6.6a is LP-tested, the parser notes that LP Constraint 1 weakly applies to the daughter categories, and thus that constraint might be violated with the addition of new information. LP Constraint 2 does not apply, and so no amount of additional information will cause it to be violated.

As a result, when Morawietz's algorithm encounters a tree for which an LP constraint weakly applies, an entry is added to a *constraint store* for that edge: a list of constraints whose applicability has not yet been determined. The first element of the entry is a pointer to the feature structure for the current local tree; the second element is a copy of that feature structure. When further completions are done with that edge and its descendants, the parser checks to see if the two elements of each entry subsume each other; if they do, then no new LP-relevant information has been provided. If they do not subsume each other, then the constraint is re-tested for applicability; if the constraint is now violated, the completion does not succeed. Thus at no point does the chart contain LP-unacceptable structures, as it does under Seiffert's approach.

6.3 The GIDLP Parsing Algorithm for Feature Structure Categories

Now that the relevant literature has been surveyed, the result of incorporating these technologies into the GIDLP parsing algorithm can be shown. While the core algorithm is unchanged, as indicated in Figure 6.8, those sections with interesting differences will be presented in the following sections.

6.3.1 Grammar Compilation

In general, there are not as many opportunities for optimizing the parse during grammar compilation. On the one hand, categories are no longer simply equal or non-equal; they instead fit into a subsumption hierarchy, making sets of such categories harder to work with. Additionally, it is no longer always possible to determine useful reachability information. Consider the rule in (95).

(95) $\text{VP} \rightarrow \text{V}\begin{bmatrix}\textsc{subcat} & \boxed{1}\end{bmatrix}, \boxed{1}$

In the presence of such a rule, the set of categories reached by VP cannot easily be determined – while V is often a lexical category in many grammars, it need not be. More critically, it may be the case that the elements of $\boxed{1}$ may themselves be incompletely specified; this is the case, for instance, with grammars that implement argument attraction (described in section 2.3). Consider the lexical entry in (96), which yields (97) when unified into the first category of (95)'s RHS.

(96) $\text{V}\begin{bmatrix}\textsc{subcat} & \left\langle \text{V}\begin{bmatrix}\textsc{subcat} & \boxed{1}\end{bmatrix} \right\rangle \oplus \boxed{1}\end{bmatrix}$

(97) $\text{VP} \rightarrow \text{V}\begin{bmatrix}\textsc{subcat} & \left\langle \text{V}\begin{bmatrix}\textsc{subcat} & \boxed{1}\end{bmatrix} \right\rangle \oplus \boxed{1}\end{bmatrix}, \text{V}\begin{bmatrix}\textsc{subcat} & \boxed{1}\end{bmatrix}, \boxed{1}$

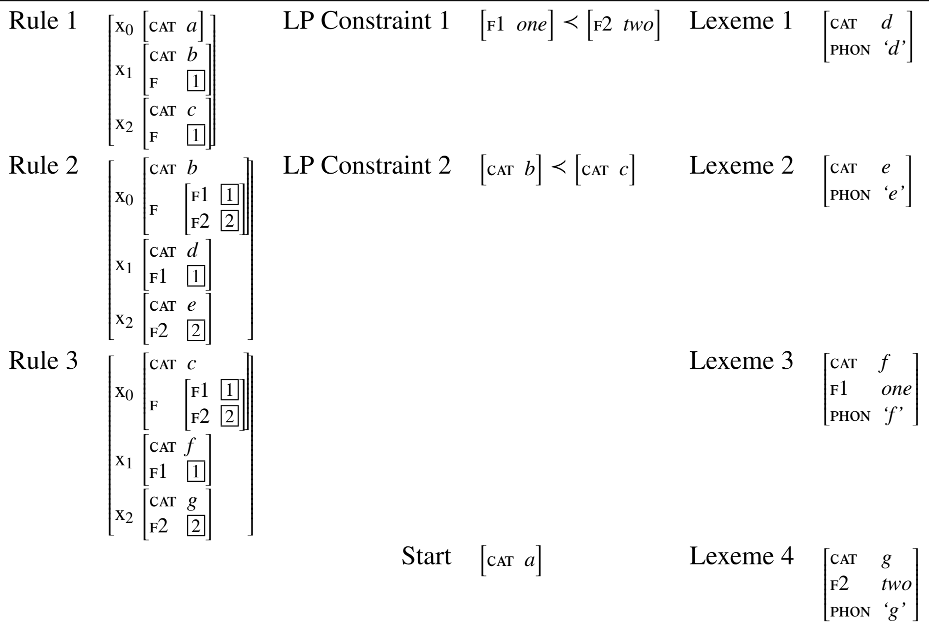


Figure 6.5: Seiffert's grammar

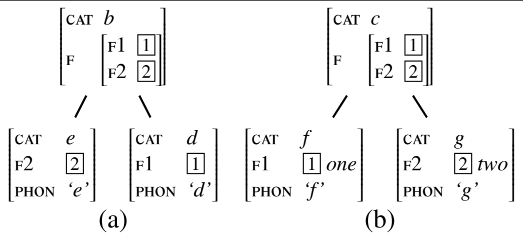


Figure 6.6: Two passive edges created during parse of *edfg*

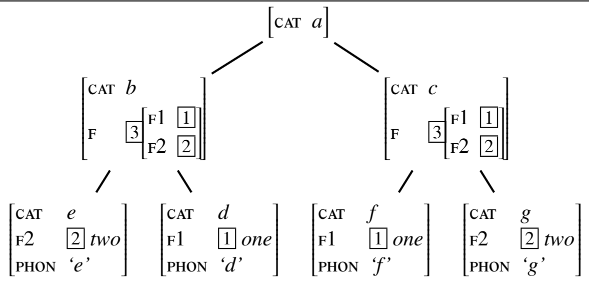


Figure 6.7: Result of completing passive edges in Figure 6.6 via rule 1

```
PROCEDURE earley:
  COMPILE the grammar (section 6.3.1)
  INITIALIZE the chart and the agenda
  WHILE the action queue is non-empty
    DEQUEUE the top action and perform it
  EXAMINE the chart for successful parses

PROCEDURE predict(A): (section 6.3.2)
  FOREACH grammar rule providing A's active category
    COMPUTE the masks for the new edge
    EVALUATE the masks for edge viability
    attempt to ADD the edge to the chart
    ENQUEUE prediction and completion with this edge

PROCEDURE complete(P, A): (section 6.3.3)
  CHECK that A and P do not overlap
  UPDATE the LP constraints and domain declarations on the edge
  attempt to ADD the new edge to the chart
  ENQUEUE prediction and completion with this edge
```

Figure 6.8: GIDLP Parsing Algorithm

More complicated cases (in this example, longer chains of modals) may require more than two levels of indirection, and in general, there is no theoretical limit to the amount of indirection involved. As a result, it could require an infinite amount of processing to determine the set of categories provably reached by a certain category, and one would need to instead approximate the reachability of that category – in the worst case, the category is just said to reach $\top$, the most general feature structure.

Grammar compilation therefore plays a much smaller role in the feature structure GIDLP parser than in the atomic GIDLP parser. It will be left to future work to decide whether further grammatical analysis could lead to more efficient parsing.

6.3.2 Prediction

During prediction, the parser examines each rule in the grammar and tests its restricted LHS against the active category on the rule being predicted. The active category is then unified with the rule's LHS to form the feature structure for the new edge. If this succeeds, then processing continues with mask generation and testing as for atomic categories.

6.3.3 Completion

During completion, the parser tests the active and passive categories for restricted compatibility and then unifies the passive category into the active edge that triggered completion. The result, if successful, proceeds schematically as for atomic categories, updating the LP constraints on the active edge. The process of LP updating is complicated by two factors, however: the potential for non-modular constraints and the need to implement a constraint store akin to Morawietz's (1995). These factors will be discussed in the next two sections.

6.3.3.1 Non-Modular LP Constraints

As has been mentioned, the conditions adopted by the GIDLP formalism under which an LP constraint is considered to be satisfied or violated were originally developed by Kasper et al. (1995). These conditions are inherently based on pairs of categories, largely due to the potential for what will be termed **non-modular** constraints. A non-modular constraint is any constraint for which there exists a node *n* in the constraint's feature structure such that paths starting from the LHS feature and the RHS feature both reach *n*. Each such *n* is referred to as a *point of non-modularity*. For example, (98) is a non-modular constraint, since the node representing the LHS of the constraint can also be reached from the RHS node after traversing the SUBCAT arc.

(98) $\boxed{1} < \left[\text{SUBCAT } \boxed{1}\right]$

The applicability of such constraints cannot be determined by knowing whether a category is subsumed by either side of the constraint; instead, constraint applicability must be determined by a pair-wise test. This presents a complication for parsing, in which categories are only located one at a time.

As an example, consider the rule in (99) as it parses (100).

(99) $A \rightarrow B_1\ B_2\ B_3\ B_4\ ; \left[\text{X } \boxed{1}\right] < \left[\text{Y } \boxed{1}\right]$

(100) $B\left[\text{X } +\right] B\left[\text{Y } +\right] B\left[\text{X } -\right] B\left[\text{Y } -\right]$

When the first category is located, the parser will recognize that the constraint applies, as the constraint's LHS subsumes the category. Under the constraint update strategy outlined in section 5.2.2.12, the parser would update the left frontier of the constraint and move on. On locating the second category, the right frontier (as only the constraint's RHS subsumes the category) would be correspondingly updated and the parser would confirm that the left frontier still preceded the right. Next, the third category is located, and since it is subsumed by the constraint's LHS, the left frontier will be updated. At this point, the constraint's left frontier has passed its right frontier, and a constraint violation would be signified. This is clearly not the intent of the constraint.

Instead, when updating any non-modular constraint, a new constraint must be generated whose feature structure has become minimally more specific. In particular, for each point of non-modularity on the side of the constraint that the triggering category could unify with, the value of the triggering category at that point is unified into that side of the constraint. For instance, the point of non-modularity in (99) is accessed through the path X on the left, and Y on the right. If this constraint were to be updated with the location of a category C found to subsume the LHS of the constraint, the new constraint would be formed by unifying C's value for X with the LHS of the constraint.

It is then only in this new constraint that the appropriate frontier is updated. Then, if the resulting constraint is subsumed by any other constraint in the list, it is merged with that constraint. Under this strategy, (100) would be parsed as sketched in (101) – (105).

(101) $A \rightarrow \bullet\ B_1\ B_2\ B_3\ B_4\ ;$
$\left[\text{X } \boxed{1}\right]: 0000 < \left[\text{Y } \boxed{1}\right]: 0000$

(102) $A \rightarrow B\left[\text{X } +\right] \bullet\ B_2\ B_3\ B_4\ ;$
$\left[\text{X } \boxed{1}\right]: 0000 < \left[\text{Y } \boxed{1}\right]: 0000, \left[\text{X } \boxed{1}+\right]: 0001 < \left[\text{Y } \boxed{1}\right]: 0000$

(103) $A \rightarrow B\left[\text{X } +\right] B\left[\text{Y } +\right] \bullet\ B_3\ B_4\ ;$
$\left[\text{X } \boxed{1}\right]: 0000 < \left[\text{Y } \boxed{1}\right]: 0000, \left[\text{X } \boxed{1}+\right]: 0001 < \left[\text{Y } \boxed{1}\right]: 0010$

(104) $A \rightarrow B\big[\text{X } +\big] B\big[\text{Y } +\big] B\big[\text{X } -\big] \bullet B_4$;

$\big[\text{X } \boxed{1}\big]: 0000 < \big[\text{Y } \boxed{1}\big]: 0000, \big[\text{X } \boxed{1}+\big]: 0001 < \big[\text{Y } \boxed{1}\big]: 0010, \big[\text{X } \boxed{1}-\big]: 0100 < \big[\text{Y } \boxed{1}\big]: 0000$

(105) $A \rightarrow B\big[\text{X } +\big] B\big[\text{Y } +\big] B\big[\text{X } -\big] B\big[\text{Y } -\big] \bullet$;

$\big[\text{X } \boxed{1}\big]: 0000 < \big[\text{Y } \boxed{1}\big]: 0000, \big[\text{X } \boxed{1}+\big]: 0001 < \big[\text{Y } \boxed{1}\big]: 0010, \big[\text{X } \boxed{1}-\big]: 0100 < \big[\text{Y } \boxed{1}\big]: 1000$

The initial setup in (101) is as above. Now, when the first category is seen to be subsumed by the constraint's LHS, the constraint itself is not updated; instead, a new constraint is added to the resulting edge, shown in (102). The second category is then located, and the new constraint is updated on its right frontier. At the same time, a third constraint is generated, representing the result of unifying the category with the right-hand side of the original constraint; this third constraint turns out to be identical to the second constraint, and so the two are merged, resulting in (103). This process continues through the remaining two categories in edges (104)–(105), where the third category no longer triggers a constraint violation, as it is not subsumed by the second constraint.

6.3.3.2 Constraint Stores

Morawietz's (1995) original constraint store is implemented by adding a pointer to and a copy of an edge's feature structure when an LP constraint is found to weakly apply to that edge. Whenever the pointed-to feature structure is found to differ from the copy, the edge is re-tested for acceptability with respect to the weakly-applying constraint.

The GIDLP parsing algorithm incorporates a refinement of this idea. Instead of storing two copies of the edge's feature structure, the parser marks the appropriate category of the edge's feature structure as *watched* and makes note of the triggering constraint in the constraint store. In any parsing step thereafter, if any aspect of a watched feature structure changes, the edge is re-tested for acceptability. As a result, the constraint store requires much less space on each edge. (Note also that, as will be discussed in section 7.2, adding the concept of watched nodes to the unifier adds almost no complexity, since very similar functionality is already present.)

Just as an LP constraint may weakly apply, so can a description-based domain declaration. As a result, the GIDLP constraint store can deal with weakly-applying domain declarations as well. When such is encountered, two edges are added to the chart: a 'true edge' where the domain declaration is assumed to apply, and a 'false edge' where it is assumed not to apply. Throughout the rest of the parse, the parser will check to see that these assumptions are maintained. Specifically, at each following step, if the relevant edge's feature structure changes, it will be examined to see if the declaration now strongly applies. If so, on the descendent of a 'true edge', the entry will be removed from the new edge's constraint store; on a 'false edge', the edge will fail. Similarly, no passive edge will be allowed to provide the start category of the grammar if its constraint store still contains a 'true' entry is still present at the end of the parse (i.e. on an edge whose result category is *dollar*), the edge will fail.

6.4 Conclusion

With the changes discussed in this section, the parsing algorithm can now handle feature structure categories as well as atomic categories, making it suitable for processing linearization-HPSG grammars. The next chapter will turn to the evaluation of the parser.

CHAPTER 7

Evaluation

In general, a new parsing algorithm should contribute to the state of the art in either (or both) of two ways: it allows grammars to express concepts more transparently, or it is more efficient than previous approaches. It has been shown in chapter 3 that no other formalism allows linearization-HPSG-based grammars to be transparently encoded; the latter quality, however, has yet to be evaluated. This section will first discuss what is expected of the GIDLP formalism in terms of efficiency and then turn to the evaluation itself.

7.1 Linguistic Expectations

Before presenting specific data on the efficiency of the parser, it is worthwhile to examine the expectations that exist for the GIDLP parser based on previous research.

7.1.1 Discontinuous Constituency as Grammar Optimization

Müller (2004) examines a series of analyses that have been offered for various phenomena in German and concludes that discontinuous constituents are often preferable on the grounds of linguistic elegance and computational efficiency. Most prominent is the discussion of the position of the finite verb. As has been shown earlier in section 6.3.1, when a verbal complex is present in a verb-initial or verb-second sentence, that complex is separated from the finite verb even though it determines the finite verb's arguments. Thus any system that can only handle continuous constituents will often be searching blindly for verbal arguments, since the verb and the verbal complex cannot form a continuous constituent. Müller goes on to show that phenomena such as extraposition and free middle-field constituent order create additional inefficiencies for continuous constituent grammars.

Müller confirms this with data comparing two broad-scale grammars of German, one that allows discontinuous consitutents, and one that does not. Müller observes that the grammar with discontinuous constituents creates far fewer[1] passive edges during parsing than does the continuous constituent grammar.

As a result, the GIDLP formalism should be expected to generalize this result: the formalism should be able to provide a discontinuous-constituent grammar for a language with relatively free word order that outperforms any phrase-structure grammar for that same language.

[1]Müller presents a graph of edges created versus sentence length, but no specific figures.

7.1.2 The Value of RHS Ordering

Recall that one of the major innovations of the GIDLP grammar format defined in section 4.1 over previous discontinuous constituent formalisms is that the order of the RHS of a grammar rule does not encode the terminal order of the daughters. Instead, it expresses the order in which the parser will search for these elements, as discussed in section 5.2.1.2.

To see why this is valuable, consider a grammar covering raising verbs in Icelandic. Many verbs in Icelandic assign "quirky case" (i.e. a non-nominative case) to their subjects; these case assignments persist when the subject is raised to be the subject or object of a matrix verb. From a parsing perspective, the embedded verb must be known before it can be determined whether a given noun phrase is an acceptable subject for the matrix verb. This is illustrated by the data in (106) – (111).

(106)

Hana	virðist	vanta	peninga
her.ACC	seems	to-lack	money

'She seems to lack money.'

(107)

Barninu	virðist	hafa	batnað	veikin
the-child.DAT	seems	to-have	recovered-from	the-disease

'The child seems to have recovered from the disease.'

(108)

Verkjanna	virðist	ekki	gæta
the-pains.GEN	seem	not	to-be-noticeable

'The pains don't seem to be noticeable.'

(109)

Hann	telur	mig	vanta	peninga
he.NOM	believes	me.ACC	to-lack	money

'He believes that I lack money.'

(110)

Hann	telur	barninu	hafa	batnað	veikin
he	believes	the-child.DAT	to-have	recovered-from	the-disease

'He believes the child to have recovered from the disease.'

(111)

Hann	telur	verkjanna	ekki	gæta
he	believes	the-pains.GEN	not	to-be-noticeable

'He believes the pains to be not noticeable.'

In other words, the fact that the subject in (106) and (109) is accusative is a reflection of the embedded verb 'lack' rather than the matrix verbs 'seem' or 'believe'; the same situation holds for the dative [(107), (110)] and genitive [(108), (111)] examples. In all other respects, however, the matrix verb is still the head of its clause (it must agree in number with the subject, for example).

Consider a head-driven parser (van Noord 1997): a variant of a phrase-structure parser in which a designated element (the head) is parsed before any other complement; the non-head daughters are then parsed in the usual left-to-right order. With such a parser, the grammar writer would write a rule like (112) to license the matrix clause.

(112) $S \rightarrow NP_{\text{subj}}\ V^{\text{head}}\ VP_{\text{inf}}$

With this rule, the parser will first locate the head (here, the **V**), then the **NP**, and finally the **VP**. As a consequence, the constraints in the **VP** on the case of the subject will not be known until after the subject has been found. The parser will therefore try all possible **NP**s as subjects, and then see which the embedded verb phrase rejects.

With the GIDLP formalism, in contrast, the grammar writer could specify the rule as (113) to avoid this generate-and-test pattern.

(113) $S \rightarrow V^1\ VP^2_{inf},\ NP^3_{subj}$

Now the parser will not look for the subject of the clause until the embedded verb phrase has been located, and so only **NP**s with the appropriate case will be considered.

7.1.3 Computational Complexity

The computational complexity of the GIDLP formalism itself has not been investigated. Such results do exist, however, for Suhre's LSL grammar formalism, which is a subset of the GIDLP formalism. As a result, the GIDLP formalism can be no less complex than the LSL formalism. Suhre's (1999) central finding is that the membership problem for his LSL grammar formalism is NP-complete, both when considering the grammar plus the string as input (general membership problem) as well as when only the string is considered as input (fixed membership problem).

As Huynh (1983) has shown, the general membership problem for unordered context-free grammars (ID/LP grammars without LP statements) is also NP-complete, so Suhre's first result is not surprising. That the fixed membership problem for LSL grammars is also NP-complete is less straightforward; Suhre (1999, 61ff) demonstrates that it stems from the potential for recursive growth of discontinuities. As a result, when the parser can assume an upper bound on the number of discontinuities in any given constituent, the fixed membership problem becomes polynomial. Formally, this can be achieved by requiring that the number of discontinuities introduced by a recursive non-terminal is bounded by some constant.

Interestingly, a related practical proposal based on linguistic argumentation is discussed by Müller (1999b). He proposes a continuity constraint for linearization-based HPSG which requires saturated phrasal elements (that is, maximal projections) to be continuous.[2] Müller shows that adding his continuity constraint results in a significant reduction in the number of passive edges and thereby significant improvements in parsing performance.

This continuity constraint is weaker than Suhre's condition in that recursion on the level of adjunction is not restricted. It is, however, interesting to note in this context that a grammar incorporating the $\overline{X}$-schema (Jackendoff 1977) will require all non-head constituents to be maximal projections. This result therefore ties in with Blevins's (1990) (see section 2.1.3) suggestion that maximal projections should coincide with word order domains. In sum, Müller's result strongly suggests that further research on linguistically-motivated continuity constraints can result in efficient parsing of those GIDLP grammars which include such constraints.

7.2 Implementation Notes

Both the atomic parsing algorithm of chapter 5 and the feature structure parsing algorithm of chapter 6 have been implemented in C++. The feature structure parser incorporates the feature structure handling routines from PET (Callmeier 2001), which provides a state-of-the-art implementation of unification and subsumption for typed feature structures. Throughout development, each parser was tested on the example grammars shown throughout this thesis to ensure accurate implementation of the various aspects of the algorithm.

[2]If extraposition is handled via discontinuous constituents, a more complex constraint is required.

7.2.1 Performance on Suhre's Grammars

A second check for accuracy arises through the nine grammars provided by Suhre (1999). Each grammar consists the rule in (114) with differing LP and isolation constraints. The grammars are summarized in (115). For instance, grammar #1 illustrates right isolation (in which each RHS category is its own domain) and immediate precedence.

(114) A → A, A

(115)

	no prec.	weak prec.	immed. prec.	constraint(s)
no isol.	#9	#8	#3	
left isol.	#5	#4	#2	⟨{0}, A, []⟩
right isol.	#7	#6	#1	⟨{1}, A, []⟩, ⟨{2}, A, []⟩
constraint		$1 < 2$	$1 \ll 2$	

Grammars 1, 2, 3, and 4 are effectively context-free grammars; grammar 9 is the fully exponential case, where every subset of the input has a corresponding passive edge. The remaining grammars occupy points between these two extremes.

Suhre (1999) reports the size of the final chart when each grammar is used to parse sentences of the form a^n. For grammars 5, 7, and 9, the GIDLP parser achieves identical chart size; for the remaining grammars, the GIDLP parser creates exactly twice as many active edges. This is the result of the parser's treatment of p-masks during prediction. Since the sole rule of the grammar is recursive, it is initially predicted with a full positive mask (see section 5.2.2.6) and then re-predicted with an empty positive mask (since daughter edges do not get the positive mask of their mother during prediction). Since the parser lacks retroactive ambiguity packing (see section 5.2.1.5), every subsequent completion occurs with both a descendant of the first edge and a descendant of the second edge, leading to the exact doubling of active edges.

As a result, since Suhre's grammars illustrate the worst-case scenario for each combination of grammar features, the identity or near-identity in chart size confirms that the GIDLP is properly creating all necessary edges.

7.3 Efficiency

In the optimal case, the efficiency of the GIDLP parser would be tested by comparing two grammars with identical coverage. One would be a GIDLP grammar; the other, a grammar modelling discontinuous constituents with existing technology – a grammar for LKB (Copestake 1992) or ALE (Haji-Abdolhosseini and Penn 2003), for instance. Differences in execution time on a common testsuite between the two parsers could then be used to determine the empirical effect of switching from order constraints expressed through general constraint satisfaction and relations to order constraints used directly by the parser during prediction and completion.

Unfortunately, grammars of the requisite scale that are documented well enough to be quickly ported to a new system do not exist, and their development is a research project in its own right. As an example, both the ERG grammar of English (Flickinger 2000) and the JaCY grammar of Japanese (Siegel 2000) are still in active development despite having been worked on by teams of researchers for nearly ten years. This section will therefore concentrate on comparative results from smaller-scale grammars.

7.3.1 Performance on Context-free Grammars

The GIDLP grammars form a superset of the context-free grammars. Thus it would be desirable for a GIDLP parser to be just as efficient as a context-free parser when presented with a context-free grammar encoded in the GIDLP format, especially in light of Volk (1996), which showed that in terms of efficiency and expressivity, it is advantageous to be able to combine ID/LP and ordinary context-free rules in one grammar.

This raises the question of how the parsing algorithm proposed in this paper performs when used to parse grammars that contain only context-free rules. To investigate this, the parser's performance has been tested with the three types of context-free grammars discussed in Earley (1970) – those that require linear, quadratic, and cubic time for strings to be recognized; following Earley, the number of edge insertion attempts (whether successful or unsuccessful) is used as a metric (represented as **cost** in the charts that follow).

Earley uses the context-free grammar in (116) (presented here with its GIDLP equivalent) to test the linear and quadratic aspects of the algorithm.

(116)

CFG	GIDLP
root(X)	root(X, [])
$X \rightarrow A$	$[X] \rightarrow A_1$
$X \rightarrow X\ B$	$[X] \rightarrow X_1\ B_2\ ;\ 1 \ll 2$
$X \rightarrow Y\ A$	$[X] \rightarrow Y_1\ A_2\ ;\ 1 \ll 2$
$Y \rightarrow E$	$[Y] \rightarrow E_1$
$Y \rightarrow Y\ D\ Y$	$[Y] \rightarrow Y_1\ D_2\ Y_3\ ;\ 1 \ll 2, 2 \ll 3$

Consider the input string **(ed)**^{x}**eab**^{y} (where a^x abbreviates x copies of a). With this grammar, Earley reports that the number of edge insertion attempts for his algorithm increases linearly with y and quadratically with x. This can be empirically verified by looking at the ratio of additional cost (the total number of edge insertion attempts during parsing) to some power of the increase in work (the difference in string length). If that ratio converges, then it can be said that that power is an upper bound on the complexity of the task.

With the linear case, the ratio between additional cost and additional work should converge. The results in (117) show that this is attained by the GIDLP parser.

(117)

Input	Cost	Work	ΔCost	ΔWork	$\frac{\Delta\text{Cost}}{\Delta\text{Work}}$
ededeab	76	1	n/a	n/a	n/a
ededeab^2	89	2	13	1	13
ededeab^3	102	3	26	2	13
ededeab^5	128	5	52	4	13
ededeab^{10}	193	10	117	9	13

That the rightmost column converges illustrates the linear performance achieved by the GIDLP parser on this grammar.

For the quadratic case, the ratio between additional cost and the square of additional work should converge. The results in (118) show that this is not attained by the GIDLP parser.

(118)

Input	Cost	Work^2	ΔCost	Δ(Work^2)	$\frac{\Delta\text{Cost}}{\Delta\text{Work}^2}$	$\frac{\Delta\text{Cost}}{\Delta\text{Work}^3}$
(ed)^2**ea**	61	4	n/a	n/a	n/a	n/a
(ed)^3**ea**	95	9	34	5	6.80	1.79
(ed)^4**ea**	142	16	81	12	6.75	1.45
(ed)^5**ea**	204	25	143	21	6.81	1.22
(ed)^6**ea**	281	36	220	32	6.88	1.06

Since the sixth column does not converge, it cannot be the case that the GIDLP parser operates on this aspect of the grammar in quadratic time; this performance goal has not yet been met. From the fact that the rightmost column converges, however, it can be seen that the performance is no worse than cubic.

Earley's sample cubic-time CFG is given in (119). The corresponding results are given in (120).

(119) $B \to B\ B$
$B \to A$

(120)

Input	Cost	Work^3	ΔCost	Δ(Work^3)	$\frac{\Delta\text{Cost}}{\Delta\text{Work}^3}$
A^5	68	125	n/a	n/a	n/a
A^6	104	216	36	91	0.40
A^7	153	343	85	218	0.39
A^8	213	512	145	387	0.37
A^{10}	378	1000	310	875	0.35

Since the right-hand column converges, the parser can be seen to have attained the goal of cubic time on this grammar.

In summary, the GIDLP parser achieves performance comparable to that of a CFG parser when asked to parse a context-free grammar encoded in the GIDLP format for most types of context-free grammars. In all cases, the GIDLP grammar parses GIDLP-encoded CFGs in no worse than cubic time.

7.3.2 Larger-scale Grammar Evaluation

To test the effectiveness of the GIDLP formalism, a moderate-scale grammar of German was obtained from Professor Martin Volk (Stockholm University). This grammar combines ID/LP rules with PS rules, as argued for in (Volk 1996); it contains rules for most of the basic clause structure of German, but lacks provisions for truly non-local phenomena. As each ID/LP rule necessarily encodes its own domain (see section 2.1.3), the flexibility of the German middle-field is modelled by the use of a flat structure. As an example, the rule for ditransitive verbs is given in (121).

(121)
```
id( 'Satz'(_),
   ['N2'(numerus:N..kasus:'NOM'.. person:P),
    'V'(numerus:N..person:P..subcat:'AD00'(_,_)..vform:fin),
    'N2'(kasus:'DAT'),
    'N2'(kasus:'AKK'),
    stern('ADV'(_)),
    opt('ERG'(_)),
    opt('PRAEF'(_))]).
```

The rule requires a verb with the appropriate subcategorization class, the three verbal complements, as well as an optional complement labelled ERG, an optional prefix (for separable verbs), and any number of adverbs. A set of linear precedence constraints accounts for the position of the finite verb (and prefix, if present) while allowing the arguments and adverbs to occur in any order.

This grammar can be directly translated into GIDLP format, following several straightforward steps. Each of Volk's PS rules corresponds to a GIDLP rule (as illustrated in section 4.2.1), as does each of Volk's ID rules. Since there is no difference in the resulting notation between the rules that were originally PS or ID rules, each of Volk's LP constraints must be stated as local constraints to each ID rule, or else they would apply to the originally-PS rules as well. Second, since the GIDLP formalism lacks support for optional and Kleene-starred categories, the rule must be split into several versions, one for each combination of present and absent categories. Here, three optional categories will generate eight rule versions, schematized in (122).

(122) Satz → N2 V N2 N2 ADVSTAR ERG PRAEF
Satz → N2 V N2 N2 ADVSTAR ERG
Satz → N2 V N2 N2 ADVSTAR PRAEF
Satz → N2 V N2 N2 ADVSTAR
Satz → N2 V N2 N2 ERG PRAEF
Satz → N2 V N2 N2 ERG
Satz → N2 V N2 N2 PRAEF
Satz → N2 V N2 N2

Finally, a category ADVSTAR must be introduced that dominates any number of ADVs without forming a word order domain; this will allow the ADVs to be permuted among the arguments for the rule.

This establishes an 'initial' grammar. To evaluate the empirical advantage of the GIDLP formalism, the grammar was altered to take advantage of several of the GIDLP features. The first revision concerned the ability to have word order domains larger than the local tree. As noted above, each ID rule in Volk's grammar had to be converted into multiple GIDLP rules. As a result, the baseline grammar contains 371 rules, in contrast to the 121 rules (68 PS and 53 ID) in Volk's grammar. This can be avoided in the GIDLP formalism by creating rules of the form *Satz → ADV Satz* that do not form word order domains; in addition, the LP constraints are moved from the various Satz rules to a new level of structure (the category 'clause'). Following this modification, the 'medial' grammar contains 143 rules. In effect, this ability to structurally recurse without creating additional word order domains generalizes Volk's use of Kleene star, which can only recurse on a single category without creating structure.

A second optimization results from taking advatange of the GIDLP ability to freely specify the RHS order of a rule. While it is not clear what principle underlay the orderings present in Volk's ID rules, it is clear that they are not optimal for GIDLP. Virtually all sentence-level rules have a nominative NP as their first argument – the least discriminatory argument of a German sentence. In the 'final' version of the grammar, the right-hand sides were ordered to put the finite verb first, followed by the verbal complex (if any); the order of the other arguments was left unchanged. It is clear that a more detailed analysis of each rule could lead to further reorderings; nonetheless, the effect of a simple reordering on this grammar can be taken as a lower bound on the potential for increased efficiency.

To compare these three grammars (the original, the first modified version with additional structure for optional arguments, and the second modified version with reordered RHSs), a testsuite of 150 sentences was constructed. The sentences were generally chosen to equally cover the sentence types recognized by the grammar. The results from parsing this testsuite with each grammar are summarized in Figure 7.1, which shows the average number of chart insertion attempts at each sentence length. (Chart insertion attempts have traditionally been used as an overall metric for parsing efficiency, as parse time tends to be dominated by the time taken searching the chart for blocking edges.) Overall, the final grammar shows a clear improvement over the medial and initial grammars.

The raw data from this experiment is shown in Figure 7.2. All sentences in the test suite were assigned a serial number in order of increasing sentence length, followed by increasing chart insertion attempts on the initial grammar; this graph then shows the number of chart insertion attempts for each of the 150 sentences, grouped by sentence length. As can be seen, sentence length does not directly correlate with parsing complexity, as the lines do drop at sentence length boundaries.

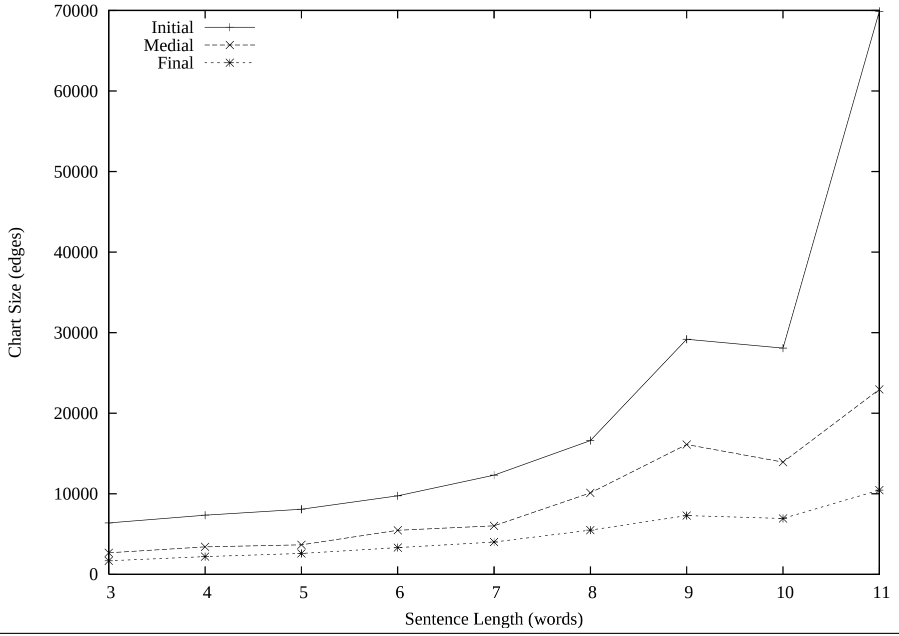


Figure 7.1: Average Chart Size per Sentence Length

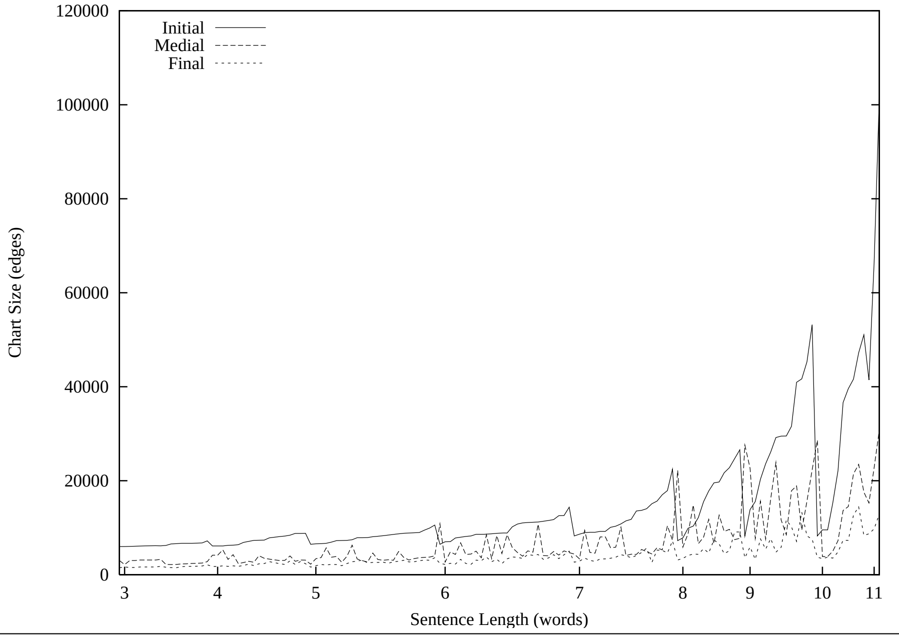


Figure 7.2: Chart Size per Sentence Length for Individual Sentences

Averaging over all 150 sentences, the final grammar sees a decrease of 69.2% in the number of chart insertion attempts compared to the initial grammar. Thus the expressive capabilities provided by the GIDLP formalism lead directly to improvements in parsing efficiency.

7.4 Conclusion

The GIDLP formalism was designed to allow grammar writers to specify linearization-HPSG grammars that can be efficiently processed. The results described in this section show that the two main aspects of the GIDLP formalism – the ability to specify word order domains larger than the local tree and to order the RHS according to discriminatory capacity – provide clear enhancements in chart size and execution time when exploited.

Chapter 8

Conclusion

8.1 Future Work

Future work with the GIDLP formalism and parser centers around two goals: the goal to verify the empirical worth of GIDLP, and the goal to improve the efficiency of the GIDLP parser.

True comparisons of the parser, whether against other systems, or against other versions of itself, can only be done in the context of a large-scale, broad-coverage grammar making use of all of the GIDLP facilities. Such a grammar could be constructed in two ways: by writing a GIDLP grammar from scratch, or by extracting a GIDLP grammar from an already-existing resource – either an existing broad-coverage grammar for a different formalism or a suitably-encoded treebank.

With such a grammar in hand, not only could the parser be tested against other systems – in the sense of the apples-to-apples comparison described in section 7.3 – but it could also be tested against alternate implementations of the same algorithm.

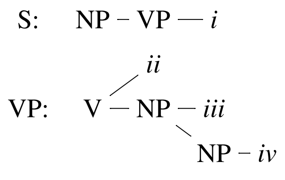
For instance, as discussed in section 5.2.1.5, the GIDLP parser does not currently implement retroactive ambiguity packing – the process of removing edges from the chart whenever a more general edge is added, and updating all references to that old edge to point to the new edge instead. It is clear that this process involves a tradeoff: the edge insertion mechanism becomes more complex, as both the chart itself as well as the parsing agenda would have to be examined for references to the old edge, but the overall chart size would not grow as quickly, speeding up those aspects of the parser that retrieve edges from the chart. By building a second implementation of the parser that performs retroactive ambiguity packing and comparing its performance on a testsuite for a suitably broad-coverage grammar, the empirical merit of this feature could be assessed.

Toward the second goal, this thesis has concentrated on those parsing optimizations of particular importance to grammars recognizing discontinuous constituents. Further optimizations applicable to all parsing systems remain to be considered. For example, one could store the grammar as a trie (Fredkin 1960) – a tree-like data structure originally developed to store strings in such a way as to minimize the redundancy incurred by duplicating prefixes. In a trie, each internal node represents a common prefix shared by two or more of the contained strings; each string has its own leaf node. In the context of a grammar, each LHS category has an associated trie; Figure 8.1 illustrates the correspondence between a CFG and its corresponding trie-encoded form. Here, each of the four rules (i)–(iv) in the CFG corresponds to a leaf node in the trie encoding.

Encoding the grammar in this manner has the effect of collapsing many similar parse edges into one edge. For instance, with the grammar in Figure 8.1a, once VP is predicted, there will be three active edges awaiting a V, one for each of rules (ii)–(iv). Once a passive

i)	S → NP VP
ii)	VP → V
iii)	VP → V NP
iv)	VP → V NP NP

(a)



(b)

Figure 8.1: Storing Grammar in a Trie

edge providing V is found, all three will be completed with that edge to produce three new edges, and so on. With the trie-encoded grammar, this repetitive work would only be done once.

The completion process does become more complicated as a result – instead of the ‘dot’ moving serially from one position to the next, it will periodically need to split, following all of the out-edges of a node simultaneously. As a result, the completion of two edges may yield multiple result edges.

Adaptation of this concept to the GIDLP parser would be complex, as it would be possible to have LP constraints and domain declarations only applying to certain paths through the trie. The implementation would likely use a mechanism similar to that for dormant constraints (section 5.2.1.4), with provisions for pruning constraints for paths-not-taken. Empirical verification would be crucial to determining whether this proposed optimization presents an overall improvement or not.

8.2 Conclusion

This thesis was initially motivated by the observation that discontinuous constituents, long a part of syntactic theory and recently popularized in HPSG via work in linearization, present a dilemma to linguists with an interest in grammar implementation. Should they adopt discontinuous constituents for their theoretical elegance and suffer the price they impose in terms of processing efficiency, or should they surrender elegance and adopt the coerced system of continuous consitutents that current processing systems can handle?

The GIDLP formalism allows linguists to specify grammars with LP constraints operating within explicitly declared word order domains, and with ID rules in which the grammar writer can order the RHS as to minimize the number of parsing hypotheses that must ultimately be abandoned. With these two empirically-proven optimizations, the GIDLP formalism is seen to resolve this dilemma, providing linguists with the tools they need to efficiently work with discontinuous constituents in any language regardless of where it falls on the word order continuum from completely-fixed to completely-free.

Bibliography

Bach, Emmon. 1983. On the relationship between word-grammar and phrase-grammar. *Natural Language and Linguistic Theory*, 1.65–89.

Blevins, James. 1990. *Syntactic Complexity: Evidence for Discontinuity and Multidomination*. Ph.D. thesis, University of Massachusetts, Amherst, MA.

Bloomfield, Leonard. 1933. *Language*. New York: Henry Holt and Company.

Bonami, Olivier, Danièle Godard, and Jean-Marie Marandin. 1999. Constituency and word order in French subject inversion. Gosse Bouma, Erhard W. Hinrichs, Geert-Jan M. Kruijff, and Richard T. Oehrle, editors, *Constraints and Resources in Natural Language Syntax and Semantics*, Studies in Constraint-Based Lexicalism, 21–40. Stanford, CA: CSLI.

Bröker, Norbert. 1998. Separating surface order and syntactic relations in a dependency grammar. *Proceedings of the 17th International Conference on Computational Linguistics (COLING) and the 36th Annual meeting of the Association for Computational Linguistics (ACL)*, 174–180. Montreal. URL `http://www.aclweb.org/anthology/P98-1026`.

Bunt, Harry. 1991. Parsing with discontinuous phrase structure grammar. Masaru Tomita, editor, *Current Issues in Parsing Technology*, 49–64. Boston: Kluwer.

Bunt, Harry and K. van der Sloot. 1996. Parsing as dynamic interpretation of feature structures. Harry Bunt and Masaru Tomita, editors, *Recent Advances in Parsing Technology*. Dordrecht: Kluwer Academic Publishers.

Bunt, Harry and Arthur van Horck, editors. 1996. *Discontinuous Constituency*, volume 6 of *Natural Language Processing*. Berlin and New York, NY: Mouton De Gruyter.

Callmeier, Ulrich. 2001. *Efficient Parsing with Large-Scale Unification Grammars*. Diplomarbeit, Fachrichtung Informatik, Universität des Saarlandes. URL `http://www.coli.uni-sb.de/~uc/thesis/thesis.pdf`.

Campbell-Kibler, Kathryn. 2002. Bech's problem, again: Using linearization on Dutch R-pronouns. Frank Van Eynde, Lars Hellan, and Dorothee Beermann, editors, *Proceedings of the 8th International Conference on Head-Driven Phrase Structure Grammar*, 87–102. URL `http://cslipublications.stanford.edu/HPSG/2/hpsg01-toc.html`.

Chomsky, Noam. 1957. *Syntactic Structures*. Number 4 in Janua Linguarum. The Hague: Mouton.

Chung, Chan. 1993. Korean auxiliary verb constructions without VP nodes. Susumo Kuno, Ik-Hwan Lee, John Whitman, Joan Maling, Young-Se Kang, and Young Joo Kim, editors, *Proceedings of the 1993 Workshop on Korean Linguistics*, number 5 in Harvard Studies in Korean Linguistics, 274–286. Cambridge, MA: Harvard University Department of Linguistics.

Copestake, Ann. 1992. The ACQUILEX LKB: Representation issues in semi-automatic acquisition of large lexicons. *Proceedings of the Third ACL Conference on Applied*

Natural Language Processing, 88–96. Trento, Italy. URL http://www.aclweb.org/anthology/A92-1012.

Covington, Michael A. 1990. A dependency parser for variable-word-order languages. Technical Report Research Report AI-1990-01, Artificial Intelligence Programs, University of Georgia. URL http://www.ai.uga.edu/ftplib/ai-reports/ai199001.pdf.

Covington, Michael A. 1992. A dependency parser for variable-word-order languages. K. R. Billingsley, Hilton U. Brown III, and Ed Derohanes, editors, *Computer assisted modeling on the IBM 3090: Papers from the 1989 IBM Supercomputing Competition*, volume 2, 799–845. Athens, GA: Baldwin Press.

Covington, Michael A. 1994. Discontinuous dependency parsing of free and fixed word order: Work in progress. Technical Report Research Report AI-1994-02, Artificial Intelligence Programs, University of Georgia. URL http://www.ai.uga.edu/ftplib/ai-reports/ai199402.pdf.

Curry, Haskell B. 1961. Some logical aspects of grammatical structure. *Structure of Language and its Mathematical Aspects: Proceedings of the Twelfth Symposium in Applied Mathematics*, 56–68. Providence: American Mathematical Society.

Daniels, Michael W. and W. Detmar Meurers. 2002. Improving the efficiency of parsing with discontinuous constituents. Shuly Wintner, editor, *Proceedings of NLULP-02: The Seventh International Workshop on Natural Language Understanding and Logic Programming*, 49–68. Copenhagen, Denmark: Roskilde University, Computer Science Department. URL http://www.cs.haifa.ac.il/~shuly/nlulp02/papers/meurers.pdf.

Daniels, Michael W. and W. Detmar Meurers. 2004a. A grammar formalism and parser for linearization-based HPSG. *Proceedings of the Twentieth International Conference on Computational Linguistics*, 169–175. URL http://www.ling.ohio-state.edu/~daniels/gidlpparser.pdf.

Daniels, Mike and Detmar Meurers. 2004b. GIDLP: A grammar format for linearization-based HPSG. Stefan Müller, editor, *Proceedings of the Eleventh International Conference on Head-Driven Phrase Structure Grammar*, 93–111. Stanford: CSLI Publications. URL http://cslipublications.stanford.edu/HPSG/5/.

Davis, Paul C. 2002. *Stone Soup Translation: The Linked Automata Model*. Ph.D. thesis, Ohio State University, Columbus, OH. URL http://www.ling.ohio-state.edu/~pcdavis/papers/diss.html.

Donohue, Cathryn and Ivan A. Sag. 1999. Domains in Warlpiri. *Abstracts of the Sixth Int. Conference on HPSG*, 101–106. Edinburgh: University of Edinburgh. URL http://www-csli.stanford.edu/~sag/papers/warlpiri.ps.

Dowty, David R. 1982. More on the categorial analysis of grammatical relations. *Ohio State Working Papers in Linguistics*, 26.102–133.

Dowty, David R. 1996. Towards a minimalist theory of syntactic structure. Bunt and van Horck (1996).

Dowty, David R. 1997. Non-constituent coordination, wrapping, and multimodal categorial grammars. M. L. Dalla Chiara et al., editor, *Structures and Norms in Science*, 347–368. Kluwer.

Drach, Erich. 1937. *Grundgedanken der deutschen Satzlehre*. Frankfurt: Diesterweg.

Earley, Jay. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2).94–102. Also in Grosz et al. (1986).

Erbach, Gregor. 1997. *Bottom-Up Earley Deduction for Preference-Driven Natural Language Processing*. Ph.D. thesis, Universität des Saarlandes. URL http://www.coli.uni-sb.de/~erbach/pub/.

Flickinger, Dan. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6(1).15–28.

Fouvry, Frederik and Detmar Meurers. 2000. Towards a platform for linearization grammars. Erhard W. Hinrichs, Detmar Meurers, and Shuly Wintner, editors, *Proceedings of the Workshop on Linguistic Theory and Grammar Implementation*, 153–168. Birmingham, UK: ESSLLI 2000. URL `http://www.ling.ohio-state.edu/~dm/papers/fouvry-meurers2000.html`.

Fredkin, E. 1960. Trie memory. *Communications of the ACM*, 3(9).490–499.

Gazdar, Gerald, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. 1985. *Generalized Phrase Structure Grammar*. Cambridge, MA: Harvard UP.

Gazdar, Gerald and Geoffrey Pullum. 1981. Subcategorization, constituent order, and the notion head. Michael H. Moortgat, Harry van der Hulst, and Teun Hoekstra, editors, *The Scope of Lexical Rules*, 107–123. Dordrecht: Foris.

Gerdemann, Dale. 1991. Parsing and generation of unification grammars. Technical Report CS-91-06, Beckman Institute, University of Illinois.

Gerdes, Kim and Sylvain Kahane. 2001. Word order in German: A formal dependency grammar using a topological hierarchy. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*. URL `http://www.aclweb.org/anthology/P01-1029`.

Götz, Thilo and Walt Detmar Meurers. 1995. Compiling HPSG type constraints into definite clause programs. *Proceedings of the 33nd Annual Meeting of the Association for Computational Linguistics (ACL 95)*, 85–91. Cambridge, MA: MIT. URL `http://ling.osu.edu/~dm/papers/acl95.html`.

Götz, Thilo and Walt Detmar Meurers. 1997a. The ConTroll system as large grammar development platform. *Proceedings of the Workshop "Computational Environments for Grammar Development and Linguistic Engineering (ENVGRAM)" held at ACL/EACL*, 38–45. Association for Computational Linguistics, Madrid: Universidad Nacional de Educación a Distancia. URL `http://www.aclweb.org/anthology/W97-1506`.

Götz, Thilo and Walt Detmar Meurers. 1997b. Interleaving universal principles and relational constraints over typed feature logic. *Proceedings of the 35th Annual Meeting of the ACL and the 8th Conference of the EACL*, 1–8. Madrid, Spain: Universidad Nacional de Educación a Distancia. URL `http://www.aclweb.org/anthology/P97-1001`.

Grosz, Barbara, Karen Sparck Jones, and Bonnie Lynn Webber, editors. 1986. *Readings in Natural Language Processing*. Los Altos, CA: Morgan Kaufmann.

Haji-Abdolhosseini, Mohammad and Gerald Penn. 2003. ALE reference manual. URL `http://www.ale.cs.toronto.edu/docs/ref/ale_ref.pdf`, ms., Univ. Toronto.

Halpern, Aaron. 1995. *On the Placement and Morphology of Clitics*. Stanford: CSLI.

Harrison, S. P. and T. M. Ellison. 1992. Restriction and termination in parsing with feature-theoretic grammars. *Computational Linguistics*, 18(4).519–530.

Hays, David G. 1964. Dependency theory: A formalism and some observations. *Language*, 40.511–525.

Hepple, Mark. 1994. Discontinuity and the lambek calculus. *Proceedings of the 15th Conference on Computational Linguistics (COLING-94)*. Kyoto. URL `ftp://ftp.dcs.shef.ac.uk/home/hepple/papers/coling94.ps`.

Hinrichs, Erhard, Julia Bartels, Yasuhiro Kawata, Valia Kordoni, and Heike Telljohann. 2000. The Tübingen treebanks for spoken German, English, and Japanese. Wolfgang Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*, 552–576. Berlin: Springer.

Hinrichs, Erhard and Tsuneko Nakazawa. 1990. Subcategorization and VP structure in German. Unpublished Manuscript. From a talk delivered at the Third Symposium on Germanic Linguistics.

Höhle, Tilman. 1978. *Lexikalistische Syntax: Die Aktiv-Passiv-Relation und andere Infinitkonstruktionen im Deutschen*. Tübingen: Max Niemeyer Verlag.

Höhle, Tilman. 1986. Der Begriff "Mittelfeld": Anmerkungen über die Theorie der topologischen Felder. *Kontroversen, alte und neue: Akten des 7. Internationalen Germanisten-Kongresses, Göttingen 1985, Band 3*, 329–340. Tübingen: Niemeyer.

Huck, Geoffrey. 1985. Exclusivity and discontinuity in phrase structure grammar. *West Coast Conference on Formal Linguistics (WCCFL)*, volume 4, 92–98. Stanford University, CSLI Publications.

Huck, Geoffrey and Almerindo Ojeda, editors. 1987. *Discontinuous Constituency*. Number 20 in Syntax and Semantics. New York: Academic Press.

Huynh, Dung T. 1983. Commutative grammars: The complexity of uniform word problems. *Information and Control*, 57(1).21–39.

Jackendoff, Ray. 1977. *X-Bar Syntax: A Study of Phrase Structure*. Cambridge, Mass.: MIT Press.

Johnson, Mark. 1985. Parsing with discontinuous constituents. *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, 127–132. Chicago. URL `http://www.aclweb.org/anthology/P85-1015`.

Jourdan, Martin and Didier Parigot. 1990. Techniques for improving grammar flow analysis. Neil Jones, editor, *European Symposium on Programming*, volume 432 of *Lecture Notes in Computer Science*, 240–255. Copenhagen: Springer-Verlag. URL `http://www-rocq.inria.fr/oscar/ftp/fnc2/publications/esop90-t.ps.gz`.

Kasami, T. and K. Torii. 1969. A syntax-analysis procedure for unambiguous context-free grammars. *Journal of the Association for Computing Machinery*, 16(3).423–431.

Kasper, Robert, Andreas Kathol, and Carl Pollard. 1995. A relational interpretation of linear precedence constraints. The Ohio State University.

Kathol, Andreas. 1995. *Linearization-Based German Syntax*. Ph.D. thesis, The Ohio State University.

Kathol, Andreas. 2000. *Linear Syntax*. Oxford: Oxford University Press.

Kathol, Andreas and Carl Pollard. 1995. Extraposition via complex domain formation. *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 174–180. URL `http://www.aclweb.org/anthology/P95-1024`.

Kay, Martin. 1980. Algorithm schemata and data structures in syntactic processing. Grosz et al. (1986), 35–70.

Kiss, Tibor. 1995. *Infintive Komplementation*. Number 333 in Linguistische Arbeiten. Tübingen: Max Niemeyer Verlag.

Koch, Ulrich. 1993. The enhancement of a dependency parser for latin. Technical Report Research Report AI-1993-03, Artificial Intelligence Programs, University of Georgia. URL `http://www.ai.uga.edu/ftplib/ai-reports/ai199303.pdf`.

Kroch, Anthony S. and Aravind K. Joshi. 1987. Analyzing extraposition in a tree adjoining grammar. Huck and Ojeda (1987).

Lee, Sun-Hee. 1999. Argument composition and linearization in Korean noun-verb complex predicate constructions. Amalia Todirascu, editor, *Proceedings of the ESSLLI Student Session 1999*, 65–78. Utrecht University.

Lewis, Harry R. and Christos H. Papadimitriou. 1998. *Elements of the Theory of Computation*. Upper Saddle River, NJ: Prentice-Hall, second edition.

Maekawa, Takafumi. 2004. Constituency, word order and focus projection. Stefan Müller, editor, *Proceedings of the Eleventh International Conference on Head-Driven Phrase Structure Grammar*, 168–188. Stanford: CSLI Publications. URL `http://cslipublications.stanford.edu/HPSG/5/`.

Malouf, Rob, John Carroll, and Ann Copestake. 2000. Efficient feature structure operations without compilation. *Natural Language Engineering*, 6(1).29–46. Reprinted in (Oepen et al. 2002), 105–125.

McCawley, James D. 1982. Parentheticals and discontinuous constituent structure. *Linguistic Inquiry*, 13(1).91–106.

Meurers, Walt Detmar. 1999. *Lexical Generalizations in the Syntax of German Non-Finite Constructions*. Ph.D. thesis, Seminar für Sprachwissenschaft, Universität Tübingen, Tübingen, Germany. URL `http://ling.osu.edu/~dm/papers/diss.html`, published 2000 as Volume 145 in Arbeitspapiere des SFB 340, ISSN 0947-6954/00.

Möncke, Ulrich and Reinhard Wilhelm. 1982. Iterative algorithms on grammar graphs. H. J. Schneider and Herbert Göttler, editors, *Conference on Graphtheoretic Concepts in Computer Science*, 177–194. München: Hanser Verlag.

Morawietz, Frank. 1995. Formalization and parsing of unification–based id/lp grammars. Arbeitspapiere des SFB 340. Nr. 68, Universität Tübingen. URL `http://www.sfs.uni-tuebingen.de/sfb/reports/berichte/68/68abs.html`.

Morrill, Glynn V. 1995. Discontinuity in categorial grammar. *Linguistics and Philosophy*, 18.175–219.

Müller, Stefan. 1997. Yet another paper about partial verb phrase fronting in German. Research Report RR-97-07, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken. URL `http://www.cl.uni-bremen.de.de/~stefan/Pub/pvp.html`.

Müller, Stefan. 1999a. *Deutsche Syntax deklarativ: Head-Driven Phrase Structure Grammar für das Deutsche*. Number 394 in Linguistische Arbeiten. Tübingen: Niemeyer.

Müller, Stefan. 1999b. Restricting discontinuity. Verbmobil Report 237, DFKI, Saarbrükken. URL `http://www.dfki.de/~stefan/Pub/e_restricting.html`, also published in the Proceedings of GLDV 99 (Frankfurt/Main).

Müller, Stefan. 2004. Continuous or discontinuous constituents? A comparison between syntactic analyses for constituent order and their processing systems. *Research on Language and Computation, Special Issue on Linguistic Theory and Grammar Implementation*, 2(2).209–257. URL `http://www.cl.uni-bremen.de/~stefan/Pub/discont.html`.

Nerbonne, John. 1986. 'Phantoms' and German fronting: Poltergeist constituents. *Linguistics*, 24(5).857–870.

Nerbonne, John. 1994. Partial verb phrases and spurious ambiguities. Nerbonne et al. (1994), 109–150.

Nerbonne, John, Klaus Netter, and Carl Pollard, editors. 1994. *German in Head-Driven Phrase Structure Grammar*. Number 46 in CSLI Lecture Notes. Stanford, CA: CSLI Publications.

Oepen, Stephan and John Carroll. 2000. Ambiguity packing in constraint-based parsing: Practical results. *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, 162–169. Seattle, WA. URL `http://www.aclweb.org/anthology/A00-2022`.

Oepen, Stephan, Dan Flickinger, Jun-Ichi Tsujii, and Hans Uszkoreit, editors. 2002. *Collaborative Language Engineering: A Case Study in Efficient Grammar-Based Processing*. Number 118 in CSLI Lecture Notes. Stanford: CSLI.

Ojeda, Almerindo. 1987. Discontinuity, multidominances and unbounded dependency in generalized phrase structure grammar. Huck and Ojeda (1987).

Partee, Barbara H., Alice ter Meulen, and Robert E. Wall. 1990. *Mathematical Methods in Linguistics*. Springer.

Penn, Gerald. 1999. A generalized-domain-based approach to Serbo-Croatian second position clitic placement. Gosse Bouma, Erhard Hinrichs, Geert-Jan Kruijff, and Richard

Oehrle, editors, *Constraints and Resources in Natural Language Syntax and Semantics*, 119–136. CSLI.

Penn, Gerald and Mohammad Haji-Abdolhosseini. 2003. Topological parsing. *Proceedings of the Tenth Conference of the European Chapter of the Association for Computational Linguistics (EACL-03)*, 283–290. URL `http://www.aclweb.org/anthology/E03-1039`.

Pereira, Fernando and David Warren. 1983. Parsing as deduction. *21st Annual Meeting of the Association for Computational Linguistics*, 137–144. Cambridge, MA. URL `http://www.aclweb.org/anthology/P83-1021`.

Plátek, Martin, Tomáš Holan, Vladimir Kuboň, and Karel Oliva. 2001. Word-order relaxations and restrictions within a dependency grammar. G. Satta, editor, *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT)*, 237–240. Beijing: Tsinghua University Press.

Pollard, Carl, Robert Kasper, and Robert Levine. 1994. Studies in constituent ordering: Towards a theory of linearization in head-driven phrase structure grammar. Unpublished research proposal to the National Science Foundation.

Pollard, Carl and Ivan Sag. 1983. Reflexives and reciprocals in English: An alternative to the binding theory. M. Barton, D. Flickenger, and M. Westcott, editors, *Proceedings of the West Coast Conference on Formal Linguistics*, 189–203.

Pollard, Carl J. 1984. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language*. Dissertation, Stanford University, Stanford, CA.

Pollard, Carl J. and Ivan A. Sag. 1987. *Information-based Syntax and Semantics. Volume 1: Fundamentals*. Number 13 in CSLI Lecture Notes. Stanford, CA: Center for the Study of Language and Information.

Pollard, Carl J. and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. Chicago: University of Chicago Press.

Postal, Paul. 1964. Constituent structure: A study of contemporary models of syntactic description. *International Journal of American Linguistics*, 30(1).part III.

Przepiórkowski, Adam. 2001. arg-st on phrases: Evidence from Polish. Dan Flickinger and Andreas Kathol, editors, *Proceedings of the 7th International Conference on Head-Driven Phrase Structure Grammar*, 267–284. Stanford, CA: Center for the Study of Language and Information. URL `http://cslipublications.stanford.edu/HPSG/1/hpsg00.html`.

Pullum, Geoffrey. 1982. Free word order and phrase structure rules. *Proceedings of the North Eastern Linguistic Society*, volume 12, 209–220.

Rambow, Owen and Aravind Joshi. 1994. A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena. L. Wanner, editor, *Current Issues in Meaning-Text-Theory*. London: Pinter. URL `http://arxiv.org/abs/cmp-lg/9410007`.

Ramsay, Allan M. 1999. Direct parsing with discontinuous phrases. *Natural Language Engineering*, 5(3).271–300.

Reape, Mike. 1989. A logical treatment of semi-free word order and bounded discontinuous constituency. *Proceedings of the Fourth Meeting of the European Association for Computational Linguistics*, 103–110. URL `http://www.aclweb.org/anthology/E89-1014`.

Reape, Mike. 1990. A theory of word order and discontinuous constituency in west continental germanic. Elisabeth Engdahl and Mike Reape, editors, *Parametric Variation in Germanic and Romance: Preliminary Investigations*, DYANA R1.1.A, ESPRIT BR 3175, 25–39. Edinburgh: Centre for Cognitive Science, University of Edinburgh.

Reape, Mike. 1991a. Parsing bounded discontinuous constituents: Generalisations of some common algorithms. Reape (1991b), 41–70.

Reape, Mike, editor. 1991b. *Word Order in Germanic and Parsing*. Centre for Cognitive Science, University of Edinburgh.

Reape, Mike. 1993. *A Formal Theory of Word Order: A Case Study in West Germanic*. PhD thesis., Univ. of Edinburgh.

Reape, Mike. 1994. Domain union and word order variation in German. Nerbonne et al. (1994), 151–197.

Reape, Mike. 1996. Getting things in order. Bunt and van Horck (1996), 209–253. Published version of a Ms. from 1990.

Rentier, Gerrit. 1994. Dutch cross serial dependencies in HPSG. *Proceedings of the 15th International Conference on Computational Linguistics*, 818–822. URL `http://www.aclweb.org/anthology/C94-2130`.

Richter, Frank and Manfred Sailer. 1995. *Remarks on Linearization: Reflections on the Treatment of LP-Rules in HPSG in a Typed Feature Logic*. Master's thesis, University of Tübingen. URL `http://www.sfs.uni-tuebingen.de/~fr/cards/thesis.html`.

Richter, Frank and Manfred Sailer. 2001. On the left periphery of German finite sentences. Detmar Meurers and Tibor Kiss, editors, *Constraint-Based Approaches to Germanic Syntax*, Studies in Constraint-Based Lexicalism, 257–300. Stanford, CA: CSLI.

Robinson, Jane J. 1970. Dependency structures and transformation rules. *Language*, 46.259–285.

Sag, Ivan. 1999. Deconstructing grammatical constructions. Ms, Stanford University.

Seiffert, Roland. 1987. Chart-parsing of unification-based grammars with ID/LP-rules. E. Klein and J. van Benthem, editors, *Categories, Polymorphism, and Unification*, 335–354. Edinburgh/Amsterdam: CCS/ILLI.

Seiffert, Roland. 1991. Unification–ID/LP grammars: Formalization and parsing. Otthein Herzog and Claus-Rolf Rollinger, editors, *Text Understanding in LILOG*, number 546 in Lecture Notes in Artificial Intelligence, 63–73. Berlin: Springer Verlag.

Shieber, Stuart. 1985. Using restriction to extend parsing algorithms for complex-feature-based formalisms. *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, 145–52. Chicago. URL `http://www.aclweb.org/anthology/P85-1018`.

Shieber, Stuart M. 1984. Direct parsing of ID/LP grammars. *Linguistics and Philosophy*, 7.135–154.

Siegel, Melanie. 2000. HPSG analysis of Japanese. Wolfgang Wahlster, editor, *Verbmobil: Foundations of Speech-To-Speech Translation*, 265–280. Heidelberg: Springer.

Skut, Wojciech, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP)*. Washington, D.C. URL `http://www.aclweb.org/anthology/A97-1014`.

Suhre, Oliver. 1999. *Computational Aspects of a Grammar Formalism for Languages with Freer Word Order*. Diplomarbeit, Department of Computer Science, University of Tübingen. URL `http://www.sfs.uni-tuebingen.de/sfb/reports/berichte/154/154abs.html`, published 2000 as Volume 154 in Arbeitspapiere des SFB 340.

Tesnière, Lucien. 1959. *Eléments de Syntaxe Structurale*. Paris: Klincksiek.

Tomabechi, Hideto. 1995. Design of efficient unification for natural language. *Journal of Natural Language Processing*, 2(2).23–58.

van der Sloot, Ko. 1989. The TENDUM 2.7 parsing algorithm for DPSG. Unpublished manuscript.

van Noord, Gertjan. 1991. Head corner parsing for discontinuous constituency. *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 114–121. URL `http://www.aclweb.org/anthology/P91-1015`.

van Noord, Gertjan. 1997. An efficient implementation of the head-corner parser. *Computational Linguistics*, 23(3).425–456.

Vogel, Carl M. and Tomaž Erjavec. 1994. Restricted discontinuous phrase structure grammar and its ramafications. Carlos Martin-Vide, editor, *Current Issues in Mathematical Linguistics*, 131–140. Amsterdam: Elsevier Science B.V.

Volk, Martin. 1996. Parsing with ID/LP and PS rules. *Natural Language Processing and Speech Technology. Results of the 3rd KONVENS Conference (Bielefeld)*, 342–353. Berlin: Mouton de Gruyter.

Wells, Rulon S. 1947. Immediate constituents. *Language*, 23(2).81–117.

Yatabe, Shuichi. 1996. Long-distance scrambling via partial compaction. Masatoshi Koizumi, Masayuki Oishi, and Uli Sauerland, editors, *Formal Approaches to Japanese Linguistics 2*, 303–317. Cambridge, MA: MITWPL. URL `http://gamp.c.u-tokyo.ac.jp/~yatabe/fajl.pdf`.

Younger, D. H. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10.189–208.

Zwicky, Arnold. 1986. Concatentation and liberation. *Papers from the 22nd Regional Meeting*, 65–74. Chicago Linguistic Society.

Citation Index

Bach (1983), 1, 91
Blevins (1990), 1, 5, 82, 91
Bloomfield (1933), 3, 91
Bonami et al. (1999), 1, 91
Bröker (1998), 1, 91
Bunt and van Horck (1996), 91, 92, 97
Bunt and van der Sloot (1996), 25, 91
Bunt (1991), 25, 91
Callmeier (2001), 82, 91
Campbell-Kibler (2002), 10, 91
Chomsky (1957), 4, 91
Chung (1993), 18, 91
Copestake (1992), 1, 83, 91
Covington (1990), 22, 24, 92
Covington (1992), 24, 92
Covington (1994), 24, 92
Curry (1961), 33, 92
Daniels and Meurers (2002), 33, 41, 92
Daniels and Meurers (2004a), 33, 41, 92
Daniels and Meurers (2004b), 33, 92
Davis (2002), 44, 92
Donohue and Sag (1999), 1, 15, 92
Dowty (1982), 1, 92
Dowty (1996), 1, 92
Dowty (1997), 14, 92
Drach (1937), 12, 92
Earley (1970), 41, 84, 92
Erbach (1997), 70, 92
Flickinger (2000), 83, 92
Fouvry and Meurers (2000), 32, 93
Fredkin (1960), 89, 93
Götz and Meurers (1995), 1, 93
Götz and Meurers (1997a), 1, 93
Götz and Meurers (1997b), 1, 93
Gazdar and Pullum (1981), 4, 15, 93
Gazdar et al. (1985), 4, 93
Gerdemann (1991), 72, 93
Gerdes and Kahane (2001), 26–28, 30, 93
Grosz et al. (1986), 92–94
Höhle (1978), 20, 93
Haji-Abdolhosseini and Penn (2003), 1, 50, 83, 93
Halpern (1995), 14, 93
Harrison and Ellison (1992), x, 71, 73, 74, 93
Hays (1964), 24, 93
Hepple (1994), 1, 93
Hinrichs and Nakazawa (1990), 17, 93
Hinrichs et al. (2000), 1, 93
Huck and Ojeda (1987), 94, 95
Huck (1985), 1, 94
Huynh (1983), 82, 94
Höhle (1986), 12, 94
Jackendoff (1977), 82, 94
Johnson (1985), 21, 43, 44, 94
Jourdan and Parigot (1990), 51, 94
Kasami and Torii (1969), 42, 94
Kasper et al. (1995), 34, 78, 94
Kathol and Pollard (1995), 10, 94
Kathol (1995), 1, 10, 12, 94
Kathol (2000), 12, 17, 35, 94
Kay (1980), 41, 94
Kiss (1995), 18, 94
Koch (1993), 24, 94
Kroch and Joshi (1987), 1, 94
Lee (1999), 10, 94
Lewis and Papadimitriou (1998), 38, 94
Müller (1997), 18, 95
Müller (1999a), 1, 95
Müller (1999b), 82, 95
Müller (2004), 1, 18, 80, 95
Maekawa (2004), 10, 94
Malouf et al. (2000), 73, 94
McCawley (1982), 1, 4, 95
Meurers (1999), 18, 95
Morawietz (1995), 75, 77, 79, 95
Morrill (1995), 1, 95
Möncke and Wilhelm (1982), 51, 52, 95
Nerbonne et al. (1994), 95, 97
Nerbonne (1986), 17, 95

Nerbonne (1994), 10, 95
Oepen and Carroll (2000), 48, 95
Oepen et al. (2002), 95
Ojeda (1987), 1, 95
Partee et al. (1990), 4, 95
Penn and Haji-Abdolhosseini (2003), 30, 31, 96
Penn (1999), 1, 14, 95
Pereira and Warren (1983), 70, 96
Plátek et al. (2001), 1, 96
Pollard and Sag (1983), 6, 96
Pollard and Sag (1987), 6, 96
Pollard and Sag (1994), 1, 96
Pollard et al. (1994), 1, 17, 96
Pollard (1984), 1, 6, 14, 96
Postal (1964), 4, 96
Przepiórkowski (2001), 20, 96
Pullum (1982), 5, 96
Rambow and Joshi (1994), 1, 96
Ramsay (1999), 44, 96
Reape (1989), 1, 6, 96
Reape (1990), 1, 6, 96
Reape (1991a), 22, 44, 96
Reape (1991b), 1, 6, 96
Reape (1993), 1, 6, 97
Reape (1994), 1, 6, 97
Reape (1996), 1, 6, 97
Rentier (1994), 18, 97
Richter and Sailer (1995), 1, 97
Richter and Sailer (2001), 1, 97
Robinson (1970), 22, 97
Sag (1999), 16, 97
Seiffert (1987), 74, 97
Seiffert (1991), x, 74, 76, 97
Shieber (1984), 5, 97
Shieber (1985), 69, 71, 97
Siegel (2000), 83, 97
Skut et al. (1997), 1, 97
Suhre (1999), 31, 33, 36, 44, 82, 83, 97
Tesnière (1959), 22, 24, 97
Tomabechi (1995), 72, 97
Vogel and Erjavec (1994), 25, 98
Volk (1996), 44, 84, 85, 98
Wells (1947), 3, 98
Yatabe (1996), 10, 98
Younger (1967), 42, 98
Zwicky (1986), 7, 98
van Noord (1991), 21, 22, 97
van Noord (1997), 81, 97
van der Sloot (1989), 25, 97