# Discontinuous Grammar

## A dependency-based model of human parsing and language learning

*Matthias Buch-Kromann*

Copenhagen Business School
HANDELSHØJSKOLEN

# Discontinuous Grammar. A dependency-based model of human parsing and language learning

Matthias Buch-Kromann
*Copenhagen Business School*

This thesis has been accepted for public defence at the Faculty of Language, Communication and Cultural Studies of the Copenhagen Business School for the degree of *doctor linguae mercantilis* in conformity with Ministerial Order No. 750 of 14 August 1996 of the Danish Ministry of Science, Technology and Innovation.

Copyright © 2002–2006 by Matthias Buch-Kromann <mtk.id@cbs.dk>.

This thesis should not be copied or distributed without the written consent of the author. The thesis was submitted as a dr.ling.merc. dissertation to the Copenhagen Business School in December 2005, and assessed by a committee consisting of Prof. Per Anker Jensen (Copenhagen Business School), Prof. Joakim Nivre (University of Växjö), and Prof. Hans Uszkoreit (Saarland University). The thesis was printed with minor modifications in July 2006 by Frederiksberg Bogtrykkeri in 30 copies.

The manuscript was typeset in Palatino with LaTeX2e. The figures were created with the PostScript library PSMath and the treebank tool DTAG.

The photo on the front cover shows a stabile by Steffen Jørgensen titled "Desargues" because it depicts a geometric configuration attributed to the French mathematician G. Desargues. The mobile can be seen in the library of the Copenhagen Business School at Solbjerg Plads.

**To the memory of my father**
**Hans-Peder Kromann**

ού τοι έτι δηρόν γε φίλης άπὸ πατρίδος αίης
έσσεται, ούδ΄ εί πέρ τε σιδήρεα δέσματ΄ έχησι·
φράσσεται ώς κε νέηται, έπεὶ πολυμήχανός έστιν.

The Odyssey, I, 203–5

*Matthias Buch-Kromann* (born 1972) received an MA in mathematics from the University of Pennsylvania in 1997, and an MA in computational linguistics from the Copenhagen Business School in 1999. He was a Ph.D. student at the Department of Computational Linguistics at CBS from 1999–2002, and became an assistant research professor at the Center for Computational Modelling of Language at CBS in 2002. His research interests are centered around dependency treebanks, parsing, statistical language modelling, and machine translation.

# Contents

# Preface

In my view, the purpose of computational linguistics is to make computational models of how humans speak with each other, and use these models to create machines that can help us with tedious language-related tasks, such as machine translation, speech recognition, and dialogue systems. That is, I view computational linguistics as a cognitive science with important technological applications.

This view may differ slightly from the majority view among computational linguists today. Undoubtedly, most of us would be delighted if our processing models reflected the way humans process language. But in practice, we mostly care about how promising our models seem from an applied point of view, and pay little attention to cognitive aspects, even when psycholinguistic evidence proves our models wrong. For example, we know chart parsers are incompatible with our knowledge about garden-path sentences, but they still play a dominant role in parsing because they seem to work extremely well in practical applications. As long as our models result in good technology, cognitive plausibility isn't much of a concern to us. And frankly, why bother?

The main reason for bothering is that all the engineering problems of our science have been solved once already, in the human brain, and there is a real risk that nature's solution is the only viable solution if we want to solve all the problems in the field. So when our artificial processing models depart from what humans do, we should at least think carefully about the differences: How does our artificial solution differ from nature's solution, and why didn't nature come up with our solution? Did we make some implicit theoretical assumptions that work well in a constrained setting, but fail in a more general setting, thereby hindering long-term progress? Can we find clues about a better solution by trying to reverse-engineer human language processing?

In this dissertation, I will argue that, even with the relatively restricted knowledge we have about human parsing today, taking a principled cognitive and theoretical view will reveal some problematic assumptions about parsing and syntax formalisms, and lead to a revision of our assumptions which can be used to construct parsers and syntax formalisms that are good candidates for cognitive models of parsing and at the same time a promising basis for parsing technology. In particular, I will argue that:

- Rather than viewing parsing as a satisfaction problem, we should view parsing as an optimization problem and think of grammars as utility measures that encode a speaker's preferences with respect to the choice of analyses, possibly defined in terms of probability measures.
- Rather than considering the problem of parsing pre-tokenized, sentence-sized input, we should consider the more general problem of parsing unsegmented, discourse-sized input where almost-linear time complexity seems to be a prerequisite for computational feasibility.
- Rather than looking for parsing and generation algorithms that are guaranteed to find a globally optimal analysis, we should look for almost-linear heuristic parsing and generation algorithms — such as serial parsing with repair based on local search — that usually succeed in finding a globally optimal analysis, but fail for some constructions (eg, garden-paths).
- Rather than ensuring computational feasibility by restricting the linguistic expressiveness of the grammars and the probabilistic language models, we should utilize the computational complexity we gain from a heuristic parsing algorithm to formulate linguistically adequate grammars and probabilistic language models.
- Rather than using phrase-structure based theories, where the edit distance between different analyses of ambiguous phrases makes repair operations unnecessarily complex, we should use dependency-based theories where the edit distance and the complexity is low.
- Rather than ignoring psycholinguistics, we should try to improve our parsing models by comparing them with the psycholinguistic evidence about human language processing.

There are many excellent linguistic theories, and I do not wish to claim that the theory presented here is the only "true" theory — a meaningless claim, since theories are always provisional hypotheses about the real world which can be disproved by our observations, but can never be proven true, cf. Popper (1935/1982) — nor do I wish to claim that my theory is the only interesting or sensible theory. On the contrary, I take the view that the purpose of linguistics is to provide as many different models for human language processing as possible, and let them compete with each other in terms of empirical coverage and accuracy (ie, in terms of how well they predict what we can observe about human language use), and in terms of their usefulness in practical applications. Bad theories will be slowly abandoned, whereas good theories will survive until a better alternative appears. What I do hope to demonstrate is that the theory presented here is an interesting alternative to existing theories, which differs from them in some important ways, and resembles them in many others.

The dissertation is organized as follows: Chapter 1 explains why we believe that human parsing is best viewed as an optimization problem. Inspired by computational and psycholinguistic considerations, it proposes a serial parsing model with repair based on local search, and explains why this parsing model encourages the use of a formalism based on dependency structure rather than phrase structure.

Chapter 2 describes the rule-based aspects of Discontinuous Grammar (DG). It describes the lexical rules that define our space of possible analyses, and describes how lexemes can be organized in an inheritance lexicon to avoid redundancy. It also develops the mechanisms that are needed to model lexical transformations, complements and adjuncts, landing sites and word order, and filler constructions and gapping coordinations.

Chapter 3 describes how the formalism assigns costs to linguistic analyses. It shows how primitive linguistic notions can be formalized by means of a vocabulary of cost operators which can be combined to express complex constraints on word order, obligatoriness, agreement, islands, selectional restrictions, word distances, and extraction path lengths. It shows how cost measures can be viewed as generalizations of OT constraints and probabilistic constraints, and how cost measures can be interpreted from a cognitive and linguistic point of view. Finally, it proves that, given a linguistically plausible set of assumptions about idealized speakers, a speaker's preference ordering on the space of all possible analyses can always be expressed by means of a utility measure.

Chapter 4 demonstrates how the syntax formalism can be used to account for a wide range of syntactic phenomena. The phenomena include dependencies and functor-argument structures; word order phenomena such as topicalizations, extrapositions, and scramblings; filler phenomena such as control constructions, relatives, and parasitic gaps; coordination phenomena such as sharing and gapping coordinations; discourse structure and anaphora; punctuation phenomena; and lexical transformations such as inflections, expletives, and passives. We also outline our work on the Danish Dependency Treebank.

Chapter 5 introduces a new statistical estimation technique, called HPM and XHPM estimation, which is suited to the estimation of probability distributions for random vectors with associated classification hierarchies. The HPM and XHPM models function as corrections to a prior distribution, which ensures that the prior distribution is used if there are few data, and that probability mass is moved from low-probability regions to high-probability regions if there is statistically significant evidence for it, where regions are defined as set differences between classes in the hierarchies. The method is tested by means of a simulation study.

Chapter 6 describes how to construct probabilistic language models for DG from classification hierarchies and treebanks, using a mixture of HPM and XHPM distributions, and other statistical distributions. The chapter introduces a formal language for specifying probabilistic language models and describes how their parameters are computed from a treebank. It then proposes a generative probabilistic language model for Discontinuous Grammar with detailed submodels for landing sites and word order, complements and adjuncts, filler generation and filler consumption, gapping coordinations, deep roots, antecedents, and punctuation marks.

Chapter 7 describes our serial parsing model with repair. It outlines how the basic parsing algorithm can be extended to deal with segmentation and multi-speaker dialogue. It presents different families of parsing operations and their computational complexity, and identifies a subset of parsing operations that is both computationally efficient and psycholinguistically plausible. It discusses the effect of island constraints on the computational efficiency of the parsing algorithm, and how the efficiency can be improved by means of error-driven parsing combined with efficient search strategies for each parsing step. We also show that with some refinements, the parsing model is compatible with the psycholinguistic evidence about head-driven parsing. Finally, we describe the steps we have taken towards evaluating the parser.

In the conclusion, we summarize our findings and discuss how far we have reached the goals set out in the beginning, and what is still missing. Our detailed probabilistic DG language model is included in Appendix A. The Danish summary in Appendix B is a translation of the conclusion.

Although the dissertation to some degree represents a rethinking of the underlying assumptions about the design of parsers and syntax formalisms, it also borrows heavily from the ideas of researchers in various frameworks — so much that the dissertation is perhaps best described as a rearrangement of other people's ideas. This indebtedness can be illustrated by a rough outline of the history of the dissertation.

The dependency formalism originally grew out of a desire in 1998 to combine the formal well-definedness of HPSG (Pollard and Sag 1994) with Helbig's (1971/1969) lexicographically oriented school of dependency theory. The cognitive parsing model grew out of a desire in 1999 to model human parsing, and was first based on self-inspection, but later complemented by advice from psycholinguists Patrick Sturt and Don C. Mitchell in 1999–2000. In 2000, I discovered that many of my ideas on syntax had already been proposed by Richard Hudson (2003) within Word Grammar, and my conception of syntax has since been greatly influenced by him.

In 2001, OT (Tesar 1995) and PCFGs filled a gap in my parsing algorithm by providing the insight that parsing is an optimization problem. Coupled with a book on combinatorial optimization by Papadimitriou and Steiglitz (1982), this allowed me to recast my parsing algorithm as an instance of local search and invent cost operators as a way to formalize my linguistic vocabulary and express linguistic constraints formally. My notion of constraints deviated from HPSG and OT, but was strongly inspired by the emphasis on constraints in these theories.

In 2002, my colleagues Daniel Hardt and Peter Juel Henrichsen inspired me to look at problems related with elliptic coordinations and anaphora, spoken language recognition, and machine learning. The development of the Danish Dependency Treebank (Kromann et al. 2003) in 2002–2003, made necessary by the need to build linguistic resources at my department and test my parser, further prompted me to look for ways of inducing a lexicon from a treebank. Here, I received great inspiration from the literature on machine learning and statistics. Thus, the dissertation would have been impossible without the ideas I have borrowed from a lot of other people.

In the text, I have tried to state references to all the researchers and frameworks that have influenced me directly. Whereever possible, I have also tried to mention proposals in the literature that have not influenced me directly, but which either resemble my proposals or differ from them in an interesting way. However, the reader should be warned that the literature is so vast that only a historian of computational linguistics can sort out everything, and that my citations are therefore necessarily incomplete.

A number of researchers deserve mention because I have discussed my work with them at some point, and they have suggested small or large ideas that have somehow found their way into this dissertation. They include Jens Allwood, Nicolas Asher, Niels Davidsen-Nielsen, Anders Frankild, Daniel Hardt, Peter Juel Henrichsen, Richard Hudson, Per Anker Jensen, Kyle Johnson, Aravind Joshi, Stig W. Jørgensen, Yuki Kamide, Ronald Kaplan, Sabine Kirchmeier-Andersen, Alex Klinge, Line H. Mikkelsen, Don C. Mitchell, Joakim Nivre, Barbara H. Partee, Christopher Potts, Geoffrey K. Pullum, Owen Rambow, Maribel Romero, Ivan Sag, Gerold Schneider, Peter Rossen Skadhauge, Patrick Sturt, Carl Vikner, Sten Vikner, and Bjarne Ørsnes. I am grateful to Line H. Mikkelsen and Stine Kern Lynge for their contributions to the Danish Dependency Treebank, and to my former Ph.D. advisors Carl Vikner and Sabine Kirchmeier-Andersen for their advice. I am particularly grateful to Daniel Hardt, who took the time to read through large parts of the manuscript and provide valuable criticism.

The Danish Research Council for the Humanities deserves thanks for supporting me with a Ph.D. grant in 1999–2001 and with a grant that covers my employment as an assistant research professor in 2004–2006. The Department of Computational Linguistics and the president of the Copenhagen Business School, Finn Junge Jensen, deserve thanks for supporting me as an assistant research professor in 2002–2003.

I would also like to extend my thanks to Bernhelm Booß-Bavnbek, Lars Kadison, and Richard Kadison for their lasting influence on me during my formative years as a mathematician, and to my father for introducing me to linguistics long before any of us knew I would end up as a linguist. Finally, I would like to thank my family, my mother, and especially my wife Tine and my son August for their unwavering patience, support, and advice.

*Matthias Buch-Kromann*
*Copenhagen, July 2006*

# Part I
# Introduction

# Chapter 1

# Motivation and overview

We explain why human parsing is best viewed as an optimization problem, and argue that human parsing is based on heuristic rather than exact algorithms. Inspired by computational and psycholinguistic considerations, we propose a serial parsing model with repair based on local search, and explain why this parsing model works better with dependency analyses than with phrase structure analyses.

## 1.1 Human communication as an optimization problem

***Summary.*** *We describe how speakers and hearers are faced with the problem of pairing meanings and speech signals in a way that optimally facilitates communication, and appeal to utility theory to argue that the underlying speaker and hearer preferences can be modelled by means of a real-valued utility measure. Based on this insight, we argue that human parsing and generation are best viewed as optimization problems. In a historical note, we contrast our utility-based view with the classical Chomskyan view of grammars.*

Both verbal and non-verbal human communication can be viewed as a sequence of exchanges between speakers and hearers via sound, text, or gesture. In each exchange, a speaker uses speaker language $L$ and speaker context $C$ to encode intended meaning $M$ as speech signal $S$. The speech signal $S$, after being distorted by noise, is then perceived as speech signal $S'$ by a hearer, who tries to guess the speaker's intended meaning by using hearer language $L'$ and hearer context $C'$ to construct a perceived meaning

$M'$. The exchange is shown schematically below:

$$M \xrightarrow[L,C]{\text{production}} S \xrightarrow{\text{noise}} S' \xrightarrow[L',C']{\text{understanding}} M'$$

The hearer misunderstands the speaker, causing the communication to fail, if the perceived meaning $M'$ differs substantially from the intended meaning $M$. Many factors contribute to the risk of misunderstanding. First of all, communication is difficult if speaker and hearer have languages or perceptions about contexts that differ fundamentally from each other, or if the speech signal is distorted too much by noise. However, even in an ideal world without noise where speaker and hearer are assumed to share the same language and the same perceptions about contexts, communication remains difficult because of the inherent ambiguity of language, which allows speakers to express a meaning and hearers to interpret a speech signal in many different ways. Communication therefore involves a substantial amount of guessing and uncertainty: speakers must try to find the speech signal $S$ that they think best conveys the meaning $M$ to the hearers, given speaker language $L$ and context $C$ (the problem of *production* or *generation*); similarly, hearers must try to find the meaning $M'$ that they think fits best with the noisy speech signal $S'$ originating from the speaker, given hearer language $L'$ and context $C'$ (the problem of *understanding* or *parsing*).

To formalize the problem slightly, we define an *analysis* as a tuple $A = (S, M, C)$ where $S$ is a speech signal, $M$ is a meaning, and $C$ is a context. With this terminology, we can say that hearers and speakers are faced with identical problems of finding an analysis $A = (S, M, C)$ for speakers or $A' = (S', M', C')$ for hearers that best facilitates the communication while being compatible with the known information $(M, C)$ for speakers and $(S', C')$ for hearers. In order to find the analysis that seems best in terms of facilitating the communicative goals, hearers and speakers must obviously be capable of comparing different analyses with each other. Their preferences in this respect can be modelled as complete orderings (cf. Definition 3.13) on the set of all analyses of the form $A = (S, M, C)$, with either fixed $(M, C)$ as speakers, or fixed $(S, C)$ as hearers.

Preference orderings can be rather complicated, and it is difficult to say anything interesting about their properties in general, except that they allow us to compare arbitrary objects in a consistent way. However, there are subclasses of preference orderings whose properties are well-known, and measurable utilities (cf. Definition 3.15) stand out as the most important and best understood example. A measurable utility is a linear function $u$ that maps a totally ordered space $\mathcal{S}$ into the real numbers $\mathbb{R}$ in an order-preserving manner, ie, $u(a) \geq u(a')$ if and only if $a \succsim a'$. Thus, a measurable utility defines preferences in absolute rather than relative terms. Preference orderings and measurable utilities have been the subject of intense study because of a famous theorem by John von Neumann and Otto Morgenstern (1944). The theorem (which we will return to in section 3.4) states that if $\succsim$ is a preference ordering on a set $\mathcal{S}$, then there exists a measurable utility $u \colon \mathcal{S} \rightarrow \mathbb{R}$ for $\succsim$ provided $\mathcal{S}$ is a so-called mixture set that satisfies the axioms of utility.

When applied to our model of human communication, the von Neumann-Morgenstern theorem states that speaker and hearer preferences can be modelled in absolute terms by a measurable utility $u$, provided the preferences satisfy the axioms of utility. The discussion of whether speaker and hearer preferences satisfy the axioms of utility is a rather technical discussion, but in section 3.4, we will argue that it is reasonable to assume that they do — ie, we will formulate a linguistically reasonable model of an idealized speaker, and show that this model satisfies the axioms of utility.

We will interpret a negated measurable utility $c = -u$ as a linguistic *ill-formedness measure* or *cost measure* that quantifies how ill-formed an analysis is in terms of the speaker's or hearer's language. That is, it quantifies the degree to which the analysis deviates from the perceived phonological, morphological, syntactic, semantic, and pragmatic conventions of the language.

So far, we have assumed that a person's speaker and hearer preferences only allow us to compare analyses $A_1 = (S_1, M_1, C_1)$ and $A_2 = (S_2, M_2, C_2)$, with either fixed $(M_1, C_1) = (M_2, C_2)$ as speaker, or fixed $(S_1, C_1) = (S_2, C_2)$ as hearer. This is rather inconvenient, because it gives us a large number of unrelated ill-formedness measures: one for each possible $(M, C)$ and $(S, C)$. From a theoretical point of view, it is more convenient to assume that a person has one large ill-formedness measure which can be used to compare analyses with arbitrary speech signals, meanings, and contexts, in both production and understanding. We therefore make the following two simplifying assumptions:

- **Totality assumption.** A person is capable of comparing arbitrary analyses with each other, ie, the preference ordering $\succsim$ is defined on the entire set $\mathcal{S}$ consisting of all analyses $(S, M, C)$, and for all possible $(S, M, C)$ and $(S', M', C')$ in $\mathcal{S}$, the person's corresponding ill-formedness measure $c$ satisfies $c(S, M, C) \leq c(S', M', C')$ if and only if $(S, M, C) \succsim (S', M', C')$.

- **Uniqueness assumption.** A person has exactly one ill-formedness measure $c$, which is used for both production and understanding.

With the addition of these assumptions, our linguistic model allows ill-formedness measures to be used to compare unrelated linguistic exchanges with each other (similar to what language teachers do when they use a grade to quantify the linguistic ability of students who have written essays on different topics). However, the model makes few assumptions in most other respects. In particular, it does not assume that a person's ill-formedness measure cannot differentiate between different languages, dialects, or stylistic registers,[1] or that different speakers of the same language cannot have slightly different ill-formedness measures.[2]

With these two assumptions and the resulting reformulation of preference orderings in terms of ill-formedness measures, we can now model human generation and parsing as mental processes where speakers and hearers try to find an analysis $A = (S, M, C)$ where the ill-formedness measure is minimal, given a fixed $(M, C)$ for speakers and a fixed $(S, C)$ for hearers. In the terminology of computer science, this means that we have characterized human generation and parsing as optimization problems.

**Definition 1.1** An *optimization problem* is a computational problem where we are given a *search space* $\mathcal{S}$ and a real-valued *cost measure* $c\colon \mathcal{S} \to \mathbb{R}$, and must find an *optimal solution* (or *minimal-cost solution*) in $\mathcal{S}$, ie, a point $s_{\text{opt}}$ in $\mathcal{S}$ with the property that $c(s_{\text{opt}}) \leq c(s)$ for all $s \in \mathcal{S}$ (cf. Papadimitriou

[1]If language, dialect, and stylistic register is considered to be a parameter that is encoded in a person's context, an appropriate ill-formedness measure will be capable of detecting inconsistencies between the context, the meaning and the speech signal with respect to these parameters.

[2]Indeed, it would be surprising if two persons had totally identical ill-formedness measures: Just as people speak different dialects and sociolects, differences between people's grammaticality judgments suggest that they also speak different idiolects. Moreover, there is no a priori reason to assume that different people encode meanings and speech signals in completely identical ways: given the partly random growth process of the human brain (cf. Kandel et al. 1991, p. 943), slight incompatibilities in the way different brains store linguistic representations are perfectly conceivable.

and Steiglitz 1982, p. 4). An algorithm that specifies how to find an optimal solution is called an *optimization algorithm*.

In our linguistic model of language users (or *speakers*, as we will refer to them from now on), the search space corresponds to the speaker's *space of all possible linguistic analyses*, the cost measure corresponds to the speaker's *grammar* (construed as the ill-formedness measure that expresses the speaker's phonological, morphological, syntactical, semantic and pragmatic preferences), and the optimization algorithm corresponds to the *processing principles* used by the speaker for parsing or generation. We will assume that the grammar is constructed in such a way that there always is at least one optimal solution, and that if there is more than one optimal solution, then the speaker can choose freely between them. The linguistic distinction between competence and performance corresponds to the algorithmic distinction between *exact algorithms*, which always return a globally optimal solution, and *heuristic algorithms*, which may sometimes return a non-optimal solution.

The search spaces, cost measures, and optimization algorithms used in models of human language processing are obviously theory-dependent. By recasting human parsing and generation as optimization problems and grammars as ill-formedness measures, we have therefore produced a general model of human communication where most of the details remain to be filled in. In particular, we must provide answers to the following questions before we have a complete linguistic model:

- How should we design the *search space*, ie, what is the best way of modelling a speaker's notion of linguistic analyses?
- How should we design the *cost measure*, ie, what is the best way of modelling a speaker's ill-formedness measure?
- And how should we design the *optimization algorithms* for parsing and generation, ie, what is the best way of modelling the linguistic processing performed by speakers and hearers during understanding and production?

Different answers to these questions lead to different linguistic theories. For example, a phrase-structure-based theory may stipulate that the search space consists of the set of all finite phrase-structure trees with labels chosen from a given set of possible words and word classes, and a dependency-based theory may stipulate that the search space consists of all dependency trees with node and edge labels chosen from a given set of possible words and grammatical functions. Linguistic theories can also have more sophisticated notions of analyses that include contexts, semantic representations, etc., resulting in search spaces that are correspondingly more complicated. The way the cost measure and the optimization algorithms are designed are equally important for determining the shape of the resulting theory. In the following chapters, we will explore one particular set of answers, and relate them to answers given by some other theories.

## Historical note

The utility-based view of language presented so far differs from the classical Chomskyan view, which has dominated linguistics for the past 50 years. The classical Chomskyan view is expressed succinctly in the following quote:

> "The fundamental aim in the linguistic analysis of a language *L* is to separate the *grammatical* sequences which are the sentences of *L* from the *ungrammatical* sequences which are not sentences of *L* and to study the structure of the grammatical sequences. The grammar of *L* will thus be a device that generates all of the grammatical sequences of *L* and none of the ungrammatical ones." (Chomsky 1957, p. 13)[3]

In our utility-based view of language — which is partly inspired by the views proposed in Optimality Theory (Tesar 1995; Kager 1999) and probabilistic language models such as Probabilistic Context-Free Grammar (cf. Manning and Schütze 1999, ch. 11), Data-Oriented Parsing (Bod 1998), and probabilistic unification grammars (Brew 1995; Johnson 2003) — parsing and generation are viewed as optimization problems, and grammars are viewed as devices that define a fine-grained preference ordering on the space of all grammatical and ungrammatical analyses. In contrast, Chomskyan theories have nothing to say about ungrammatical analyses and

[3]Chomsky wrote his book partly in opposition to linguists like C.F. Hockett (1955, p.10), who thought that grammaticality represents a linguist's schematized notions of probability, where low probabilities correspond to "ungrammatical," and high probabilities correspond to "grammatical" — ie, in Hockett's view, the well-formedness of an expression could be quantified by a probability (ie, by a special kind of utility). Thus, in many respects, our utility-based view of linguistics can be seen as a return to Hockett's pre-Chomskyan views.

have no built-in preference ordering on the set of grammatical analyses. This means that within a purely Chomskyan theory, parsing and generation can be defined as constraint satisfaction problems.

**Definition 1.2** A *constraint satisfaction problem* is a computational problem where we are given a *search space* $\mathcal{S}$ and a *constraint* $c\colon \mathcal{S} \to \{0,1\}$, and must find a solution to the constraint $c$, ie, a point $s$ in $\mathcal{S}$ with the property $c(s) = 0$. An algorithm that specifies how to find such a solution is called a *constraint satisfaction algorithm*.

Constraint satisfaction problems can be seen as special cases of optimization problems where the cost measure is restricted to the values 0 ("grammatical") and 1 ("ungrammatical"). In terms of speaker preferences, Chomskyan grammars therefore predict that all grammatical analyses are equally good, and that all ungrammatical analyses are equally bad. As a model of speaker preferences, the Chomskyan approach is inadequate because it fails to model the human ability to resolve ambiguities and recover from ungrammatical input. The problem partly persists in many linguistic theories that have taken steps towards a utility-based or probabilistic view, because most of them have maintained the notion that all ungrammatical analyses are equally bad.

In this section, we have argued that human communication can be viewed as a decision problem where speaker and hearer are faced with the problem of choosing a pairing of meaning and speech signal (ie, a linguistic analysis) that best facilitates their communicative goals. We have argued that the linguistic choices of speakers and hearers can be modelled by means of a preference ordering on the set of all analyses, and that this preference ordering can in turn be expressed by means of a measurable utility, if our idealized axiomatic model of human speakers presented in section 3.4 is correct. This means that human parsing and generation must be modelled as optimization problems rather than satisfaction problems. In the following section, we will therefore take a closer look at optimization from a computational point of view.

## 1.2 Optimization from a computational point of view

***Summary**. We introduce the notions of time complexity and* NP*-hardness, and prove that the Discontinuous Dependency-based Optimality Parsing problem is* NP*-hard. We*

*describe how exact and heuristic parsing methods represent different trade-offs between precision and time complexity, and argue for the use of heuristic parsing methods as a way of ensuring reasonable precision while avoding the NP-hardness of formalisms such as HPSG, LFG, DOP, and OT. Finally, we briefly compare different heuristic strategies proposed in the parsing literature, and present the psychological considerations that have motivated our choice of serial parsing with repair as the basis of our parsing method.*

So far, we have characterized generation and parsing as optimization problems, ie, as problems where the speaker must search through a large search space in order to find an analysis which is optimal among the analyses that match a given meaning or speech signal. Optimization problems have been studied intensively in computer science. Before we proceed with specifying a search space and cost measure within our linguistic theory, we should therefore ask how algorithm theory restricts our choice of optimization algorithm, and how our choice of optimization algorithm in turn restricts our choice of linguistic search space and cost measure.

As a preparation for our discussion of optimization algorithms, we will first give a formal definition of the elementary notions in algorithm theory: problems, algorithms, computations, running times, space requirements, and precision (cf. Aho et al. 1987; Papadimitriou and Steiglitz 1982).

**Definition 1.3** A *computational problem* $p$ is a function that returns a *problem instance* $p(i)$ for each *input* $i$ in a set $I$ of possible inputs. An *algorithm* $A$ with inputs $I$ is a function that returns a *computation* $A_i$ with *output* $A_i^{\text{output}}$ for each $i \in I$. The *running time* $A_i^{\text{time}}$ of a computation $A_i$ is the number of elementary machine instructions performed during the computation, and the *space requirement* $A_i^{\text{space}}$ of $A_i$ is the highest number of elementary memory units needed during the computation. The *precision* of an algorithm $A$ with respect to the problem $p$ is the fraction of inputs $i \in I$ for which the output $A_i^{\text{output}}$ is a solution to the problem instance $p(i)$.

In order to compare different algorithms with respect to their time and space efficiency, it is useful to define a measure of an algorithm's time and space requirements. Obviously, an algorithm's time and space requirements vary greatly with the input: for small inputs, computations tend to be fast and require little memory, whereas for large inputs, computations tend to require more time and memory. For this reason, it is convenient to find a way of measuring the size of the input (it is usually chosen to reflect the number of elementary memory units needed to represent the input), and express the time and space requirements of an algorithm as functions of the size of the input. These functions, known as time and space complexities, are defined formally below.

**Definition 1.4** Let $A$ be an algorithm with inputs $I$, and let $\sigma\colon I \to [0, \infty[$ be a function that assigns *size* $\sigma(i)$ to any input $i \in I$. The *(worst-case) time complexity* $A^{\text{time}}_{\sigma}(n)$ of $A$ with respect to $\sigma$ is then defined as the largest processing time for any computation $A_i$ with $\sigma(i) \leq n$, ie, as the function:

$$A^{\text{time}}_{\sigma}(n) = \max_{i \in I \text{ with } \sigma(i) \leq n} A^{\text{time}}_{i}$$

and the *(worst-case) space complexity* $A^{\text{space}}_{\sigma}(n)$ of $A$ with respect to $\sigma$ is defined as the largest space requirement for any computation $A_i$ with $\sigma(i) \leq n$, ie, as the function:

$$A^{\text{space}}_{\sigma}(n) = \max_{i \in I \text{ with } \sigma(i) \leq n} A^{\text{space}}_{i}$$

The way an algorithm is implemented usually affects the time and space requirements of the corresponding computations with a constant factor, which means that the growth of a time or space complexity is often more informative than its numerical value. Moreover, for most algorithms, the time and space complexities are difficult to calculate exactly, so they are usually characterized in terms of asymptotic upper or lower bounds, rather than being stated exactly. For these reasons, we introduce the notation below to state upper and lower bounds for the asymptotic growth of time and space complexities (cf. Papadimitriou 1994, p. 5).

**Definition 1.5** Let $f$ and $g$ be functions from $[0, \infty[$ to $[0, \infty[$. We write $f = O(g)$ and say that $g$ is an *asymptotic upper bound for the growth* of $f$ if there exists constants $c > 0$ and $n_0 > 0$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$. We write $f = \Omega(g)$ and say that $g$ is an *asymptotic lower bound for the growth* of $f$ if there exists constants $c > 0$ and $n_0 > 0$ such that $f(n) \geq cg(n)$ for all $n \geq n_0$. Finally, we write $f = \Theta(g)$ and say that $f$ has *asymptotic growth* $g$ if $f = O(g)$ and $f = \Omega(g)$.

With these formal definitions at our disposal, we are now ready to return to our discussion of optimization problems.

Optimization problems have a prominent place in computer science, both because a large number of important everyday problems can be described as optimization problems, and because many of the hardest and most interesting problems in computer science are optimization problems. The most famous example is the Travelling Salesman Problem (TSP), where the problem is to find the shortest tour connecting a given set of cities. Finding an algorithm that solves TSP, or any other optimization problem with a finite search space, is easy, since we can always use a *brute-force search* where we try all points in the search space one by one, computing their cost and keeping track of the best solution. After searching through the entire search space, the recorded solution will be globally optimal.

Unfortunately, brute-force search is very inefficient in terms of running time. For example, in a TSP problem with $n$ cities, there are $\frac{1}{2}(n-1)!$ different tours, so that 10 cities correspond to 3.6 million tours, and 20 cities correspond to $2.4 \cdot 10^{18}$ tours. This means that brute-force search has time complexity $\Theta((n-1)!)$, ie, brute-force search is too slow to be used in practice for all but the smallest $n$. For this reason, the time complexity of an algorithm is as important as its ability to produce the correct results.

In algorithm theory, computational problems are divided into complexity classes according to the time complexity of the algorithms needed to solve the problems. The most important classes are P and NP, defined below (cf. Papadimitriou and Steiglitz 1982, p. 347–348).

**Definition 1.6** A *recognition problem* is a problem where every problem instance is a yes-no question. The class P of *(deterministic) polynomial-time recognition problems* consists of all recognition problems where a solution can be computed by an algorithm with time complexity bounded by a polynomial. The class NP of *non-deterministic polynomial recognition problems* consists of all recognition problems with the property that for each "yes" instance, there must exist a *certificate* with polynomially bounded length which can be used to verify the "yes" answer in polynomial time.

**Example 1.7** The problem of determining whether a given string is generated by a given context-free grammar can be solved in $O(n^3)$ time with a chart parsing algorithm, so this recognition problem belongs to P. As an example of an NP problem, we can consider TSP reformulated as a recognition problem, where we must determine whether there exists a tour with length $\leq d$ for some given distance $d$. This version of TSP is in NP, for if the answer to the question is "yes," then the answer can be verified in polynomial time using any tour with length $\leq d$ as the certificate.

The class P is obviously a subset of NP. Although it has never been proven that P is a proper subset of NP, the class NP contains a large number of difficult computational problems with great practical importance (such as TSP) for which no polynomial-time algorithm has yet been found, despite persistent efforts by brilliant researchers. It is therefore widely believed that P $\neq$ NP, and that problems like TSP can only be solved with exponential-time algorithms (cf. Papadimitriou and Steiglitz 1982, p. 343).

In algorithm theory, we are often interested in comparing the hardness of two problems. Intuitively, a problem is as hard as another problem if an instance of the second problem can be transformed into an instance of the first problem by means of a polynomial-time algorithm. This intuition is formalized below.

**Definition 1.8** Let $p$ and $p'$ be recognition problems with inputs $I$ and $I'$, respectively. An algorithm $T$ which for each $i \in I$ computes some $T(i) \in I'$ is called a *transformation* of $p$ to $p'$ if $T$ has the property that $i$ is a "yes" instance of $p$ iff $T(i)$ is a "yes" instance of $p'$. A transformation $T$ is called *polynomial* if $T$ has polynomial time complexity.

Transformations allow us to define the crucial notions of NP-completeness and NP-hardness. Intuitively, NP-complete and NP-hard problems have the property that they are at least as hard as any problem in NP.

**Definition 1.9** A problem $p$ is called NP-*hard* if every problem in NP can be polynomially transformed to $p$. NP-hard problems that belong to NP are called NP-*complete*.

The existence of NP-complete problems is far from obvious, and it is one of the great achievements of algorithm theory to have shown that NP-complete problems actually exist, and that the group of NP-complete problems includes many well-known problems, such as the recognition version of TSP (cf. Papadimitriou 1994, p. 397). The standard method for proving that a problem is NP-hard is to prove that a known NP-hard problem can be polynomially transformed into the new problem, ie, if we can solve the new problem, we can solve the known NP-hard problem and hence all problems in NP.

With these tools at hand, we are now ready to prove that unrestricted discontinuous dependency-based optimality parsing is NP-hard. We start by defining a highly simplified dependency formalism. If we can prove that parsing in the simplified formalism is NP-hard, then it follows that parsing in any dependency formalism which has the simplified formalism as a special case is NP-hard as well. In particular, the syntax formalism Discontinuous Grammar presented in the following chapters has the simplified formalism as a special case, so it is NP-hard as well.

**Definition 1.10** A *simplified dependency tree* is a tree whose nodes represent words, and whose labeled directed edges represent dependency relations between words. A *simplified weighted dependency grammar* is a function $W$ that for each local dependency tree $t$ computes a weight $W(t)$ in $[0, \infty]$, where the weight $\infty$ indicates that the local tree is ill-formed. The *total weight* $W(T)$ of a dependency tree $T$ is defined as the sum of the weights for all local trees, ie:

$$W(T) = \sum_{t \in L(T)} W(t)$$

where $L(T)$ is the multi-set consisting of all local dependency trees in $T$.

As we have argued in section 1.1, parsing should be viewed as an optimization problem. In particular, dependency parsing can be characterized as the following optimization problem:

**Definition 1.11** In the *Discontinuous Dependency-based Optimality Parsing problem* (*DDOP*), we are given a weighted dependency grammar $W$ and a string $s$ of words, and must find a dependency tree $T$ with string $s$ that is optimal with respect to the grammar $W$, ie, which minimizes $W(T)$.

It turns out that discontinuous dependency parsing is as hard as TSP, ie, it is a highly demanding computational problem.[4]

[4]Neuhaus and Bröker (1997) have proven the NP-hardness of discontinuous dependency-based satisfaction parsing (DDSP). Their proof is more general, but also more complicated than the proof presented here.

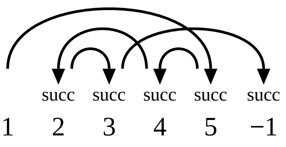


Figure 1.1: The dependency tree corresponding to the tour $(1, 5, 4, 2, 3)$.

**Theorem 1.12** *The Discontinuous Dependency-based Optimality Parsing problem is* NP*-hard.*

*Proof.* The proof proceeds by showing that TSP can be transformed into DDOP. In TSP, we have cities $C = \{1, \ldots, n\}$ and distance function $D \colon C \times C \rightarrow [0, \infty[$. We must construct a transformation from TSP to a weighted dependency grammar $W$ and a string $s$ with the property that any optimal dependency tree for $s$ corresponds to an optimal tour.

Since a tour is a simple cycle which must include all cities, we can assume without any loss of generality that a tour starts and ends with the city 1. In order to represent a cyclic tour by means of a non-cyclic dependency tree, we will introduce the special city name $-1$ to represent the city 1 viewed as the end-point of the tour, so that any tour can be represented as a non-cyclic path $(1, T(2), \ldots, T(n), -1)$, where $T$ is some permutation of $\{2, \ldots, n\}$. This path can in turn be translated into a (possibly discontinuous) linear dependency tree with nodes $\{1, \ldots, n, -1\}$ where successive nodes on the path are indicated by a "succ" dependency from the previous node to the successor node. Figure 1.1 exemplifies the procedure by showing the dependency tree corresponding to the tour $(1, 5, 4, 2, 3)$.

As our input string, we use $s = (1, 2, \ldots, n, -1)$. As our dependency grammar $W$, we define:

$$W(t) = \begin{cases} D(i,j) & \text{if } t = i \xrightarrow{\text{succ}} j \text{ with } 1 \leq i \leq n \text{ and } 2 \leq j \leq n \\ D(i,1) & \text{if } t = i \xrightarrow{\text{succ}} -1 \text{ with } 1 \leq i \leq n \\ 0 & \text{if } t = -1 \\ \infty & \text{otherwise} \end{cases}$$

The grammar $W$ ensures that 1 must be a root node (ie, 1 cannot be the "succ" dependent of any other node), and that $-1$ must be a terminal node (ie, $-1$ cannot take any other node as its "succ" dependent). The linearity of the tree is ensured by requiring any node $1, \ldots, n$ to have exactly one "succ" dependent. Moreover, the weight assignment ensures that the weight assigned to the dependency tree equals the distance of the corresponding tour. This gives a one-to-one correspondence between tours and well-formed dependency trees where an optimal dependency tree corresponds to an optimal tour. Since the transformation from TSP to DDOP can be performed in linear time, this proves that TSP can be polynomially transformed into DDOP, and hence that DDOP is NP-hard. □

The dependency trees used in the proof may seem implausible from a linguistic point of view, but it is in practice difficult to constrain discontinuity enough to ensure polynomial-time parsing without sacrificing the linguistic expressiveness of the formalism. Dikovsky (2001) has shown that it is possible to construct a restricted discontinuous dependency formalism that leads to polynomial-time parsing if parsing is viewed as a satisfaction problem, but it is not clear that there exists a linguistically adequate dependency formalism with polynomial-time parsing if parsing is viewed as an optimization problem.

The NP-hardness of DDOP is caused by the potential discontinuity of the dependency tree, since the corresponding continuous version of DDOP can be solved with a standard probabilistic chart-parsing algorithm with time complexity $O(n^3)$, and hence cannot be NP-hard.

**Definition 1.13** In the *Continuous Dependency-based Optimality Parsing problem* (*CDOP*), we are given a dependency-grammar $W$ and a string $s$ of words, and must find a continuous dependency tree $T$ with string $s$ that is optimal with respect to the grammar $W$, ie, which minimizes $W(T)$.

The reason why discontinuous dependency parsing (DDOP) is so much harder than continuous dependency parsing (CDOP) is perhaps best understood by considering why chart parsing algorithms work for CDOP, but fail for DDOP. The success of chart parsing for CDOP (and PCFG parsing in general) is caused by the fact that a string with $n$ words has $\frac{1}{2}n(n+1)$ different continuous substrings, so that we can use dynamic programming and create a chart containing the optimal analyses of all continuous substrings in $O(n^3)$ time. In contrast, in the unrestricted discontinuous case, there are $2^n - 1$ non-empty discontinuous substrings, so a chart containing the optimal analyses of all discontinuous substrings must contain exponentially many entries, which results in a chart parsing algorithm with exponential worst-case time complexity. The NP-hardness of DDOP compared to CDOP can therefore be viewed as a consequence of the combinatorial explosion caused by the discontinuity of DDOP.[5]

In the simplified version of dependency grammars presented so far, we have restricted ourselves to local cost measures where the cost of a local tree only depends on the parent node and the child nodes, and not on all other nodes in the dependent trees. This is a very restrictive assumption about the cost measure, and without this assumption, it is quite conceivable that even CDOP becomes NP-hard.

Most linguistically interesting syntax formalisms lead to NP-hard parsing, either because of their linguistic expressivity, or because they use optimization-based parsing with a complex cost measure. Thus, NP-hardness results have been proven both for linguistically expressive syntax formalisms with a binary notion of grammaticality where parsing is viewed as a satisfaction problem — such as LFG (Maxwell and Kaplan 1993), unification-based formalisms like HPSG (Trautwein 1994), and discontinuous dependency grammar (Neuhaus and Bröker 1997) — as well as for syntax formalisms where parsing is viewed as an optimization problem with a complex cost measure — such as Optimality Theory with unrestricted constraints (Wareham 1998), and Stochastic Tree-Substitution Grammars and Data-Oriented Parsing (Sima'an 1996). PCFG and TAG are examples of syntax formalisms where parsing is polynomial, since they can be parsed in $O(n^3)$ and $O(n^6)$ time, respectively, but they are not based on optimization, and there are linguistic phenomena that they cannot properly account for (cf. Abeillé and Rambow 2000, p. 31–35).[6]

The formalisms can be classified according to whether they view parsing as a satisfaction problem or an optimization problem, and whether they are discontinuous or continuous (ie, whether they are expressive enough to allow grammars that can generate all discontinuous dependency trees), as

[5]It can be calculated by means of so-called *Prüfer symbols* that a string with $n$ words has $n^{n-1}$ possible dependency trees excluding functional labels, and $(kn)^{n-1}$ possible dependency trees including functional labels, where $k$ is the number of possible function labels (cf. Biggs 1989, p. 204).

[6]Chart parsers for CFG and TAG create a packed chart that may contain an exponential number of parses (e.g., consider the grammar $X \rightarrow X\ X$ with terminal rule $X \rightarrow x$). Thus, given a sufficiently linguistically sophisticated cost measure on top of CFG and TAG (say, a measure that checks semantic and pragmatic plausibility), it is quite conceivable that the corresponding optimality parsing problem in CFG and TAG will turn out to be NP-hard.

| | Satisfaction problem | Optimization problem |
|---|---|---|
| **Continuous formalism** | polynomial: CFG, TAG | polynomial: PCFG, CDOP<br>NP-hard: STSG, Tree-DOP |
| **Discontinuous formalism** | NP-hard: HPSG, LFG, DDSP | NP-hard: OT, LFG-DOP, DDOP |

Figure 1.2: Computational complexity of different syntax formalisms.

summarized in Figure 1.2. In light of Theorem 1.12, it is no surprise that all existing discontinuous syntax formalisms with optimization are NP-hard. However, it would be wrong to interpret the NP-hardness as evidence that efficient computation is inherently impossible in discontinuous syntax formalisms with optimization, for we need to distinguish between exact and heuristic algorithms, and the NP-hardness of DDOP only applies to exact algorithms.

**Definition 1.14** An optimization algorithm is called *exact* if it is guaranteed to always find a globally optimal solution. An optimization algorithm that is not guaranteed to find a globally optimal solution, but is designed to often result in an optimal or near-optimal solution anyway, is called a *heuristic algorithm*.

Heuristic and exact algorithms represent different trade-offs between precision and computational efficiency, ie, between the need to find a guaranteed globally optimal solution and the need to perform the computations within a reasonable amount of time: exact algorithms tend to be precise but slow, whereas heuristic algorithms tend to be fast but not always precise.

Garden-path sentences such as "The horse raced past the barn fell" (Bever 1970) show that humans sometimes fail to find the best parse of a sentence, thereby providing evidence that human parsing is heuristic rather than exact. From a computational perspective, it is not surprising that nature has chosen this path: the NP-hardness of DDOP shows that a heuristic algorithm may be necessary if we want to escape from exponential worst-case time complexity, ie, that we may have to give up precision in order to ensure speed.[7]

[7]It is sometimes argued that exact parsing in an NP-hard formalism can be reasonably efficient when restricted to "well-behaved" grammars that are carefully crafted to avoid con-

In section 1.1, we have argued that optimization is necessary in order to faithfully model human parsing. In section 1.4, we will argue that a discontinuous formalism is necessary in order to give a linguistically satisfactory treatment of discontinuous constructions in natural language, ie, to ensure good linguistic coverage. In this dissertation, we will therefore argue that human parsing should be modelled by a heuristic algorithm based on a discontinuous formalism with optimization. In order to prove the viability of such an approach, we will propose a specific discontinuous formalism with optimization in chapters 2 and 3. In chapter 7, we will present promising parsing algorithms for discontinuous formalisms with optimization by presenting families of almost-linear heuristic parsing algorithms that return an optimal or near-optimal parse for most constructions, although they may fail to find an optimal solution when presented with garden-path sentences or particularly difficult ambiguities where human parsing is known to often fail.

In natural language parsing, most parsers have been based on exact algorithms, and most existing heuristic parsing models — including the *Marcus parser* (Marcus 1980), the garden-path model (Frazier and Rayner 1982), and most other psycholinguistic parsing models — are designed for syntax formalisms without optimization. The number of heuristic parsing algorithms proposed for discontinuous formalisms with optimization is therefore relatively small. The most important strategies behind heuristic parsers involving at least some degree of optimization are listed below.

- **Monte Carlo models**: Estimate the probability of parses from a sample of randomly generated analyses in Data-Oriented Parsing. (Bod 1995, 1996; Bod and Kaplan 2003)
- **Resource-limited parallel models (beam search)**: Maintain a list of the most promising partial analyses and prune low-probability analyses from the list in order to conserve computational resources. (Staab 1995; Lavie and Tomita 1996; Schneider 2003; Roark 2003)

structions with exponential parsing behaviour. However, this strategy leads to a difficult trade-off between well-behavedness and robustness, since robust handling of speaker errors often requires the use of highly ambiguous grammars that tend not to be well-behaved. It also shifts much of the burden of ensuring efficient parsing to grammar engineers, thus adding to the complexity of their task. These objections aside, researchers like Daniels and Meurers (2002) have made remarkable progress towards efficient implementation of non-optimization based HPSG grammars.

- **Resource-limited constraint satisfaction models (minimal commitment models)**: Use a resource-limited monotonic constraint satisfaction process where low-probability choices are pruned. (Trueswell and Tanenhaus 1994; Sturt and Crocker 1998)
- **Connectionist models**: Model parsing by means of competitive attachment (Stevenson 1994, 1998; Merlo and Stevenson 2000; Stevenson and Smolensky 2005) or competitive inhibition (Vosse and Kempen 2000; Kempen and Harbusch 2002) in a neural network, or by using a recursive neural network to determine probabilities of attachment sites (Costa et al. 2003; Sturt et al. 2003).
- **Serial models without repair**: Construct a parse serially by choosing a non-destructive parse action deterministically at each choice point. (Marcus 1980; Yamada and Matsumoto 2003; Nivre 2003, 2004)
- **Serial models with repair**: Use local search where analyses are constructed incrementally by local structure-changing operations. (Lewis 1998, 1999; Kromann 1999a, 2001; Schulz 2000; Daum and Menzel 2002)

The different strategies have very different properties in terms of how many analyses they store in parallel, how they select a currently optimal analysis among the available analyses, and how they account for garden-path phenomena. The differences are summarized in Figure 1.3. With respect to the number of analyses stored in parallel, Monte Carlo models and parallel models store many analyses in parallel, whereas serial models maintain only one analysis at any given time. Constraint satisfaction models and connectionist models can be seen as a hybrid between these two approaches, since they use a single underspecified representation in order to represent multiple analyses.

In order to provide a currently preferred analysis for semantic interpretation, Monte Carlo models and parallel models select the analysis with the currently highest probability, connectionist models select the analysis induced by the links with the highest activation, and serial models use their unique current analysis. It is not immediately obvious how constraint satisfaction models could produce a currently optimal analysis.

Resource-limited parallel models and resource-limited constraint satisfaction models explain garden-paths by choice points that are pruned away by the parser because they are deemed improbable, although they are in fact necessary for constructing a grammatical analysis. In connectionist

| | **Monte Carlo models** | **Resource-limited parallel models** | **Resource-limited constraint satisfaction** | **Connection-ist models** | **Serial models** |
|---|---|---|---|---|---|
| **Number of analyses stored in parallel** | many | many | underspeci-fied | underspeci-fied | one |
| **Selection of currently optimal analysis** | analysis with highest probability | analysis with highest probability | N.A. | analysis induced by links with highest activation | current anal-ysis |
| **Explanation of garden-paths** | N.A. | pruned low-probability branch | pruned low-probability variable assignment | network stabilizes on wrong analysis | repair distance too large to repair wrong analysis |

Figure 1.3: Differences between different heuristic strategies.

models, garden-paths arise when the network stabilizes on an incorrect analysis, and in serial models, garden-paths arise when the repair distance from the current partial analysis to the correct analysis becomes too large (or, in serial models without repair, when a wrong choice was made at a previous choice point). As far as we know, there has been no attempt to explain garden-paths within a Monte Carlo model.

In this dissertation, we have chosen to model human parsing by means of serial parsing with repair. This choice is not based on any conclusive evidence for serial parsing with repair — on the contrary, the relative merits of the different heuristic strategies are still an open question within psycholinguistics, and there are strong proponents of both connectionist models, parallel models, minimal commitment models, and serial models.[8] However, our preference for serial models can be explained by our

[8]Psycholinguistic studies in favour of serial parsing include Hopf et al. (2003), who describe an ERP study that is argued to provide evidence for serial parsing with repair, and Lewis (2000) who argues for either serial models with repair or ranked parallel models, based on a review of the available evidence. Psycholinguistic studies against serial parsing include Pearlmutter and Mendelsohn (2000), who argue for ranked parallel models without competition and against serial parsing on the basis of a reading time study. It is not entirely clear

perhaps incorrect belief that serial models are more promising than other models in the following respects:

- **Low memory requirements**: Humans are bad at remembering large amounts of data flawlessly, and therefore tend to use problem solving strategies with low memory requirements (cf. Miller 1956). Since serial parsing requires the storage of only one analysis at a time, serial parsing has the smallest possible memory requirements, and only underspecified models have the potential to rival it in this respect.
- **Simple selection of currently optimal analysis**: We believe humans need to maintain a single currently preferred analysis during parsing in order to perform semantic interpretation. Since serial parsing maintains only one analysis at a time, it provides a particularly simple mechanism for selecting a currently preferred analysis, whereas underspecification and multiple parallel analyses necessitate a more complicated selection mechanism.
- **Simple account of repair with multiple levels**: Human parsing seems to involve two levels of repair: a subconscious repair which only leads to a slight delay in parsing, and a conscious repair where the delay is long and the hearer is consciously aware that a processing difficulty has arisen. With serial parsing, this phenomenon can be explained by a hierarchy of increasingly powerful and time-consuming parsing operations that are triggered by the failure of low-level parsing operations. In contrast, parallel models and minimal commitment models cannot immediately restore a choice point after it has been pruned from memory, but need to reparse the sentence with the intervention of some higher-order cognitive process in order to avoid repeating the erroneous pruning. Likewise, connectionist models need to appeal to a repair phase where some higher-order cognitive process intervenes by destabilizing the network and inhibiting the wrong analyses, after the network has stabilized on the wrong analysis and realized that it has become stuck.

These beliefs merely represent our subjective reasons for exploring a serial

which of these studies is right, although we suspect that the intransitive use of transitive verbs in the critical examples in Pearlmutter and Mendelsohn (2000) may be responsible for the effects that they interpret as evidence against serial parsing with repair. In any case, this is an area where further research is needed.

strategy rather than any other heuristic strategy. It should also be noted that a serial strategy is not necessarily incompatible with a connectionist strategy, although the connection may not be straight-forward. Indeed, although it is clear that the models are not totally identical, we suspect that it may be possible to view our serial model as a high-level approximation to a connectionist model like Vosse and Kempen (2000).

In this section, we have argued that parsing with discontinuous syntax formalisms with optimization is inherently NP-hard, and that the best way to ensure computational efficiency and accommodate the psycholinguistic evidence about garden-paths is to use heuristic rather than exact optimization-based parsing algorithms. We have briefly presented different heuristic optimality-based parsing strategies proposed in the literature, and presented our psychologically motivated reasons for preferring serial parsing with repair. In the remainder of this chapter, we will specify our parsing model in more detail, and show how the choice of a serial parser with repair restricts our choice of syntax formalism.

## 1.3 Serial parsing with repair based on local search

***Summary***. *We present local search, an optimization strategy which has been used with great success in many NP-hard optimization problems, including the Travelling Salesman Problem. We explain how the successful application of local search has depended crucially on finding neighbourhoods that are small enough to lead to computational efficiency, but large enough to escape local optima that fail to be global optima. We then outline our ideas for using local search to model human parsing, and sketch how the difficult trade-off between precision and efficiency affects the choice of parsing operations.*

In the previous section, we saw that discontinous dependency-based optimality parsing has many similarities with the Travelling Salesman Problem, one of the most well-known problems in computer science. In TSP, the best heuristic strategies are all based on local search in one form or another (cf. Johnson and McGeoch 1997). Although there is no guarantee that the best algorithms for TSP are also the best algorithms for optimality parsing, it is natural to consider algorithms based on local search when trying to find good algorithms for optimality parsing. Some work has been done in this direction, including (Kromann 2001) and (Schulz 2000; Daum and Menzel 2002). We will follow the strategy outlined in (Kromann 2001).

Negative results have been proven for TSP, and they are important because they tell us what kinds of algorithms we can reasonably hope for. The best solution would be an exact polynomial algorithm, but the NP-hardness of TSP proves that such an algorithm is impossible unless P = NP, which is highly unlikely.[9] The second-best solution would be an $\epsilon$-*approximate polynomial algorithm* (Papadimitriou and Steiglitz 1982, p. 409), ie, a polynomial algorithm that returns a solution whose cost is at most the fraction $\epsilon$ larger than the globally optimal solution. Such an algorithm would not always produce a globally optimal solution, but it would at least give us some knowledge about the worst-case relative error of any solution. Unfortunately, it has been proven that there is no $\epsilon$-approximate polynomial algorithm for the TSP for any $\epsilon > 0$ unless P = NP (Papadimitriou and Steiglitz 1982, p. 427), so this is more than we can reasonably hope for as well. Thus, if we want a polynomial algorithm for solving TSP and the even more general Discontinuous Dependency-based Optimality Parsing problem, we can only hope for a heuristic algorithm where we have no knowledge about the relative error of the solutions.

Local search was made famous when Lin (1965) used it to solve large instances of TSP, and it has often turned out to be the best heuristic algorithm available for many NP-hard problems (cf. Papadimitriou and Steiglitz 1982, p. 481). The basic idea behind local search is to select an initial solution, modify it incrementally with a series of cost-decreasing changes, and return the final solution when no further improvement is possible. Local search is guaranteed to find a local optimum, and in problems that are suited to local search, this local optimum will often be globally optimal.

**Definition 1.15** Let $(\mathcal{S}, c)$ be an optimization problem with search space $\mathcal{S}$ and cost function $c\colon \mathcal{S} \to \mathbb{R}$, and let $N\colon \mathcal{S} \to \mathrm{Pow}(\mathcal{S})$ be a *neighbourhood function* that assigns a neighbourhood to each solution in $\mathcal{S}$. The algorithm for *general local search* shown in Figure 1.4 can be used to compute a heuristic solution $s$ to the optimization problem (cf. Papadimitriou and Steiglitz

[9]The most efficient exact algorithms for solving TSP are based on a branch-and-bound strategy (Papadimitriou and Steiglitz 1982, p. 438), and have been reported to work well for instances of TSP with up to 64 cities (Papadimitriou and Steiglitz 1982, p. 442). Exact algorithms for TSP based on dynamic programming have attained time complexity $O(n^2 2^n)$ — a significant reduction of the time complexity $O((n-1)!)$ attained by a brute-force algorithm, but still exponential. However, these algorithms have been reported to be less efficient in practice than branch-and-bound algorithms (Papadimitriou and Steiglitz 1982, p. 450).

```
**procedure** local search
**begin**
    $s$ := some initial starting point in $\mathcal{S}$;
    **while** improve($s$) $\neq$ 'no' **do**
        $s$ := improve($s$);
    **return** $s$;
**end**
```

Figure 1.4: The algorithm for general local search.

1982, p. 454). The subroutine improve($s$) searches for an improvement $n$ of a solution $s$ within the neighbourhood of $s$, and is given by:

$$\text{improve}(s) = \begin{cases} \text{any } n \in N(s) \text{ with } c(n) < c(s) \text{ if such an } n \text{ exists} \\ \text{'no' otherwise} \end{cases}$$

The subroutine improve($s$) can be implemented in two ways: in a *first-improvement* strategy, the subroutine returns the first solution within the neighbourhood having a smaller cost, thereby avoiding a possibly costly search of the entire neighbourhood, whereas in a *best-improvement* (or *steepest descent*) strategy, the subroutine searches the entire neighbourhood and returns a solution whose cost is minimal within the neighbourhood.

**Definition 1.16** Let $(\mathcal{S}, c)$ be an optimization problem with search space $\mathcal{S}$ and cost function $c\colon \mathcal{S} \to \mathbb{R}$, and let $N\colon \mathcal{S} \to \text{Pow}(\mathcal{S})$ be a neighbourhood function. A solution $s \in \mathcal{S}$ is called *locally optimal* with respect to $N$ and $c$ whenever $c(s) \leq c(n)$ for all $n \in N(s)$. A neighbourhood $N$ is called *exact* if all locally optimal solutions are globally optimal as well.

**Example 1.17** The local search algorithm is illustrated in Figure 1.5. Local neighbourhoods around points in the search space are shown with gray circles. The gray area along a search path therefore represents the part of the search space that is actually searched by the algorithm, whereas the white area represents the part of the search space that is pruned away. The algorithm first selects an initial solution (eg, $A$ or $B$). In each step, it then searches the neighbourhood of the current solution for an improved solution that can be selected as the new current solution, and stops when the

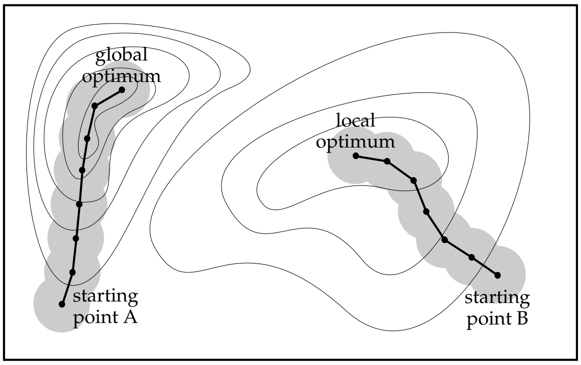


Figure 1.5: An illustration of local search.

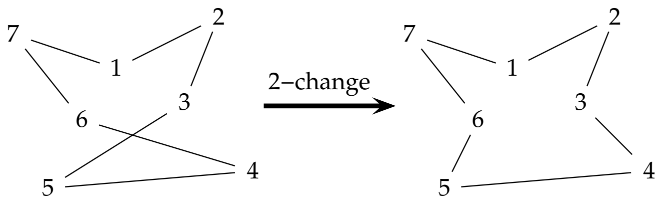


Figure 1.6: Two tours related by a 2-change operation.

current solution is locally optimal. Starting point $A$ results in a globally optimal solution, whereas $B$ only results in a locally optimal solution.

When applying local search to TSP, it is natural to use the $k$-change neighbourhoods defined formally below.

**Definition 1.18** In the Travelling Salesman Problem, the *k-change neighbourhood* of a tour $t$ is the set of all tours that can be obtained from $t$ by removing $k$ edges and replacing them with $k$ new edges.

**Example 1.19** Figure 1.6 shows how the tour $(1,2,3,4,5,6,7)$ is obtained from the tour $(1,2,3,5,4,6,7)$ by a 2-change operation that removes the edges $3 \leftrightarrow 5$ and $4 \leftrightarrow 6$, and replaces them with the edges $3 \leftrightarrow 4$ and $5 \leftrightarrow 6$.

**Remark 1.20** In TSP, the size of a $k$-change neighbourhood grows quickly with $k$. It is difficult to calculate the exact number of tours in a $k$-change neighbourhood, but an upper bound for the TSP with directed tours is given by $(k-1)!\binom{n}{k}$.[10] When $k = n$, the $k$-change neighbourhood contains all $(n-1)!$ different tours.

Lin (1965) empirically demonstrated the power of 3-change neighbourhoods in TSP. He found that 2-change neighbourhoods usually resulted in a suboptimal local optimum, whereas 4-change neighbourhoods were much more computationally expensive than 3-change neighbourhoods, resulting only in a marginal improvement in terms of precision (cf. Papadimitriou and Steiglitz 1982, p. 456). Although optimality parsing differs from TSP in many respects, the general experience from TSP and other hard optimization problems is that the choice of neighbourhoods is crucial for the success of local search. If neighbourhoods are chosen too small, local search may be unable to escape non-global local optima. If neighbourhoods are chosen too large, too much time is spent searching the local neighbourhoods, without significantly affecting the ability to escape non-global local optima.

The insights from TSP and other hard optimization problems should guide our choice of neighbourhoods when applying local search to optimality parsing. A first step in this direction is to note that most syntax formalisms embody a sensible notion of $k$-change neighbourhoods.

**Definition 1.21** In a search space consisting of all dependency graphs, we can define the *k-change neighbourhood* of a dependency graph $d$ as the set of all dependency graphs that can be obtained from $d$ by removing at most $k$ edges, and adding at most $k$ new edges.

**Definition 1.22** In a search space consisting of all phrase-structure graphs, we can define the *k-change neighbourhood* of a phrase-structure graph $p$ as the set of all phrase-structure graphs that can be obtained from $p$ by removing at most $k$ non-terminal nodes or edges, and adding at most $k$ new non-terminal nodes or edges.

[10]Calculation: The tour contains $n$ edges, and there are $\binom{n}{k}$ different ways of deleting $k$ edges. After the deletion, we have a graph with $k$ connected components, which can be ordered in at most $(k-1)!$ different ways, when the component containing city 1 must be selected as the first component. This means that there are at most $(k-1)!\binom{n}{k}$ different tours in the $k$-change neighbourhood.

```
procedure local optimality parsing (LOP)
begin
    $s$ := segmented speech signal;
    $a$ := empty analysis;
    while $s$ is non-empty or improve($a$) $\neq$ 'no' do
        if improve($a$) $=$ 'no' then
            $a$ := partial analysis obtained by appending $s_1$ to $a$;
            $s$ := $s$ minus first segment $s_1$;
        else
            $a$ := improve($a$);
    return $a$;
end
```

Figure 1.7: The LOP algorithm for local optimality parsing.

The parsing algorithm below is an adaptation of local search where the speech signal is represented as a sequence of segments that are appended incrementally to the current analysis in temporal order whenever a local optimum is reached, and where neighbourhoods around analyses are defined by means of parsing operations that modify the analysis in some way — for example, neighbourhoods could be defined by means of $k$-change operations.

**Definition 1.23** Let $\mathcal{A}$ be the set of all partial analyses within some linguistic formalism, let $c\colon \mathcal{A} \to \mathbb{R}$ be the cost measure induced by some grammar, and let $\pi_1, \ldots, \pi_k\colon \mathcal{A} \to \mathrm{Pow}(\mathcal{A})$ be a set of associated parsing operations. The algorithm for *local optimality parsing* (LOP) shown in Figure 1.7 can be used to compute a locally optimal analysis for the segmented speech signal $s = (s_1, \ldots, s_n)$. The subroutine improve($a$) searches for an improvement $a'$ that can be obtained by applying a parsing operation $\pi_i$ to $a$, and is given by:

$$\text{improve}(a) = \begin{cases} \text{any locally optimal } a' \in \bigcup_{i=1}^{k} \pi_i(a) \text{ with } c(a') < \\ c(a) \text{ if such an } a' \text{ exists;} \\ \text{`no' otherwise} \end{cases}$$

Local optimality parsers can be characterized as serial parsers with repair that process speech signals incrementally. These notions are defined formally below.

**Definition 1.24** A parser is called *serial* if it maintains only one analysis at a time. A serial parser that can only add new nodes and edges to its current analysis is called a *serial parser without repair*, and a parser that is also allowed to change nodes and edges in its current analysis is called a *serial parser with repair*. A serial parser is called *incremental* if it does not process a speech segment starting at time $t$ before it has attempted to produce a locally optimal analysis for the entire speech signal up to time $t$.

Most psycholinguistic theories of parsing assume that parsing is incremental, ie, that we interpret sentences as we hear them, without waiting for the completion of the sentence. The requirement of incrementality dictates the local optimality parser's use of the empty analysis as its initial analysis in Algorithm 1.7.

Local optimality parsing is a general algorithm which leads to different parsers depending on the particular choice of parsing operations and underlying analysis space. It is therefore relevant to ask whether certain kinds of analysis space are better suited to local optimality parsing than others, and whether we can identify parsing operations which are neither so weak that the local optimality parser fails on many naturally occurring constructions, nor so powerful that parsing becomes too time-consuming to be computationally realistic. In the following section, we will start by examining the analysis space. In section 7.2, we will then return to how we can design parsing operations that are neither too powerful, nor too weak.

## 1.4 How the choice of analysis space affects parsing

***Summary.*** *We explain how the success of local optimality parsing depends on the choice of analysis space. Based on the notion of edit distance between alternative analyses in syntactic ambiguities, we argue that in a local optimality parser, dependency analyses can be expected to work better than phrase-structure analyses, and discontinuous analyses can be expected to work better than continuous analyses.*

The success of local optimality parsing depends crucially on the existence of neighbourhoods that are large enough to allow the parser to escape local optima, but small enough to be computationally tractable. In local optimality parsing, local optima are usually caused by local garden-paths, in which an analysis that appears to be optimal at one point in time becomes suboptimal at a later time, after new speech segments have been added to the analysis. This notion is defined formally below.

**Definition 1.25** Let $\mathcal{A}$ be an analysis space with ill-formedness measure $c\colon \mathcal{A} \to \mathbb{R}$. A segmented speech signal $s = (s_1, \ldots, s_n)$ is said to contain a *local garden-path* at position $k$ with *disambiguating region* $(s_{k+1}, \ldots, s_{k+d})$ where $d \geq 1$, if (a) no optimal analysis of $(s_1, \ldots, s_k)$ is a subgraph of any optimal analysis of $(s_1, \ldots, s_{k+d})$, and (b) the number $d$ is minimal in this respect.

**Example 1.26** An example of a local garden-path in English is given in (1a) and (1b) below, where incrementality forces the human parser to decide on an analysis of the speech input "I saw the man with" before it sees the disambiguating input, ie, the parser must decide whether "with" specifies the manner of seeing (VP-attachment), or whether it specifies a property of the man (NP-attachment).

(1a) I saw the man with ... my own eyes. (VP-attachment)
(1b) I saw the man with ... the funny hat. (NP-attachment)

The human parser can either decide to analyze "I saw the man with" by means of NP-attachment or VP-attachment. If it chooses an NP-attachment analysis first, then (1a) will turn out to be a local garden-path when the parser, after reading the disambiguating region "my own eyes," discovers that it should have chosen a VP-attachment analysis instead of an NP-attachment analysis. Similarly, if it chooses a VP-attachment analysis first, then (1b) will turn out to be a local garden-path.

In order to escape from a local garden-path, the neighbourhood of the erroneous garden-path analysis must contain the correct analysis, or at least a lower-cost analysis that can lead the parser in the right direction towards the correct analysis. In order to estimate how large the neighbourhoods have to be for the parser to escape, it is convenient to have some measure of the distance between two alternative analyses in an ambiguity. To this end, we will define the concept of edit distance, based on the notion of $k$-change neighbourhoods presented in Definition 1.21 and 1.22.

**Definition 1.27** The *edit distance* between two analyses $a$ and $a'$ is the smallest $k$ that allows $a'$ to be obtained from $a$ by a $k$-change operation.

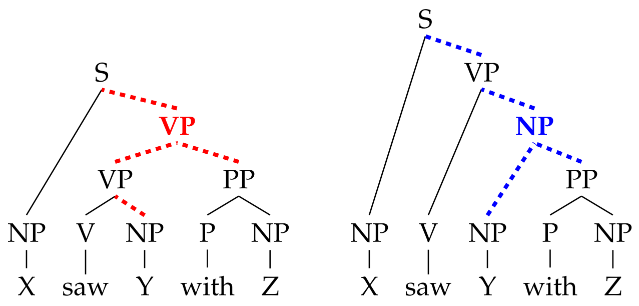


Figure 1.8: Edit distance 5 between two alternative phrase-structure analyses of a PP-attachment ambiguity.

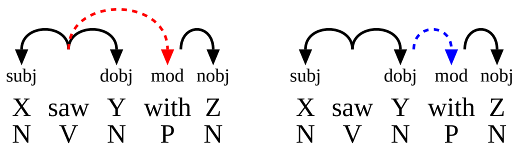


Figure 1.9: Edit distance 1 between two alternative dependency analyses of the PP-attachment ambiguity in Figure 1.8.

Because the time needed to search a $k$-change neighbourhood tends to grow exponentially with $k$, computationally efficient local optimality parsers must restrict themselves to relatively small $k$, for example, $k = 3$. The success of local optimality parsing therefore depends crucially on the edit distance between alternative analyses in ambiguous constructions: if the edit distance is small, alternative analyses will fall within the largest $k$-change neighbourhood searched by the parser and can therefore be selected by improve($a$); but if the edit distance is large, the alternative analyses will become invisible to improve($a$).

**Phrase-structure analyses or dependency analyses?**

In general, the edit distance between alternative analyses depends on the choice of analysis space. For example, the edit distance between alternative phrase-structure analyses tends to be larger than the edit distance between alternative dependency analyses, as illustrated below.

**Example 1.28** In the PP-attachment ambiguity in Figure 1.8, the two phrase-structure analyses are related by a 5-change operation that replaces one node and four edges with another node and another four edges (shown in bold and with dotted lines). The two corresponding dependency analyses in Figure 1.9 are related by a 1-change operation that replaces a single edge with another edge (shown with dotted lines). The dependency representation therefore results in a lower edit distance for the PP-attachment ambiguity than the phrase-structure representation.

In order to understand why dependency analyses and phrase-structure analyses differ with respect to edit distance, we first need to understand the relation between them.

**Remark 1.29** All dependency theories assume that grammatical relations can be expressed as relations between pairs of words. This means that dependency analyses can be encoded as graphs consisting of nodes that represent words, and labelled directed edges that go from one word (called the *head*) to another word (called the *subordinate*). Dependency graphs can be drawn either as *arc graphs*,[11] where the words are placed on a long line, and the directed edges are drawn on top of the words (Figure 1.10 left), or as *classical dependency graphs*, where heads are placed above their subordinates, and the directed edges are drawn from the head down to the subordinate (Figure 1.10 middle). In dependency graphs, information about word class etc. can be drawn either next to or right below each word.

Phrases are a derived notion in dependency theory: Each word heads an associated phrase, which consists of the word itself and all words in the dependency graph that can be reached from the word by following the directed edges. This allows us to translate a dependency graph into an equivalent phrase-structure graph in which each phrase has a lexical head. To do so, we must replace each node in the dependency graph with two nodes in the phrase-structure graph: a terminal node that encodes the word, and a non-terminal node that encodes the word class. Figure 1.10 (right) shows the phrase-structure graph corresponding to the dependency graphs in Figure 1.10. The phrase-structure graph would look even more

[11]The arc graph layout is used in a number of dependency theories, eg, Word Grammar (Hudson 1990). In this dissertation, we will draw dependency graphs with the arc graph layout, because it has many advantages when it comes to drawing general graphs with cyclicity, multiple heads, and discontinuous edges, and large graphs that must be split across several lines and pages.

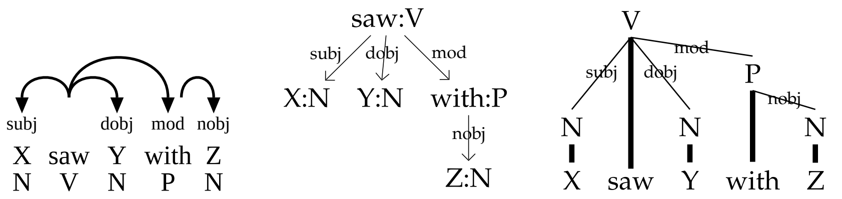


Figure 1.10: Three isomorphic representations of a dependency graph: arc graph (left), classical dependency graph (middle), and phrase-structure graph in which each phrase is lexically headed (right).

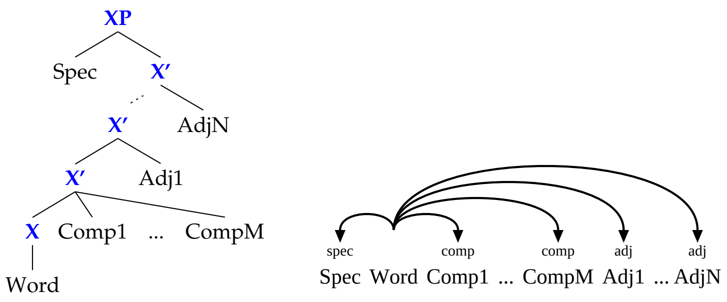


Figure 1.11: Complement-adjunct-specifier encoding in Xbar theory (left) and dependency theory (right). Additional nodes are shown in bold. Both graphs are unordered.

familiar if we also replaced lexical categories (eg, "N", "V", "P") with the corresponding phrasal categories (eg, "NP", "VP", "PP").

**Remark 1.30** In light of Remark 1.29, general phrase-structure graphs can be viewed as dependency graphs with additional non-lexically headed nodes. The three main sources of these additional nodes are that: (a) phrase-structure theories tend to encode complements and adjuncts structurally by means of *Xbar theory* (Figure 1.11 left), whereas dependency theories use a flat structure where complements and adjuncts are encoded by means of edge labels (Figure 1.11 right); (b) phrase-structure theories often posit phrasal nodes that lack a lexical head, such as the sentence node "S"; and (c) some phrase-structure theories (in particular the Chomskyan tradition) posit a large number of phrases with phonetically empty functional categories for tense, agreement, etc., which are encoded lexically in dependency theories.

The additional nodes in phrase-structure analyses mean that the edit distance between alternative analyses in ambiguous constructions tends to be larger in phrase-structure theories than in dependency theories — how much larger depends on how many additional nodes the phrase-structure theory posits. The larger edit distance does not entirely rule out the possibility of creating computationally efficient local optimality parsers based on phrase-structure analyses, provided the neighbourhoods are designed in a clever way, but it does complicate the design of the parser and its parsing operations because the parser needs to modify a larger number of nodes and edges in each parsing step. Local optimality parsing can therefore be expected to work better with dependency analyses than with phrase-structure analyses, and dependency analyses have therefore been chosen as the basis of local optimality parsing in this dissertation.[12]

**Continuous or discontinuous analyses?**

Natural languages contain many examples of discontinuous phenomena, and discontinuity is therefore a central concern in most syntactic theories. The encoding of discontinuous phenomena is as important for the feasibility of local optimality parsing as the choice between dependency and phrase structure.

**Definition 1.31** The relationship between a complement or adjunct $D$ and its head $H$ is called a *continuous dependency* if all words between $D$ and $H$ are contained in the phrase headed by $H$, and a *discontinuous dependency* otherwise.

**Example 1.32** In (2) below, almost all linguistic theories agree that "which museum" must be analyzed as a discontinuous complement of "from."

(2) Which museum did he steal the paintings from?

[12]Ronald Kaplan (p.c.) has pointed out that other representations based on neither phrase structure nor dependency structure (eg, representations based on packed charts, underspecified constructions as in LFG, etc.) might turn out to be as good as dependency analyses in terms of edit distance, if neighbourhoods are cleverly designed. However, in the absence of a better alternative, we are content with using dependency analyses and leaving it to others to explore alternative representations.

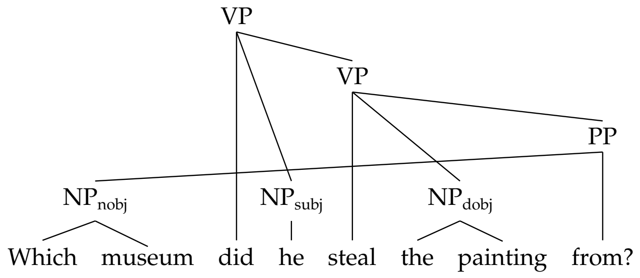


Figure 1.12: An example of a direct encoding of discontinuity with a discontinuous phrase-structure graph.

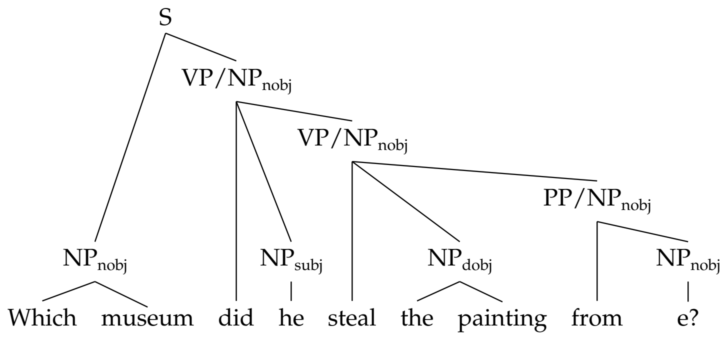


Figure 1.13: An example of an indirect encoding of discontinuity with a continuous phrase-structure graph.

However, linguistic theories differ with respect to how discontinuous dependencies should be encoded in linguistic analyses. There are basically two ways of doing it: the discontinuous dependency can either be represented *directly* in a discontinuous analysis graph by an edge between the dependent and the governor (Figure 1.12), or *indirectly* in a continuous analysis graph by attaching the dependent to a special landing site and marking each node on the path connecting the governor and the landing site in a special way (Figure 1.13); the encoding used in Categorial Grammar can be viewed as a variant of the indirect encoding.

Historically, the indirect encoding has been dominant within phrase-structure grammars, eg, the Chomskyan school of syntax, GPSG (Gazdar et al. 1985), and early HPSG (Pollard and Sag 1994). In contrast, the direct encoding has been dominant within dependency theories, eg, Tesnière

(1953), Melcuk (1988), Hudson (1990, 2003), and Hellwig (2003). However, this is only a tendency, and the direct encoding has been used, at least to some degree, in phrase-structure grammars such as LFG (Dalrymple et al. 1994) and in discontinuous versions of HPSG (see Müller (2004) for an overview).

The way we have defined edit distance, the indirect encoding leads to a larger edit distance for discontinuous dependencies than the direct encoding: in order to change a discontinuous dependency, we need to modify all the arbitrarily many nodes on the path between the governor and the landing site, instead of only changing the graph locally at a few nodes or edges around the dependent. This means that local optimality parsing can be expected to work better with direct encodings than with indirect encodings, at least if we want to restrict ourselves to neighbourhoods that can be contained within *k*-change neighbourhoods for small *k*.[13]

## 1.5 Summary

In this chapter, we have seen that human grammars can be viewed as utility measures that assign a cost to all possible analyses, and that parsing and generation can be viewed as optimization problems. We have argued that these optimization problems are NP-hard and require heuristic algorithms for their solution, and we have proposed to use local optimality parsing as our heuristic algorithm. Finally, we have argued that local optimality parsing works best if the edit distance between alternative analyses in ambiguous constructions is low, and that analysis spaces based on discontinuous dependency graphs work best in this respect.

Our ideas about analysis spaces, ill-formedness measures, parsing, and learning will be described in detail in the remainder of the dissertation. In chapters 2 and 3, we will introduce the formal machinery needed to specify analysis spaces and their associated ill-formedness measures within our discontinuous dependency framework. In chapter 4, we will show how

[13]We could conceivably construct isomorphic mappings between direct and indirect encodings that would allow us to design computationally efficient neighbourhoods for indirect encodings, even if these neighbourhoods were not contained in a *k*-change neighbourhood for any small fixed *k*. However, while this approach may be feasible, we think it is more straightforward to choose a direct encoding because it does not require a very ingenious design of neighbourhoods in order to work with local optimality parsing.

this formal framework can be used to provide analyses for a wide range of linguistic phenomena. In chapters 5 and 6, we will introduce the formal machinery needed to specify probabilistic dependency grammars whose probabilities are estimated from treebank data. Finally, in chapter 7, we will describe local optimality parsing in more detail and examine different sets of parsing operations.

# Part II

# Syntax formalism

# Chapter 2

# The analysis space: how lexemes license edges in the graph

In this chapter, we describe how the analysis space is defined in our syntax formalism, Discontinuous Grammar (DG). We describe how lexemes are organized in a multi-hierarchical default inheritance lexicon that allows lexical rules to be encoded without redundancy, and how lexical transformations are used to account for inflections, passives, and similar constructions. We then define analyses as dependency graphs whose nodes and edges are licensed by lexical rules that describe the possible complements, adjunct governors, landed nodes, fillers, and gapping fillers for a lexeme.

As we saw in the previous chapter, local optimality parsing presupposes an analysis space and an ill-formedness measure that can be used to compare different analyses with each other. In continuation of the discussion in section 1.4, we will assume that human language is dependency-based, and that a speaker's analysis space consists of all conceivable dependency graphs licensed by the speaker's grammar. The grammar rules in our proposed model will be described in detail in this chapter, and the cost operators that our speakers use to specify ill-formedness measures will be described in chapter 3.

## 2.1 Dependency graphs

***Summary**. We define dependency graphs, yields, continuous and discontinuous edges,* *cyclicity, and multiple heads.*

Before proceeding with our inventory of grammar rules, it is useful to define dependency graphs formally. Our dependency graphs will be used to encode several layers of information: deep syntactic dependencies, surface syntactic word order, the relationship between a filler and its filler source, and the relationship between an anaphor and its antecedents. This means that our dependency graphs cannot be encoded as trees, but must be encoded as general directed graphs that may include discontinuity, multiple heads, and acyclicity.

**Definition 2.1** A *general dependency graph* is a tuple $G = (N, E, \lambda, <)$ where $N$ is a finite set of nodes that is labeled by the function $\lambda \colon N \rightarrow \Lambda_n$ where $\Lambda_n$ is a set of possible node labels, and $E$ is a set of directed edges in $N \times N \times \Lambda_e$ where $\Lambda_e$ is a set of possible edge labels; the non-filler nodes in $N$ must be linearly ordered by $<$. An element $(n_1, n_2, r)$ in $E$ is called a *directed edge* from *head node* $n_1$ to *subordinate node* $n_2$ with label $r$, and is often written $n_1 \xrightarrow{r} n_2$.

**Remark 2.2** Our definition of dependency graphs does not place any restrictions on the set of node labels $\Lambda_n$ and edge labels $\Lambda_e$. This means that labels can be complex feature structures as well as atomic values. However, feature structures are not as expressive in our setup as in HPSG because our definition does not allow structure-sharing between feature structures associated with different nodes and edges — this also makes feature structures more computationally efficient in our setup than in HPSG.

In Discontinuous Grammar (DG), node labels are assumed to encode the set of candidate lexemes at a node and the currently selected lexeme from this set of candidate lexemes, along with a feature function that can be used by the parser to encode information about the node.

**Definition 2.3** A *DG graph* (or simply *dependency graph*) is a general dependency graph in which every node label is a triple of the form $(f, T, t)$ where $f$ is a local feature function that contains the feature specifications associated with the node, $T$ is the set of all potential lexemes associated with the node, and $t$ is a member of $T$ that has been singled out as the node's *active lexeme*; the remaining lexemes in $T$ are called the *non-active lexemes* associated with the node.

The notions of yield, domination, discontinuity, cyclicity, and multiple heads are defined below.

**Definition 2.4** In a dependency graph $G$, the *yield* of a node $n$ with subordinates $s_1, \ldots, s_k$ is defined recursively as the set

$$Y_G(n) = \{n\} \cup Y_G(s_1) \cup \ldots \cup Y_G(s_k)$$

consisting of $n$ and the yields of its subordinates.

**Definition 2.5** A node $n$ in a dependency graph $G$ is said to *dominate* a node $n'$ if $n'$ is contained in the yield of $n$. The dominance is called *strict* if $n \neq n'$.

**Definition 2.6** An edge $h \xrightarrow{r} s$ in a dependency graph $G$ is called *discontinuous* (*non-projective*) if there exists a non-filler node $b$ that lies between $h$ and $s$ in the linear word order such that $b$ is not contained in the yield $Y_G(h)$, and *continuous* (*projective*) otherwise. A dependency graph $G$ is called *discontinuous* (*non-projective*) if it contains a discontinuous edge, and *continuous* (*projective*) otherwise.

**Definition 2.7** A dependency graph $G$ is called *cyclic* if it contains a path consisting of edges $n_1 \longrightarrow n_2$, $n_2 \longrightarrow n_3$, $\ldots$, $n_{k-1} \longrightarrow n_k$ where $n_1 = n_k$; otherwise, $G$ is called *acyclic*.

**Definition 2.8** A node $n$ in a dependency graph $G$ is said to have *multiple heads* if $G$ contains more than one edge where $n$ is the subordinate node.

Having defined our dependency graphs and our notions of yield, discontinuity, cyclicity, and multiple heads, we can now proceed with defining our dependency lexicons.

## 2.2 The lexicon: multiple inheritance with defaults

***Summary.*** *We describe a default multiple inheritance lexicon with type transformations and dynamic feature functions, and a type specification language that can be used in lexical entries to identify nodes in a dependency graph based on their structural and lexical properties.*

In Discontinuous Grammar, we assume that the lexicon stores all information associated with the linguistic units in the language. Following standard terminology, these units are called *types*, and may represent both words, morphemes, and idioms. We also assume that types are organized in a prioritized multiple inheritance lexicon with defaults (cf. Briscoe et al. 1993, 40), so that types can inherit properties from supertypes, thereby allowing the lexicon to be encoded compactly.

In the following, we let $\mathcal{F}$ denote a set of possible *features* with four distinguished features: super, tfuncs, transforms, and dynamic. We also let $\mathcal{V}$ denote a set of possible *feature values* with two distinguished values: undef (undefined value) and null (null value).

**Definition 2.9** A *local feature function* is a function $f\colon \mathcal{F} \to \mathcal{V}$ that maps a *feature* $x \in \mathcal{F}$ into a *local feature value* $f(x)$.

**Definition 2.10** A finite set $L = \{f_t\}_{t \in T}$ of local feature functions is called a *prioritized multiple inheritance lexicon with defaults* (or simply an *inheritance lexicon*) with *types* $T$ provided each $f_t(\mathsf{super})$ is an ordered list of types whose elements are called the *ordered supertypes* of $t$.

**Definition 2.11** For each type $t$ in an inheritance lexicon $L = \{f_t\}_{t \in T}$, the *inherited feature function* $F_t\colon \mathcal{F} \to \mathcal{V}$ is defined recursively by:

$$F_t(x) = f_t(x) \,||\, F_{s_1}(x) \,||\, \ldots \,||\, F_{s_n}(x) \,||\, \mathsf{undef}$$

where $s_1, \ldots, s_n$ are the ordered supertypes of $t$, and $a \,||\, b$ is defined by:

$$a \,||\, b = \begin{cases} a & \text{if } a \neq \mathsf{undef} \text{ and } a \neq \mathsf{null} \\ b & \text{if } a = \mathsf{undef} \\ \mathsf{undef} & \text{if } a = \mathsf{null} \end{cases}$$

In our setup, the inherited feature function is well-defined regardless of whether the super feature defines an acyclic graph structure on $T$ or not.[1] However, in most situations, it is reasonable to follow standard practice and require the lexicon to be hierarchical in the sense defined below.

[1]An infinite loop during the evaluation of $F_t(x)$ can be avoided by cutting the cycles, ie, we let $F_t(x)$ return undef if it is called within an evaluation of $F_t(x)$. This only affects the computation of $F_t(x)$, without affecting the value.

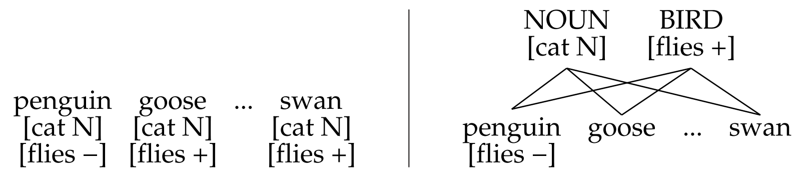


Figure 2.1: Two inheritance lexicons with the same inherited feature functions: the first lexicon (left) encodes everything locally, whereas the second (right) uses inheritance to avoid duplicating local feature assignments.

**Definition 2.12** A lexicon $L = \{f_t\}_{t \in T}$ is called *hierarchical* if the super feature defines an acyclic graph structure on $T$, ie, no type in $T$ is the transitive supertype of itself.

**Example 2.13** Two examples of hierarchical inheritance lexicons are shown in Figure 2.1. The lexicon on the left specifies all feature values locally, ie, $F_t(x) = f_t(x)$ for all $t$ and $x$, but is inefficient because feature values that are shared between many types must be specified locally at every type rather than shared by means of inheritance. The lexicon on the right uses inheritance to obtain a more compact encoding that introduces the two supertypes NOUN and BIRD; here we have $F_{\mathsf{goose}}(\mathsf{flies}) = f_{\mathsf{BIRD}}(\mathsf{flies}) = +$, but $f_{\mathsf{goose}}(\mathsf{flies}) = \mathsf{undef}$, ie, the value of the feature flies in goose is inherited from BIRD.

In some cases, it is useful to be able to combine two types $a$ and $b$ into a composite type that is a subtype of both $a$ and $b$.

**Definition 2.14** Let $a, b$ be types in an inheritance lexicon $L$. Then $a * b$ is a *composite type* that denotes a type that has an empty local feature function and $a, b$ as its supertypes.

In the program DTAG, our computational implementation of the theory presented in this thesis, the lexicon is encoded in a lexicon file where types, supertypes, and feature values are specified with the following code.

**Pseudocode 2.15** The function type(*t*) returns the object associated with the type *t* in the lexicon; the object is created if it did not exist in the lexicon before. The method type(*t*)−>set(*f*, *v*) sets the local feature *f* to the value *v* in the type *t*, and returns *t*. The method type(*t*)−>get(*f*) returns the inherited value associated with the feature *f*, and type(*t*)−>get(*regexp*) returns a hash table containing all feature-value pairs whose feature matches the regular expression *regexp*. The method type($t$)−>super($s_1$, ..., $s_n$) specifies that $s_1, \ldots, s_n$ are the ordered supertypes associated with $t$, and returns $t$.[2]

**Example 2.16** The example below shows how the types NOUN, BIRD, and penguin in the inheritance lexicon in Figure 2.1 (right) are encoded in DTAG:[3]

```
type(NOUN)−>set(cat, N);
type(BIRD)−>set(flies, '+');
type(penguin)−>super(NOUN, BIRD)−>set(flies, '−');
```

To encode inflectional morphology and lexical alternations as compactly as possible in the lexicon, it is convenient to have transformations that allow us to create a new type by transforming a source type with a transformation type. The transforms feature of the source type specifies all the transformation types that are licensed by the source type, and the tfuncs feature of the transformation type encodes how the transformed type is computed from the transformation type and the source type.

**Definition 2.17** Let $L = \{f_t\}_{t \in T}$ be an inheritance lexicon. A type $t \in T$ is called a *transformation type* if $F_t(\text{tfuncs})$ is an ordered list $(\phi_1, \ldots, \phi_k)$ of functions $\phi_i$ that given a *source type* $s \in T$ with $t \in F_s(\text{transforms})$ computes a local feature function $\phi_i(s,t)\colon \mathcal{F} \to \mathcal{V}$. We call the $\phi_1, \ldots, \phi_k$ the *transformation functions* associated with $t$, and we call

$$f_{s.t} = \phi_1(s,t) \,||\, \ldots \,||\, \phi_k(s,t) \,||\, F_s$$

the *local feature function* associated with the *transformed type* $s.t$.

**Definition 2.18** An inheritance lexicon $L = \{f_t\}_{t \in T}$ is called *transformational* provided (a) each $F_s(\text{transforms})$ is an unordered list of transformation types, and (b) $s.t \in T$ and $f_{s.t}$ is the local feature function associated with the transformed type $s.t$, for each $s, t \in T$ with $t \in F_s(\text{transforms})$.

[2]The method type($t$)−>super($s_1$, ..., $s_n$) is implemented as an abbreviation for the method type($t$)−>set(super, [$s_1$, ..., $s_n$]).

[3]DTAG is implemented in Perl, and the DTAG encoding in Example 2.16 is actually Perl code that is compiled by Perl when the lexicon is loaded. Perl will guess that character sequences like NOUN and BIRD should be interpreted as strings (rather than procedure calls), which can be specified unambiguously by enclosing them in single or double quotes. In the text, we have avoided quotes whereever possible to preserve readability. But in real DTAG lexicons, it is safer to specify the quotes explicitly.

In the DTAG lexicon, it is convenient to be able to specify licensed transformations and transformation functions independently of each other, rather than as a list. For that reason, we will assume the DTAG lexicon allows a feature value to be computed as the list of all feature values with a feature that matches a given regular expression.

**Pseudocode 2.19** In a type $t$, the feature value listof(*regexp*) returns a list containing all (non-dynamic) feature values $v_i$ of $t$ with feature $f_i$ where $f_i$ matches the regular expression *regexp*, with the $v_i$ sorted alphabetically with respect to the feature name $f_i$.

**Remark 2.20** Using the listof feature value, we can specify in the DTAG lexicon that the transforms and tfuncs feature values should be computed as the lists of all transformations and transformation functions associated with the type, using the following code:

```
type(word)
    −> set(transforms, listof('/transform:/'))
    −> set(tfuncs, listof('/tfunc:/'));
```

Transformations and transformation functions can then be specified in the DTAG lexicon as follows.

**Pseudocode 2.21** The method type(*s*)−>transform(*name*, *t*) specifies that the source type *s* licenses the transformation type *t*, using *name* as identifier. Similarly, the method type(*t*)−>tfunc(*name*, *prefix*, *sub*) specifies that the transformation type *t* is associated with the transformation function *sub*, using *name* as identifier and sortkey, and that *sub* returns a feature function whose feature names begin with *prefix*. The transformation function *sub* is encoded as a Perl subroutine that returns a Perl hash containing all generated feature-value pairs when called as &*sub*(*s*, *t*) where *s* is the source type.[4]

The example below illustrates how type transformations can be used to encode plurals in the DTAG lexicon, without having to create more than

[4]The method type(*t*)−>transform(*name*, *t*) is implemented as an abbreviation for type(*t*)−>set("transform:*name*", *t*). Similarly, the method type(*t*)−>tfunc(*name*, *prefix*, *sub*) is implemented as an abbreviation for type(*t*)−>set("tfunc:*name*", [*prefix*, *sub*]).

one type for each base form. We will present a more sophisticated encoding of morphology, including lexical alternations, in Section 4.7.

**Example 2.22** The lexical entry below specifies that all nouns are singular and license a pl transformation with the transformation type pl:s by default; that the types tiger, elephant, and glass are nouns; and that glass licenses a pl transformation with the transformation type pl:es instead of pl:s.

```
type(noun)−>set(num, sg)−>transform(pl, ”pl:s”);
type(tiger)−>super(noun);
type(elephant)−>super(noun);
type(glass)−>super(noun)−>transform(pl, ”pl:es”);
. . .
```

The transformation types pl:s and pl:es are defined as subtypes of pl, with suffixes "s" and "es", respectively. Their supertype pl specifies two transformation functions: the num transformation function assigns "pl" to the feature "num", and the "phon" transformation function creates a new phon feature value by concatenating the source type's phon feature value with the transformation type's suffix feature value.

```
type(”pl:s”)−>super(pl)−>set(suffix, s);
type(”pl:es”)−>super(pl)−>set(suffix, es);
type(pl)
    −>tfunc(num, num, sub {return {num=>pl}})
    −>tfunc(phon, phon,
        sub {my ($s,$t) = @_;
                return {phon=>$s−>get(phon) . $t−>get(suffix)};});
```

To encode filler constructions such as relatives, it is convenient to have dynamic feature functions that depend on the dependency graph surrounding the node with which the type is associated. This allows a dynamic type to mirror the feature-value pairs of another node in the graph, or to compute complement frames or semantic interpretations dynamically on the basis of types associated with lexemes in the nearby graph structure. Dynamic feature functions will be illustrated in Example 2.86.

**Definition 2.23** A *dynamic feature function* is a function $d$ that given a dependency graph $G$ and a node $n$ computes a local feature function $d(G,n): \mathcal{F} \to \mathcal{V}$.

**Definition 2.24** An inheritance lexicon $L = \{f_t\}_{t \in T}$ is called *dynamic* if $F_t(\text{dynamic})$ is an ordered list of dynamic feature functions for each $t \in T$.

**Definition 2.25** Given a dynamic inheritance lexicon $L$ and a type $t$ with inherited feature function $F_t$ and dynamic feature functions $F_t(\text{dynamic}) = (d_1, \ldots, d_k)$, we define the *dynamic inherited feature function* $F_t^*$ associated with $t$ by:

$$F_t^*(G, n) = d_1(G, n) \,||\, \ldots \,||\, d_k(G, n) \,||\, F_t$$

**Remark 2.26** Using the listof feature value, we can specify in the DTAG lexicon that the dynamic feature value should be computed as the alphabetically ordered list of all dynamic transformations encoded in a feature prefixed by "dynamic:", using the following code:

```
type(word) −> set(dynamic, listof('/dynamic:/'));
```

**Pseudocode 2.27** The method type(*t*)−>setd(*name*, *prefix*, *sub*) specifies that the type *t* has a dynamic feature function *sub* whose generated features start with *prefix*, using *name* as identifier and sortkey.[5] The dynamic feature function *sub* is encoded as a Perl subroutine that returns a Perl hash containing all dynamically generated feature-value pairs when called as &*sub*(*G*, *n*). The method *G*−>getd(*n*, *regexp*) returns a hash containing all dynamic inherited feature-value pairs induced by $F_t^*(G, n)$ at the node *n* in the graph *G* for which the feature prefix and feature name matches the regular expression *regexp*, where *t* is the active lexeme associated with *n*. The method *G*−>get(*n*, *f*) returns the dynamic value $F_t^*(G, n)$ associated with the feature $f$ of the active lexeme $t$ for node *n* in graph *G*.

In lexical descriptions, we often need to restrict the nodes and edges to which a rule applies. We will do this by means of the type specification language presented below, which allows us to test whether a particular node or edge has particular structural or lexical properties.

**Definition 2.28** An edge is called *pointed* if one of its two nodes has been marked as pointed node. The notation $h \xrightarrow{r} s$ is used for pointed edges with pointed node $h$, and $s \xleftarrow{r} h$ is used for pointed edges with pointed

[5]The method type(*t*)−>setd(*name*, *prefix*, *sub*) is implemented as an abbreviation for type(*t*) −>set("dynamic:*name*", [*prefix*, *sub*]).

node $s$. In a node context, the pointed edge refers to the associated node; in an edge context, the pointed edge refers to the associated edge. Pointed edges are always interpreted in node context, unless specified otherwise.

**Definition 2.29** A *type specification* is an expression associated with a particular node in a dependency graph $G$ that is built up from the simple expressions in Figure 2.2 and 2.3. The parentheses in type specifications are often omitted if the resulting expression is unambiguous.

**Example 2.30** In an appropriately defined lexicon, the type specification $\mathsf{rain{:}v1} + \mathsf{finite} + [\xrightarrow{\mathsf{subj}} \mathsf{it{:}n1}]$ could be used to match any node whose lexeme is the finite verb "rain", and which has an "it" subject. Similarly, $\mathsf{rain{:}v1} + \mathsf{finite} - [\xrightarrow{\mathsf{subj}} \mathsf{it{:}n1}]$ matches a node for the finite verb "rain" that lacks an "it" subject. Since type specifications can be used recursively on heads and subordinates, we can specify arbitrarily complex graph structures. For example, $\mathsf{cup{:}n1} + [\xrightarrow{\mathsf{pobj}} \mathsf{of{:}sp1} + [\xrightarrow{\mathsf{nobj}} \mathsf{tea{:}n1}]]$ can be used to specify the idiom "cup of tea".

As in other linguistic theories, we will assume that the edges in dependency graphs are licensed by rules, known as *frames*, that are encoded in the lexicon: complement and adjunct frames are used to control the deep structure that encodes dependencies, landing frames are used to control the surface structure that controls word order, and filler and gapping frames are used to control secondary dependencies. In the following sections, we will describe these frames formally and define what it means for a node to be well-formed with respect to a particular frame. By means of these frame-specific notions of well-formedness, we can specify what it means for a graph to be globally well-formed:

**Definition 2.31** A dependency graph $G$ is called *well-formed* with respect to a lexicon $L$ if:

(a) $G$ is deeply well-formed with respect to $L$ (cf. Definition 2.45);
(b) $G$ is surface well-formed with respect to $L$ (cf. Definition 2.55);
(c) all fillers in $G$ are well-formed with respect to $L$ (cf. Definitions 2.81 and 2.95);
(d) all edges in the graph are licensed by one of (a)–(c).

| Type specification | DTAG encoding | Matches |
|---|---|---|
| this | this | the node where the type specification is situated |
| $t$ | isa($t$) | any object whose dynamic type is a subtype of $t$ or equals $t$ |
| $[\xrightarrow{a} b]$ | out($a$, $b$) | any pointed edge $h \xrightarrow{r} s$ where $r$ matches $a$, and $s$ matches $b$ |
| $[\xleftarrow{a} b]$ | in($a$, $b$) | any pointed edge $s \xleftarrow{r} h$ where $r$ matches $a$, and $h$ matches $b$ |
| $[< a]$ | before($a$) | any node that precedes a node that matches type specification $a$ |
| $[> a]$ | after($a$) | any node that succeeds a node that matches type specification $a$ |
| prev$(a, b)$ | prev$(a, b)$ | any node within the set of all nodes that match $a$ that immediately precedes a node that matches $b$ (cf. Definition 2.71) |
| next$(a, b)$ | next$(a, b)$ | any node within the set of all nodes that match $a$ that immediately succeeds a node that matches $b$ (cf. Definition 2.71) |
| extract$(a)$ | extract$(a)$ | any node whose extraction path contains a node or edge matching $a$ (cf. Definition 2.64) |
| leftp$(a, i, o)$ | leftp$(a, i, o)$ | any node in the left $(i, o)$-periphery of any node matching $a$ (cf. Definition 2.73) |
| rightp$(a, i, o)$ | rightp$(a, i, o)$ | any node in the right $(i, o)$-periphery of any node matching $a$ (cf. Definition 2.73) |
| edge$(a)$ | edge$(a)$ | matches any edge that matches $a$ |
| list$(a)$ | list$(a)$ | matches the unordered list of all objects matching $a$ |
| member$(a)$ | member$(a)$ | matches any member in a list matching $a$ |

Figure 2.2: Our type specification language: $t$ denotes an arbitrary type in the lexicon, and $a$ and $b$ are arbitrary type specifications.

**Remark 2.32** We will assume that the names of the complement, adjunct, landing, and filler frames associated with different nodes in a dependency graph $G$ are encoded in the features associated with the nodes in $G$, as listed in Figure 2.4.

| Type specification | DTAG encoding | Matches |
|---|---|---|
| $\mathsf{val}(f, a)$ | $\mathsf{val}(f, a)$ | matches the value of the $f$ feature of any node matching $a$ |
| $\mathsf{etype}(a)$ | $\mathsf{etype}(a)$ | matches the edge type of any edge matching $a$ |
| $\mathsf{dist}(a, b)$ | $\mathsf{dist}(a, b)$ | matches the distance in time/position from the unique node matching $a$ to the unique node matching $b$ |
| $\mathsf{order}(a, b)$ | $\mathsf{order}(a, b)$ | matches the position of the unique node matching $b$ within the set of all nodes matching $a$ |
| $\mathsf{var}(v, a)$ | $\mathsf{var}(v, a)$ | matches any object, but with the side effect of declaring a deterministic variable $v$ that refers to the unique node matching type specification $a$ |
| $\mathsf{nvar}(v, a)$ | $\mathsf{nvar}(v, a)$ | matches any object, but with the side effect of declaring a non-deterministic variable $v$ that refers to any node matching type specification $a$ |
| $\mathsf{var}(v)$ | $\mathsf{var}(v)$ | the node that variable $v$ refers to |
| $-a$ | $-a$ | any object that does not match $a$ |
| $(a + b)$ | $a + b$ | any object that matches both $a$ and $b$ |
| $(a - b)$ | $a - b$ | any object that matches $a$, but not $b$ |
| $(a\|b)$ | $a\|b$ | any object that matches $a$ or $b$ |
| $\mathsf{where}(v, a)$ | $\mathsf{where}(v, a)$ | any object for which condition $a$ is true if the variable $v$ refers to the object |
| $a = b, a \neq b$ | eq($a$,$b$), ne($a$,$b$) | matches all pairs $(o, o')$ of objects where $o$ matches $a$, $o'$ matches $b$, and $o = o'$ (and similarly for $a \neq b$) |
| $a < b, a \leq b,$ $a > b, a \geq b$ | lt($a$,$b$), le($a$,$b$), gt($a$,$b$), ge($a$,$b$) | matches all pairs $(s, s')$ of strings where $s$ matches $a$, $s'$ matches $b$, and $s < s'$ (and similarly for $a \leq b$, $a > b$, and $a \geq b$) |

Figure 2.3: Our type specification language (continued): $t$ denotes an arbitrary type in the lexicon, and $a$ and $b$ are arbitrary type specifications.

In this section, we have defined dependency lexicons that can deal with inheritance, transformations that can be used to encode morphology and lexical alternations, and dynamic feature functions that can be used to com-

| name of | encoded in |
|---|---|
| complement frame | cframe feature of governor |
| adjunct frame | aframe feature of adjunct (set to undef if no adjunct frame is selected) |
| landing frame | lframe feature of landed node |
| filler frame | fframe feature of filler |

Figure 2.4: How frames are encoded in features.

pute feature values dynamically from the graph. We have also defined a type specification language that can be used to specify constraints on the local graph structure around a node. In the following section, we will define our notion of complements and adjuncts, and how they are encoded in the lexicon.

## 2.3 Deep trees: complements and adjuncts

***Summary***. *We explain how speech signals are segmented into phrases that consist of a head and its associated complements and adjuncts, and how the principle of compositionality acts as the basis of semantic interpretation by establishing a correspondence between dependency structure and functor-argument structure. We describe how the lexicon licenses complement and adjunct frames, and how dependency edges must form a possibly discontinuous deep tree in order to constitute a well-formed analysis. Finally, we discuss the difference between complements and adjuncts.*

Most linguistic theories assume that humans divide speech signals into speech segments that represent lexemes, and that these lexemes are subsequently grouped into larger units, called *phrases*. In dependency theories, each phrase consists of a *phrasal head* (or *governor*) lexeme that determines the syntactic and semantic type of the phrase, and a set of subphrases whose heads are called the *dependents* of the governor. The phrasal relationship between a governor and a dependent is called a *dependency*.

Most semantic theories assume that phrases are assigned meanings according to the *principle of compositionality*, which states that the meaning of a phrase is computed as a function of the meanings of its parts.[6] It is

[6]Many compositional theories assume that the interpretation function that computes the meaning of a phrase is conditioned on the context, ie, the context is always passed on as a

natural to assume that the interpretation functions used to construct the meaning $[\![G]\!]$ of a phrase $G$ with governor $g$ must be specified in the lexical entries of the governor $g$ and its dependents. In particular, we will make the following assumption.

**Assumption 2.33** Let $M$ denote the set of all possible meanings. The meaning $[\![G]\!]$ of a phrase $G$ with governor $g$ can be computed by dividing the dependents of $g$ into two ordered subsets, the *complements* $c_1, \ldots, c_n$ of $g$ with complement phrases $C_1, \ldots, C_n$, and the *adjuncts* $a_1, \ldots, a_k$ of $g$ with adjunct phrases $A_1, \ldots, A_k$, and setting:

$$[\![G]\!] = \mu_1([\![A_1]\!]) \circ \cdots \circ \mu_k([\![A_k]\!]) \circ \phi([\![C_1]\!], \ldots, [\![C_n]\!])$$

where $\phi\colon M^n \to M$ (called the *functor* associated with $g$) is provided by the lexical entry of the governor $g$, and each $\mu_i\colon M \to (M \to M)$ (called the *modifier* associated with $a_i$) is provided by the lexical entry of the adjunct $a_i$.

**Remark 2.34** The computation of the meaning of a phrase can be visualized by means of a *functor-argument tree*, shown schematically in Figure 2.5. Each node in the tree corresponds to the application of a functor or modifier supplied by a governor or an adjunct.

**Remark 2.35** As in most other linguistic theories, we assume that a complement is lexically determined by its governor, whereas an adjunct may combine freely with any governor that matches its lexical requirements. This means that a governor can only have a limited number of complements, whereas it may have an unlimited number of adjuncts. Most theories assume that complements can be both optional and obligatory, and that adjuncts are always optional, although some theories operate with obligatory adjuncts. In DG, both optional complements and obligatory adjuncts are permitted.

In our dependency graphs, a complement or adjunct dependency is encoded with a labelled edge $g \xrightarrow{r} d$ from the governor to the dependent. Dependency edges are also called *deep edges*.

parameter to the interpretation function. This assumption is necessary in order to explain contextual effects during interpretation, and we will always assume that the contextual parameter is implicitly present. However, in order to simplify the presentation, we will usually avoid stating it explicitly.

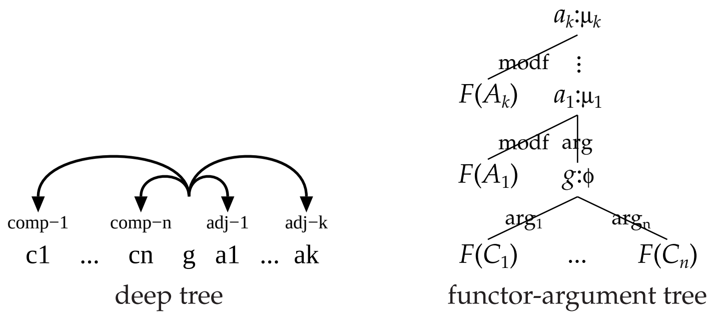


Figure 2.5: The functor-argument tree $F(G)$ (right) associated with an unordered deep tree $G$ (left) consisting of a governor $g$ with functor $\phi$, complements $c_1, \ldots, c_n$ with functor-argument trees $F(C_1), \ldots, F(C_n)$, and adjuncts $a_1, \ldots, a_k$ with modifiers $\mu_1, \ldots, \mu_k$ and functor-argument trees $F(A_1), \ldots, F(A_k)$.

**Definition 2.36** The subgraph $D(G)$ of a dependency graph $G$ consisting of all deep edges in $G$ is called the *deep forest* (or *deep graph*) associated with $G$. A connected component in the deep forest is called a *deep tree* (or a *dependency tree*).

**Definition 2.37** The *deep yield* of a node $n$ in a dependency graph $G$ is the yield of $n$ in the deep tree for $G$.

**Remark 2.38** A deep tree is shown schematically in Figure 2.5 (left). Deep trees can be viewed as underspecified functor-argument trees, since the deep tree does not encode the functional order in which the modifiers are applied (the *modifier scope*). Given a modifier scope, the deep tree determines the functor-argument tree uniquely.

**Remark 2.39** Edge labels are organized in an inheritance hierarchy where all complement edge labels are subtypes of the type complement, all adjunct edge labels are subtypes of the type adjunct, and the types complement and adjunct are immediate subtypes of the type deep. An example of an edge hierarchy is shown in Figure 2.6.

**Remark 2.40** When complements are optional, the functor may lack the argument that should have been provided by a missing complement. In

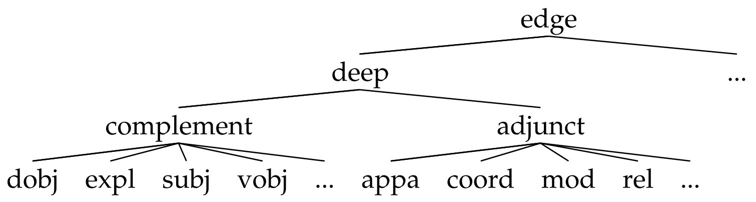


Figure 2.6: A fragment of an edge hierarchy with deep edges.

these cases, we will assume that the functor receives the special value null as the argument of the missing complement. The functor can then replace the null argument with a default value.

We will assume that each lexeme provides a set of associated complement and adjunct frames that specify an interpretation function along with syntactic and semantic restrictions on the phrases that produce the arguments to the interpretation function.

**Definition 2.41** A lexical type $t$ must have one or more associated *complement frames* of the form $(\phi, r_1 : c_1, \ldots, r_n : c_n)$, which licenses $n$ complement edges of the form $t \xrightarrow{r_i} c_i$ with complement edge type $r_i$ where the complement matches the type specification $c_i$. Relation names must be unique, ie, $r_i \neq r_j$ unless $i = j$. The function $\phi\colon M^n \to M$ (called the *functor* associated with the complement frame) computes the meaning of the complement frame from the meanings $[\![C_1]\!], \ldots, [\![C_n]\!]$ (called the *arguments* of $\phi$) associated with the complement phrases $C_1, \ldots, C_n$.

**Definition 2.42** A lexical type $t$ must have zero or more associated *adjunct frames* of the form $(\mu, r : g)$, which licenses an adjunct edge $g \xrightarrow{r} t$ with adjunct edge type $r$ where the adjunct governor matches the type specification $g$. The function $\mu\colon M \to (M \to M)$ (called the *modifier* associated with the adjunct frame) computes the meaning $\mu([\![A]\!])(m)$ of the adjunct frame from the meaning $[\![A]\!]$ associated with the adjunct phrase $A$ and the meaning $m$ associated with the governor after the governor has combined with all its complements and lower-scoped adjuncts.

**Pseudocode 2.43** Complement frames are specified with the method:

```
type($t$)->cframe($C$, $f$, $r_1$ => $c_1$, ..., $r_n$ => $c_n$);
```

| Type | Super types | Complement frame | Adjunct frame |
|---|---|---|---|
| a | PI | $\xrightarrow{\text{nobj}}$ : NC | |
| boy | NC | | |
| John | NP | | |
| little | AN | | $\xleftarrow{\text{mod}}$ : N |
| saw | VA | $\xrightarrow{\text{subj}}$ : N, $\xrightarrow{\text{dobj}}$ : N | |
| telescope | NC | | |
| today | RG | | $\xleftarrow{\text{mod}}$ : V \| N |
| with | SP | $\xrightarrow{\text{nobj}}$ : N | |

Figure 2.7: A small lexicon with supertypes, complements, and adjuncts.

where $C$ is a unique identifier for the complement frame, $f$ is the functor, and $c_i$ is the type specification for the complement with edge type $r_i$. Similarly, adjunct frames are specified with the method:

type(*t*)−>aframe(*A*, *m*, *r* => *g*);

where $A$ is a unique identifier for the adjunct frame, $m$ is the modifier, $r$ is the edge type, and $g$ is the type specification for the adjunct governor.[7] In both complement frames and adjunct frames, functors are assumed to be specified as Perl code references that compute a semantic representation &f($r_1$ => $a_1$,. . .,$r_n$ => $a_n$) for the combined functor, given the semantic representations $a_1$,. . .,$a_n$ corresponding to all the complement or adjunct governor roles $r_1$,. . . ,$r_n$ in the complement or adjunct frame.

**Example 2.44** Figure 2.7 shows a small lexicon with supertypes, complement frames, and adjunct frames for the words in the sentence "John saw a little boy with a telescope today". The immediate supertypes are the two first letters of the PAROLE-based word class hierarchy listed in Figure 4.62 on p. 205. The edge labels are listed in Figure 4.63.

The lexicon specifies that the word "a" is an indefinite pronoun (PI), which takes any common noun (NC) as its nominal object; that the words

[7]The cframe and aframe methods are defined as the following abbreviations of the set method: type(*t*)−>set("cframe:*C*", cframe_object(*f*, $r_1$ => $c_1$, . . ., $r_n$ => $c_n$)) for the cframe method, and type(*t*)−>set("aframe:*A*", aframe_object(*m*, *r* => *g*)) for the aframe method. The methods cframe_object and aframe_object return objects representing complement frames and adjunct frames, respectively.

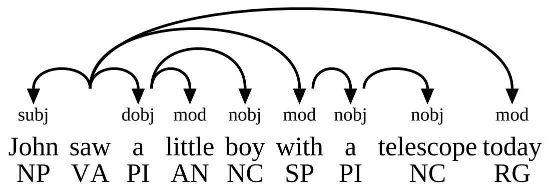

subj
dobj
mod
nobj
mod
nobj
nobj
mod
John saw a little boy with a telescope today
NP VA PI AN NC SP PI NC RG

Figure 2.8: A continuous analysis licensed by the lexicon in Figure 2.7.

"boy" and "telescope" are common nouns, and "John" a proper noun (NP); that "little" is a normal adjective (AN) that can modify any noun; that "saw" is a verb (VA) that takes two nouns as its subject and direct object; that "today" is an adverb (RG) that can modify any noun or verb; and that "with" is a preposition (SP) that can take any noun as its nominal object.

A lexicon specifies an analysis space by placing restrictions on what counts as well-formed dependency graphs. This notion of well-formedness differs from the notion of grammaticality: a highly ungrammatical analysis may be well-formed, ie, it may belong to the speaker's space of all conceivable analyses. One important restriction is that the deep graph associated with a dependency graph must be well-formed, ie, it must satisfy the complement and adjunct restrictions introduced by the lexemes associated with nodes in the graph.

**Definition 2.45** An analysis is *deeply well-formed* if (a) its deep graph consists of a set of trees; (b) each governor in the graph licenses its complements, ie, the current complement frame in the governor's active lexeme licenses all the complement edges; and (c) each adjunct in the graph licenses its adjunct governor, ie, the current adjunct frame in the adjunct's active lexeme licenses the adjunct dependency.

**Example 2.46** The lexicon in Figure 2.7 allows many analyses of the same sentence. Two well-formed analyses of the sentence "John saw a little boy with a telescope today" are shown in Figure 2.8 and 2.9. Figure 2.8 is a sensible linguistic analysis, whereas Figure 2.9 is a very poor analysis — in canonical word order, the analysis in Figure 2.9 corresponds to the sentence "A boy today with a telescope saw little John." The discontinuous example shows that complement and adjunct frames place no restrictions on word order, ie, we need a separate mechanism for controlling word order.

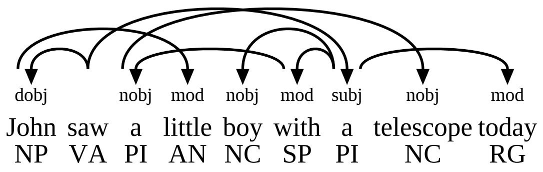

dobj
nobj
mod
nobj
mod
subj
nobj
mod
John saw a little boy with a telescope today
NP VA PI AN NC SP PI NC RG

Figure 2.9: A highly discontinuous analysis licensed by the lexicon in Figure 2.7, illustrating the need for controlling word order.

**Remark 2.47** The notions of governor, complement, adjunct, functor, argument, modifier, and deep well-formedness, and the principle of compositionality presented here are rather uncontroversial: almost all formal linguistic theories (including GB, HPSG, LFG, TAG, and Word Grammar) agree on these notions, although the theories may differ slightly with respect to the formal implementation.

Deep well-formedness ensures that a dependency graph has a well-formed complement-adjunct structure, but as is obvious from the highly discontinuous graph in Figure 2.9, deep well-formedness does not place any restrictions on word order. In section 2.4, we will therefore extend our lexicon so that it also imposes restrictions on word order. However, before proceeding with word order, we need to clarify the difference between complements and adjuncts within our theory.

**The difference between complements and adjuncts**

Although most linguistic theories distinguish between complements and adjuncts, it has proven difficult to find precise criteria for deciding whether particular phenomena should be analyzed as complement or adjunct constructions (cf. Vater (1978, 21–45), Helbig (1992, 72–98), Pustejovsky (1995, 63–65), Borsley (1996, 110–13)). This has led some authors (eg, Vater (1978, 39) and Somers (1987, 24–28)) to suggest that the complement-adjunct distinction is not a dichotomy, but a continuum of dependents with varying degrees of affinity to the governor, ranging from subjects on one end of the scale to negations like "not" on the other.

In the complement-adjunct debate, most linguists have implicitly assumed that the complement-adjunct distinction is made at the language level. However, we believe that much of the theoretical confusion about complements and adjuncts disappears if the distinction is made at the level of analyses or speakers instead, as defined below (cf. Kromann 1999b, 27–30):

- *language level distinction*: all speakers arrive at the same complement or adjunct analysis of a particular example.
- *speaker level distinction*: a speaker's grammar only allows either a complement analysis or an adjunct analysis of a particular example, but different speakers may have different analyses.
- *analysis level distinction*: each analysis considered by a speaker encodes the particular choices of complement and adjunct mechanisms in that analysis, but the speaker's grammar may allow both a complement and an adjunct analysis of a particular example.

In our view, language learning can be viewed as a mostly unconscious process where a child gradually constructs a compact mental model of the languages spoken by the groups of speakers with which the child interacts. The child will try to revise its model whenever it realizes that the model leads to misunderstandings or significant stylistic differences that might affect the child's social standing within a particular group of speakers. The random nature of this process means that speakers end up having slightly different grammars that change over time, and that speakers may conceivably have to maintain grammars that allow many competing analyses simultaneously — eg, during their transition from one model of what is grammatical to another, or because they must encode several slightly different sublanguages. If it is true that grammars are ambiguous and may differ between speakers, this suggests that the complement-adjunct distinction is best made at the analysis level.

A binary distinction between complements and adjuncts at the analysis level is perfectly compatible with a continuous scale at the language level that measures the proportion of speakers that choose to use a complement mechanism rather than an adjunct mechanism to analyze a given construction in a particular situation. There may be constructions where almost all speakers agree on one of the analyses (eg, all speakers may agree that subjects are complements, and temporal adverbials are adjuncts), but there may also be constructions where speakers are more or less evenly divided between the complement and adjunct analyses. Our conjecture is that many of the contentious examples in the complement-adjunct debate are of this kind.

This raises the question of why there are phenomena where speakers tend to agree, and other phenomena where the variation among speakers is large. Our explanation is that in principle, a speaker has a free choice between the complement and adjunct mechanisms when encoding a construction. But the free choice is tempered by the need for an encoding that is as compact as possible, to avoid wasting human brain capacity. As an extreme example, in a sufficiently sophisticated grammar formalism, any grammar can be encoded as a functionally equivalent grammar that only uses the complement mechanism, or only uses the adjunct mechanism, but the resulting grammars will be bad because they are unnecessarily large.[8]

Given a construction in a grammar $G$, we can measure the efficiency of a complement encoding over an adjunct encoding by comparing the size of two grammars $G_c$ and $G_a$ that encode the construction by a complement and adjunct mechanism, respectively, while being as small as possible among all grammars that are functionally equivalent to $G$. If $G_c$ is significantly smaller than $G_a$, then all speakers are likely to select a complement encoding, and vice versa. But if $G_c$ and $G_a$ have almost equal size, then both encodings are equally good, and the speakers will make a random choice between them.

Our theoretical account does not solve the practical problem of measuring the efficiency gain we get by using one mechanism rather than the other. It therefore remains an art to determine the most efficient encoding, and the many heuristic tests proposed in the complement-adjunct literature can be viewed as good practical guides. However, our theory provides conceptual clarity by explaining the linguistic intuition that some constructions are clearly complements, some are clearly adjuncts, and some are difficult borderline cases. It also tells us that in borderline cases, we have a free choice between the complement and adjunct mechanisms, thereby resolving a central question in the complement-adjunct debate.

In this section, we have defined complements and adjuncts, how they relate to functor-argument structure, and how they are encoded in the lexicon. We have also clarified the theoretical difference between complements and adjuncts. In the following section, we will show how word order can

[8]In a grammar using complements only, all verbs must specify a large number of complement frames that encode all possible combinations of subjects, objects, and adverbials. In a grammar using adjuncts only, all subjects and objects must encode the valency frames of all verbs, so that they only modify verbs where they can appear as dependents. Obviously, these encodings are very inefficient.

be controlled by means of a surface tree.

## 2.4 Surface trees: landing sites and word order

***Summary.*** *We show how word order can be controlled by means of a continuous surface tree. The edges in the surface tree encode the relationship between a node and its landing site whose lexical entry must license the landed node. The surface tree is linked to the deep tree by a deep upwards movement principle that ensures that a node's landing site dominates the node's governor in the deep tree. We show how the global word order can be determined by the continuity of the surface tree and weighted local word order constraints. Finally, we show how to construct surface trees with a minimal amount of upwards movement.*

In the previous section, we saw how deep trees are used to encode the dependency relations that determine a unique functor-argument structure given a modifier scope, and hence a unique semantic interpretation. We also saw in Example 2.46 that complement and adjunct frames in themselves are insufficient to control word order. In this section, we will therefore introduce the machinery needed to control word order.

In a dependency theory, it is natural to assume that the word order in a dependency graph must be controlled by lexical rules associated with the nodes in the graph. The simplest idea is to let the word order be controlled by the deep tree, ie, to let each governor be responsible for the relative word order of its complements and adjuncts. Unfortunately, this simple idea is too crude. The reason is that the global word order is not uniquely specified by a local ordering of the dependents of all governors because the potential discontinuity of the deep tree allows phrases to be intermingled. Without the discontinuity, the idea would work fine.

The obvious solution is to introduce a continuous surface tree where all discontinuous edges from the deep tree are replaced with continuous edges, ie, the continuous surface tree can be viewed as a tree that is obtained from the deep tree by moving discontinuous dependents upwards until all discontinuous edges have been eliminated. The node $l$ where a discontinuous dependent $n$ lands is called the *landing site* of the dependent, and $n$ is called the *landed node* of $l$. The landing site $l$ coincides with the governor $g$ of $n$ when there is no movement. The relationship between $l$ and $n$ is encoded with an edge $l \xrightarrow{r} n$, where the edge type $r$ is a subtype of the type surface, thereby indicating that the edge is a *surface edge*.

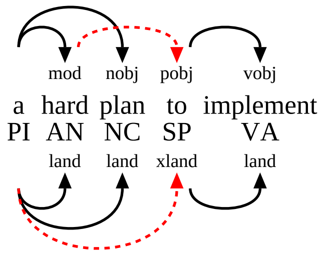


Figure 2.10: A discontinuous deep tree (top) and an associated continuous surface tree (bottom). The discontinuous dependency and its associated xland edge is shown with dotted lines.

**Definition 2.48** The subgraph $S(G)$ of a dependency graph $G$ consisting of all surface edges in $G$ is called the *surface forest* (or *surface graph*) associated with $G$. A connected component in the surface forest is called a *surface tree*.

**Example 2.49** Figure 2.10 shows a dependency analysis of the phrase "a hard plan to implement", consisting of a discontinuous deep tree (top) and an associated continuous surface tree (bottom). The surface tree is the result of moving up the dependent "to" from its governor "hard" to its landing site "a". The discontinuous edge and its replacement are shown with dotted arcs. The label land is used for surface edges where the governor and the landing site coincide, whereas the label xland (external landed node) is used for surface edges where they do not coincide, ie, where the node has been moved upwards.

We will assume that landed nodes must be lexically licensed by their landing sites, by means of a landing frame associated with each lexeme.

**Definition 2.50** A lexical type $t$ must have an associated *landing frame* of the form $(r_1 : n_1, \ldots, r_k : n_k)$, which licenses an arbitrary number of landing edges of the form $t \xrightarrow{r_i} n_i$ with landing edge type $r_i$ where the landed node matches the type specification $n_i$.

**Pseudocode 2.51** Landing frames are added to lexemes with the method:

```
type($t$)->lframe($r_1$ => $n_1$, ..., $r_k$ => $n_k$);
```

where $n_i$ is the type specification for landed nodes with edge type $r_i$. If $n_i$ equals null, then the landing edge rule for edge type $r_i$ is removed from the landing frame.[9]

**Example 2.52** Suppose we want to encode in our DTAG lexicon that words can act as landing sites for all their dependents by default, that finite verbs also allow non-dependents to land if the non-dependents are either nouns or verbs, and that infinite verbs allow all dependents except subjects to land. This can be encoded as follows (the type specification $[\overset{\text{deep}}{\longleftarrow}\ \text{this}]$ is satisfied by a node if it is a dependent of the node whose lexical entry contains the type specification):

| **Type** | **Super** | **Landing frame** |
|---|---|---|
| Word | | $\xrightarrow{\text{land}}$ : $[\overset{\text{deep}}{\longleftarrow}\ \text{this}]$ |
| V | Word | |
| V:fin | V | $\xrightarrow{\text{xland}}$ : $(\text{N}\vert\text{V})-[\overset{\text{deep}}{\longleftarrow}\ \text{this}]$ |
| V:inf | V | $\xrightarrow{\text{land}}$ : $[\overset{\text{deep}-\text{subj}}{\longleftarrow}\ \text{this}]$ |

The inheritance means that V:fin inherits the default land rule from Word, whereas V:inf overrides the default by disallowing subjects to land while allowing all other dependents to land.

**Example 2.53** Figure 2.11 shows an inheritance lexicon that licenses the analysis in Figure 2.10 of the phrase "a hard plan to implement." In the lexicon, we have assumed that adjectives only modify nouns that are not governed by a pronoun (P), ie, that adjectives attach to the determiner if there is one, and the noun otherwise.[10]

[9] The lframe method is defined as a short-hand for the following sequence of set statements: type($t$)−>set("lframe:$r_1$", $n_1$)−>...−>set("lframe:$r_k$", $n_k$).

[10] The main argument for the analysis of adjectives as modifiers of the determiner is that adjectives are marked for the form of the determiner in both Danish and German, whereas common nouns are unmarked. Analyzing the adjective as a modifier of the common noun would therefore result in non-local agreement. In Danish, the adjective receives a definite marking whenever the determiner is definite, and an indefinite marking whenever the determiner is indefinite:

| | | | |
|---|---|---|---|
| (1) | (et godt) | / (det gode) | eksempel |
| | (a good-INDEF) | / (the good-DEF) | example |

Similarly, in German, there are three determiner classes that affect the gender-case-number marking of all modifying adjectives, without affecting the gender-case-number marking of

| Type | Super | C-frame | A-frame | L-frame |
|---|---|---|---|---|
| Word | | | | $\xrightarrow{\text{land}}$ : $[\xleftarrow{\text{deep}}$ this$]$ |
| A | Word | | $\xleftarrow{\text{mod}}$ : N $-$ $[\xleftarrow{\text{nobj}}$ $P]$ | |
| N | Word | | | $\xrightarrow{\text{xland}}$ : SP$-$$[\xleftarrow{\text{deep}}$ this$]$ |
| SP | Word | | | |
| V | Word | | | |
| NC | N | | | |
| P | N | | | |
| a | P | $\xrightarrow{\text{nobj}}$ : NC | | |
| hard | A | $\xrightarrow{\text{pobj}}$ : to | | |
| plan | NC | | | |
| to | SP | $\xrightarrow{\text{vobj}}$ : V | | |
| implement | V | | | |

Figure 2.11: An inheritance lexicon for the words in Figure 2.10 with supertypes and complement, adjunct, and landing frames.

We have informally described surface trees as being obtained from deep trees by lifting discontinuous dependents to higher nodes in the deep tree. We can formalize this connection between deep trees and surface trees in terms of the deep upwards movement principle defined below. This principle holds in most other syntax formalisms as well, including GB, HPSG, LFG, TAG, and Word Grammar, although the details of the implementation may be slightly different.

**Definition 2.54** A dependency graph satisfies the *deep upwards movement principle* if the landing site of each node in the graph dominates the node's governor within the deep tree.

the common noun: the *weak inflection* is used with determiners in the "der" group, the *mixed inflection* is used with determiners in the "ein" group, and the *strong inflection* is used when the determiner is uninflected or absent.

| | | | | |
|---|---|---|---|---|
| (2) | (das gute) | / (ein gutes) | / (Schmidts gutes) | Beispiel |
| | (the good-WEAK) | / (a good-MIXED) | / (Schmidt's good-STRONG) | example |

Moreover, adjectives can modify determiners in the absence of common nouns in Danish, German, and to a lesser degree English: eg, "Give me your tired, your poor". Hudson (2003) assumes that the adjective depends on both determiner and common noun, a multiple dependency that is not possible within our formal framework. However, apart from this difference, our analysis of NPs is essentially identical to Hudson's.

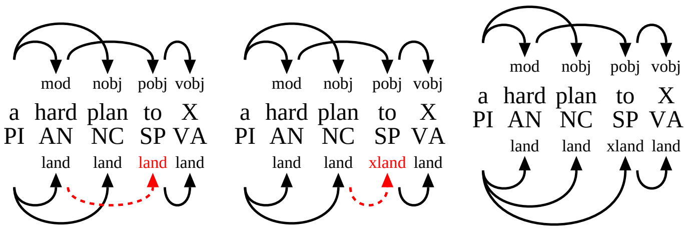


Figure 2.12: Three deep trees (top) and surface trees (bottom). The first surface tree (left) violates continuity, the second (middle) violates deep upwards movement, and the third (right) is surface well-formed.

We can now define the notion of a well-formed surface graph.

**Definition 2.55** A dependency graph is *surface well-formed* if (a) its surface graph consists of a set of continuous trees; (b) each landing site in the graph licenses its landed nodes, ie, each landing edge matches some entry in the landing frame (or filler frame, cf. Definition 2.80) associated with the landing site; and (c) the dependency graph satisfies the deep upwards movement principle.

**Example 2.56** Figure 2.12 shows three surface trees for the phrase "a hard plan to implement." The first surface tree (left) is ill-formed because it is discontinuous. The second surface tree (middle) is ill-formed because it violates the deep upwards movement principle: for the word "to", the landing site "plan" does not dominate the governor "hard" in the deep tree. The third surface tree (right) is well-formed.

The surface tree must be coupled with constraints that restrict the local ordering of the landed nodes at each landing site in order to ensure a correct global word order.

**Example 2.57** Figure 2.13 shows three dependency graphs that are well-formed according to the lexicon in Figure 2.11, although they sound ungrammatical or unnatural. This shows that the lexicon in Figure 2.11 is still incomplete. What we need are rules that restrict the local ordering of the landed nodes at each landing site. These rules could be implemented as landing frame rules, but since speakers can make small, unpredictable errors with respect to local word order, it is more convenient to implement

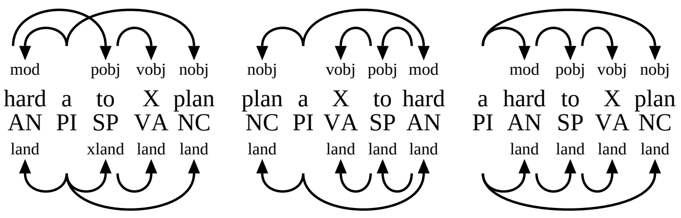


Figure 2.13: Three examples of well-formed dependency graphs for phrases that are ungrammatical because of incorrect local word order (left and middle) or incomplete landing rules (right).

word order constraints as violable weighted constraints, ie, constraints that affect the ill-formedness measure without reducing the size of the analysis space. Violable weighted constraints are the topic of chapter 3, where we will show how they can be used to deal with word order, agreement, etc., so here we will only give a preview of ideas to come.

The intuition behind violable weighted constraints is that each node in the dependency graph has an associated set of weighted constraints that are specified in the node's lexeme and evaluated at the node, triggering a fixed cost for each violation. These localized costs encode the precise type, location, and severity of grammatical violations in the graph. Weighted constraints are given in the form $c * s$ where $c$ is a non-negative real-valued cost, and $s$ is a type specification that matches nodes with violating configurations. The total cost returned by a weighted constraint $c * s$ is $cn$ where $n$ is the number of different subgraphs that match the type specification $s$.

The ungrammaticality of the analyses in Figure 2.13 can be explained if we assume that a speaker of English has a grammar with the weighted constraints (a)–(f) below. For simplicity, we will assume that the cost 100 indicates a completely ungrammatical configuration, and that the cost 50 indicates an unnatural configuration (in DG, a speaker's space of analyses may include highly ungrammatical analyses). In chapter 6, we will then return to the issue of how word order costs can be assigned by means of a probabilistic language model rather than hand-written cost functions.

- **(a) P:** $100 * ([\xleftarrow{\textsf{surface}} \textsf{this}] + [< \textsf{this}])$: pronouns with preceding landed nodes are ungrammatical.
- **(b) P:** $100 * ([\xleftarrow{\textsf{surface}} \textsf{this}] + \textsf{A} + [> [\xleftarrow{\textsf{nobj}} \textsf{this}] + [\xleftarrow{\textsf{surface}} \textsf{this}]])$: pronouns with landed adjectives that succeed landed nominal objects

| | 2.12 right | 2.13 left | 2.13 middle | 2.13 right |
|---|---|---|---|---|
| **(a)** | 0 | 100 | 100 | 0 |
| **(b)** | 0 | 0 | 100 | 0 |
| **(c)** | 0 | 100 | 0 | 0 |
| **(d)** | 0 | 0 | 100 | 0 |
| **(e)** | 0 | 0 | 100 | 0 |
| **(f)** | 0 | 0 | 0 | 50 |
| **Total** | 0 | 200 | 400 | 50 |

Figure 2.14: The violation costs for the analyses in Figure 2.12 and 2.13 computed from the weighted constraints (a)–(f) in Example 2.57.

are ungrammatical.

- **(c) P: 100** $*\ ([\xleftarrow{\textsf{surface}}\textsf{this}] + \textsf{SP} + [< [\xleftarrow{\textsf{surface}}\textsf{this}] + (\textsf{N}|\textsf{AN})])$: pronouns with landed prepositions that precede landed nouns or adjectives are ungrammatical.
- **(d) SP: 100** $*\ ([\xleftarrow{\textsf{surface}}\textsf{this}] + [< \textsf{this}])$: prepositions with preceding landed nodes are ungrammatical.
- **(e) A: 100** $*\ ([\xleftarrow{\textsf{surface}}\textsf{this}] + \textsf{SP} + [< \textsf{this}])$: adjectives with preceding landed prepositions are ungrammatical.
- **(f) A: 50** $*\ ([\xleftarrow{\textsf{surface}}\textsf{this} + [\xleftarrow{\textsf{mod}}\textsf{N}]] + [> \textsf{this}])$: adjectives with a succeding landed node are unnatural if they modify a noun.

These rules are only an approximation to English word order rules, since there are constructions where these rules are too restrictive (eg, “How good an idea is it?”). Figure 2.14 shows the violation costs associated with each analysis; from the total cost, we see that the phrase “a hard plan to implement” (cost 0) is more well-formed than the phrase “a hard to implement plan” (cost 50), which is in turn significantly more well-formed than the phrases “hard a to implement plan” (cost 200) and “plan a implement to hard” (cost 400).

We can always construct a continuous surface tree by letting all nodes land on the root node of the deep tree. However, this surface tree is unnecessarily flat, and therefore results in unnecessarily complex local word order rules. From a linguistic point of view, we usually get the simplest local word order rules by using a continuous surface tree with as little lifting as possible. This idea is formalized below.

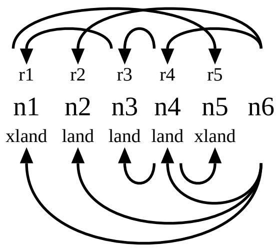


Figure 2.15: A discontinuous deep tree (top) and the associated minimal continuous lifting (bottom).

**Definition 2.58** Let $T$ be a deep tree. The *minimal continuous lifting* $T_{\text{mcl}}$ of $T$ is defined as the surface tree obtained from $T$ by replacing each edge $g \xrightarrow{r} d$ in $T$ with the edge $l \xrightarrow{s} d$ where $l$ is the lowest transitive governor of $d$ such that $Y_T(l)$ contains each node $b$ that lies between $l$ and $d$ in the linear word order, where $s$ is set to $\mathsf{land}$ if $g = l$ and $\mathsf{xland}$ if $g \neq l$.

**Example 2.59** The surface trees in Figure 2.12 (right) and Figure 2.13 are minimal continuous liftings of the corresponding deep trees. A more complex example of a minimal continuous lifting is shown in Figure 2.15.

**Remark 2.60** When drawing DG analyses, we will often omit most or all of the surface edges. When a surface edge is omitted, we will assume that it is identical to the surface edge given by a minimal continuous lifting.

We will now prove that the minimal continuous lifting is what its name suggests: a continuous tree with a minimal amount of lifting.

**Proposition 2.61** *The minimal continuous lifting $T_{mcl}$ associated with a deep tree $T$ is a continuous tree with a minimal amount of lifting among all continuous liftings of $T$.*

*Proof.* To prove that $T_{\text{mcl}}$ is a tree, we must prove that $T_{\text{mcl}}$ is acyclic and connected. The acyclicity follows by observing that a cyclic path in the surface tree $T_{\text{mcl}}$ translates into a cyclic path in the deep tree $T$ if we replace each edge $l \xrightarrow{r} n$ in the cycle in $T_{\text{mcl}}$ with the unique downwards path in $T$ from the landing site $l$ to the landed node $n$. The connectedness follows by observing that a node's landing site must be chosen among the node's transitive governors so that all nodes in $T_{\mathrm{mcl}}$ are surface dominated by the root node of $T$.

We prove the continuity of $T_{\mathrm{mcl}}$ by contradiction. Assume $l \xrightarrow{r} d$ is a discontinuous edge in $T_{\mathrm{mcl}}$. By definition of discontinuity, there exists a node $d'$ between $l$ and $d$ such that $d' \notin Y_{T_{\mathrm{mcl}}}(l)$. Lemma 2.62 shows that if there is one node between $l$ and $d$ that is not in $Y_{T_{\mathrm{mcl}}}(l)$, then all its transitive landing sites also lie between $l$ and $d$ and are not in $Y_{T_{\mathrm{mcl}}}(l)$, ie, there are infinitely many nodes between $l$ and $d$. This contradicts that deep trees are finite, thereby proving that $T_{\mathrm{mcl}}$ is continuous.

That $T_{\mathrm{mcl}}$ has the smallest amount of lifting among all continuous trees that have been constructed by lifting edges within the deep tree is obvious from the definition of discontinuity and $T_{\mathrm{mcl}}$. □

**Lemma 2.62** *Suppose $d$ is a node in $T_{mcl}$ with landing site $l$. If $d'$ lies between $l$ and $d$, and $d' \notin Y_{T_{mcl}}(l)$, then the landing site $l'$ of $d'$ also lies between $l$ and $d$, and also satisfies $l' \notin Y_{T_{mcl}}(l)$.*

*Proof.* We will assume without loss of generality that $l < d$. Suppose $d'$ satisfies $l < d' < d$ and $d' \notin Y_{T_{\mathrm{mcl}}}(l)$. There are five cases to consider: (a) If $l' < l$, then $l' < l < d'$ so that $l \in Y_T(l')$ by definition of $l'$. Since $l < d' < d$ and $Y_T(l)$ contains all nodes between $l$ and $d$, we see that $Y_T(l)$ also contains all nodes between $l$ and $d'$, ie, $l$ is a potential landing site for $d'$. Since $l'$ is the minimal potential landing site, we have $l' \in Y_T(l)$, which combined with $l \in Y_T(l')$ gives $l = l'$, a contradiction. (b) If $l' = l$, then $d' \in Y_{T_{\mathrm{mcl}}}(l') = Y_{T_{\mathrm{mcl}}}(l)$, a contradiction. (c) If $l < l' < d$, then the lemma holds since $l'$ lies between $l$ and $d$, and $d' \notin Y_{T_{\mathrm{mcl}}}(l)$ implies $l' \notin Y_{T_{\mathrm{mcl}}}(l)$. (d) If $l' = d$, we have $Y_{T_{\mathrm{mcl}}}(l') = Y_{T_{\mathrm{mcl}}}(d)$ which is a subset of $Y_{T_{\mathrm{mcl}}}(l)$, so $d' \notin Y_{T_{\mathrm{mcl}}}(l)$ implies $d' \notin Y_{T_{\mathrm{mcl}}}(l')$, a contradiction. (e) If $d < l'$, then $d' < d < l'$ so that $d \in Y_{T_{\mathrm{mcl}}}(l')$ by definition of $l'$, ie, $l'$ is a transitive governor of $d$. Since $l$ and not $l'$ is the minimal landing site of $d$, there exists $b$ with $d < b < l'$ and hence $d' < b < l'$ such that $b \notin Y_T(l')$, which contradicts the definition of $l'$. The proof shows that (c) is the only case that does not lead to contradiction, and hence that the lemma always holds. □

We will now define a number of concepts that we need in later sections. We start by defining the notion of extraction path used in Figure 2.2.

**Definition 2.63** Let $n$ be a node with governor $g$ and landing site $l$. Then the upwards path in the deep graph from $g$ to $l$ (or the zero path starting and ending at $g$ if $g = l$) is called the *extraction path* of $n$.

**Definition 2.64** Let $G$ be a graph. The type specification $\mathsf{extract}(a)$ matches any node in $G$ whose extraction path contains a node or edge that matches the type specification $a$.

**Example 2.65** In Figure 2.15, the node $n_1$ has extraction path $n_3 \xleftarrow{r_3} n_4 \xleftarrow{r_4} n_6$, the node $n_5$ has extraction path $n_1 \xleftarrow{r_1} n_3 \xleftarrow{r_3} n_4$, and the nodes $n_2$, $n_3$, $n_4$, and $n_6$ have a zero length extraction path consisting of their governor node.

We will now define the notions of surface yield, linear adjacency, and graph adjacency.

**Definition 2.66** The *surface yield* of a node $n$ in a graph $G$ is the yield of $n$ in the surface graph of $G$.

**Definition 2.67** Two nodes $n, n'$ in a graph $G$ are called *linearly adjacent* if they are adjacent in the linear order of $G$, ie, if $n$ immediately precedes $n'$, or $n'$ immediately precedes $n$.

**Definition 2.68** The *linear distance* (or *distance*) between two nodes $n$ and $n'$ is defined as the number of nodes between $n$ and $n'$.

**Definition 2.69** Two nodes $n, n'$ in a graph $G$ are called *yield-adjacent* if the surface yields of $n$ and $n'$ are disjoint, but adjacent to each other, ie, neither node dominates the other within the surface graph, and it is possible to find a node $n_1$ in the surface yield of $n$ that is adjacent to a node $n'_1$ in the surface yield of $n'$.

We will now define the notion of immediate precedence and succession, and previous and next node, used in Figure 2.2.

**Definition 2.70** Let $G$ be a graph, let $n$ be a node in $G$, and let $S$ be a set of nodes in $G$. Then a node $n' \in S$ is said to *immediately precede* $n$ within $S$ provided $n' < n$ and there is no other node $n'' \in S$ such that $n' < n'' < n$. Similarly, $n'$ is said to *immediately succeed* $n$ within $S$ provided $n < n'$ and there is no other node $n'' \in S$ such that $n < n'' < n'$.

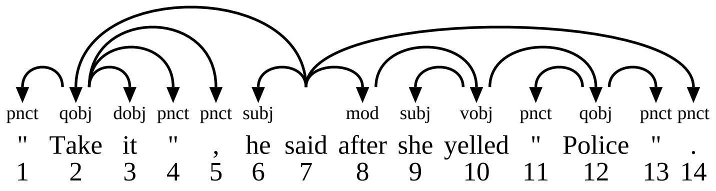


Figure 2.16: A DG analysis of an example with punctuation.

**Definition 2.71** Let $G$ be a graph, let $n$ be a node, and let $a$ and $b$ be type specifications. Then the type specification $\mathsf{prev}(a, b)$ evaluated at $n$ matches any node $n'$ that immediately precedes any node matching $b$, within the set of all nodes that match $a$. Similarly, the type specification $\mathsf{next}(a, b)$ evaluated at $n$ matches any node $n'$ that immediately succeeds any node matching $b$, within the set of all nodes that match $a$.

When encoding punctuation rules, it is convenient to have a notion of left and right periphery.

**Definition 2.72** Let $G$ be a graph, and let $n$ be a node in $G$. The *left boundary* of $n$ is the left-most node in the surface yield of $n$. Similarly, the *right boundary* of $n$ is the right-most node in the surface yield of $n$.

**Definition 2.73** Let $G$ be a graph containing a node $n$. The *left* $(i, o)$*-periphery* of $n$ is the largest adjacent set $L$ of nodes in $G$ such that (a) a node in $L$ within the surface yield of $n$ must match the type specification $i$; (b) a node in $L$ outside the surface yield of $n$ must match the type specification $o$; (c) if $L$ is non-empty, then $L$ must contain either the left boundary of $n$ or the node immediately to the left of the left boundary; and (d) $n$ is not a member of $L$. The *right* $(i, o)$*-periphery* is defined similarly, with "right" replacing "left".

**Example 2.74** To illustrate the definition of left and right periphery, we calculate the left and right $(s, s)$-periphery for some of the words in Figure 2.16, where $s$ is the type specification $[\xleftarrow{\mathsf{pnct}} \mathsf{any}]$. The results are shown in Figure 2.17. The main motivation for introducing the notions of left and right periphery is to deal with punctuation. In general, controlling punctuation is a challenge in any linguistic theory. For example, in sentence (3), a comma must obligatorily attach itself as an adjunct of "yelled":

| word | left $(s,s)$-periphery | right $(s,s)$-periphery |
|---|---|---|
| 1: “ | | |
| 2: Take | 1: “ | 4–5: ” , |
| 3: it | | 4–5: ” , |
| 4: ” | | 5: , |
| 5: , | 4: ” | |
| 7: said | 1: “ | 13–14: ” . |
| 10: yelled | | 13–14: ” . |

Figure 2.17: The left and right $(s,s)$-periphery for some of the nodes in Figure 2.16.

(3) After she yelled “Police!”, he said “Take it!”

But in Figure 2.16, the comma is obligatorily absent because of the sentence-final period, ie, our punctuation rules must have the effect that a finite verb has exactly one unary punctuation mark in its right periphery. We will return to the treatment of punctuation within DG in section 4.6.

**Comparison with other approaches to word order**

Most linguistic theories essentially agree on the notion of dependency structure described in section 2.3, but differ with respect to how they control word order. We originally constructed the word order theory in section 2.4, where surface trees are constructed by moving nodes upwards to a landing site, as a discontinuous dependency-based hybrid between the extraction theory in HPSG and GB's notion of movement, S-structure, and D-structure (cf. Chomsky 1965; Haegeman 1994/1991; Freidin 1991). However, we have departed from GB in that we construct the surface tree by moving edges while keeping nodes fixed, whereas GB changes node labellings while keeping edges fixed. We have departed from HPSG by using a surface tree based on dependency structure rather than phrase structure (ie, the surface tree does not contain any phrasal nodes).

The distinction between deep trees and surface trees is shared by most linguistic theories in some form, and surface trees and deep trees are usually designed so that the upwards movement principle holds. For example, although theories like HPSG and LFG have distanced themselves from the Chomskyan notion of movement, LFG (Dalrymple et al. 1994; Bresnan 2001) makes an explicit distinction between a phrase-structure based c-structure (which roughly corresponds to a surface tree) and an acyclic f-structure (which roughly corresponds to a deep tree), whereas HPSG (Pollard and Sag 1994; Bouma et al. 2001; Müller 2004) embodies an implicit distinction between phrase structure trees (surface trees) and the dependency trees encoded in the the valency features (deep trees). Similarly, deep trees and surface trees have natural equivalents in Tree-Adjoining Grammar (Abeillé and Rambow 2000) and Categorial Grammar (Steedman 2000).

The dependency-based word order theory we have presented seems to have been developed independently within three different dependency frameworks, using essentially identical surface trees, but slightly different word order rules: first by Hudson (1998, 2003), then by Kromann (1999a,b, 2001), and finally by Duchier and Debusmann (2001) and Duchier (2001). The most important differences are described below.

**Remark 2.75** The word order theory sketched in Hudson (1998) resembles our word order theory by being dependency-based and positing a continuous surface tree. The main differences between the two theories is that Hudson uses a different set of local word order rules and assumes that the path between the landed node and its landing site is encoded with an 'extractee' edge from each node on the path to the landing site. In 2003, his word order theory was changed so that a landed node may have more than one landing site (called a *landmark* in his new theory), which does not necessarily have to be a transitive governor of the landed node.

**Remark 2.76** The word order theory presented in Duchier and Debusmann (2001) and Duchier (2001) is almost identical to the theory we have presented in Kromann (1999a,b, 2001). The most important difference between the two theories is that the word order theory by Duchier and Debusmann is based on topological field theory (cf. Bech 1955/1983; Gerdes and Kahane 2001; Harbusch and Kempen 2002) and only involves inviolable constraints, rather than the violable weighted constraints used in our theory. It is possible to encode a grammar based on topological field theory within our framework, by assigning a unique surface edge label to each field. The fixed linear order of the fields can then be encoded by means of cost functions that assign a cost whenever two surface edges violate the relative order of the fields.

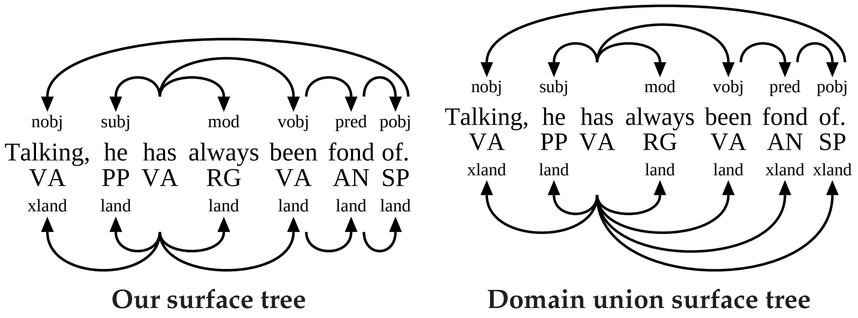

nobj
subj
mod
vobj
pred
pobj
Talking, he has always been fond of.
VA PP VA RG VA AN SP
xland land land land land land
Our surface tree
nobj
subj
mod
vobj
pred
pobj
Talking, he has always been fond of.
VA PP VA RG VA AN SP
xland land land land xland xland
Domain union surface tree

Figure 2.18: The surface tree proposed in our theory (left), and the flat surface tree resulting from domain unions (right).

It is also interesting to compare our word order theory with the notion of domain unions proposed by Reape (1994).

**Remark 2.77** In our theory, whenever we have a discontinuous dependent, we construct a continuous surface tree by moving up the discontinuous dependent within the deep tree. Reape (1994) has proposed a slightly different strategy based on *domain unions*, which has also been adopted within a dependency framework by Gerdes and Kahane (2001). In Reape's proposal, we move up the discontinuous dependent by flattening the local tree containing the dependent — ie, we move up the dependent and its siblings so that they become siblings of the governor, and repeat this flattening procedure until the node reaches its final landing site. Reape's approach involves moving a larger number of nodes than in our theory, which results in significantly flatter surface trees and correspondingly more complex word order rules. Figure 2.18 shows our surface tree for the sentence "Talking, he has always been fond of" (which coincides with the minimal continuous lifting) and the corresponding domain union surface tree.

In our theory, we have assumed that movement takes place within the deep tree, but there are also theories where movement takes place within the surface tree.

**Remark 2.78** In *deep movement*, a word's landing site must dominate the word's governor in the deep tree, whereas in *surface movement*, the landing site must dominate the governor in the surface tree. Surface movement is more restrictive than deep movement, since the surface tree can be seen as a flattened version of the deep tree. Our movement theory is based on deep movement, whereas slash-propagation in classical HPSG and movement in GB correspond to surface movement. Interestingly, Bouma, Malouf, and Sag (Bouma et al. 2001, §3.1) have recently proposed changing slash-propagation within HPSG so that it corresponds to deep movement.

In this section, we have shown how word order can be controlled by means of a continuous surface tree where the landing sites are responsible for ordering their landed nodes, and we have formulated a deep upwards movement principle that relates the surface tree with the deep tree. In the next section, we will show how to encode secondary dependencies, ie, dependencies where a lexeme satisfies more than one dependency.

## 2.5 Fillers and gapping: secondary dependencies

***Summary**. We introduce phonetically empty fillers to explain how lexemes can fill more than one dependent role in constructions such as verbal complexes, relatives, control constructions, and gapping coordinations. Fillers are lexically licensed by a filler licensor, a special landing site that identifies a filler source in the dependency graph from which the syntactic and semantic properties of the filler can be computed. The fillers can also be used to encode gapping conjuncts where some of the dependencies in the filler source tree have been replaced by material from the gapping conjunct.*

In section 2.3, we saw that lexemes are equipped with complement and adjunct frames that license the dependency relations among the lexemes in a dependency graph, and that a lexeme cannot have more than one governor in a well-formed deep tree. In this section, we will show how to uphold this assumption while providing analyses for constructions such as verbal complexes, relatives, and control constructions, where a lexeme with a primary governor seems to satisfy a complement or adjunct role of one or more secondary governors.

For example, in the verbal complex (4) below, the word "truth" seems to satisfy the subject roles of both "will" and "prevail". Similarly, in the control construction (5), "the opera" seems to satisfy both the direct object role of "want" and the subject role of "stop". Finally, in the relative construction (6), "the painting" seems to satisfy both the subject role of "was" and the direct object role of "bought".

(4) Truth will always prevail. (verbal complex)
(5) We want the opera to stop right now. (control)
(6) The painting we bought was a fake. (relative)

A lexeme can even satisfy a dependent role in arbitrarily many governors, as shown by (7), where "they" satisfies the subject role of all verbs in the sentence: "should", "have", "been", "permitted" (analyzed as a passive form), "seek", and "agree".

(7) They should have been permitted to seek to agree on a compromise.

A simple idea for dealing with these examples would be to assume that lexemes can have arbitrarily many governors; but if we did that, "John likes" could be interpreted to mean "John likes John", where "John" acts as the simultaneous subject and object of "likes", and this is not what we want. As in other syntactic frameworks, the key to explaining secondary dependencies is to note that a secondary dependency is not licensed by some particular property of the dependent or its secondary governor, but is lexically licensed by a third lexeme, called the *filler licensor*, that acts as an intermediary between the secondary dependent and its secondary governor. For example, it is a property of a modal verb like "will" that it passes on its subject to its verbal complement; it is a property of a control verb like "want" that it passes on its direct object as the subject of the infinitival verb within its "to" complement; and it is a property of any verb that attaches itself as a relative modifier to a noun that it passes on the relativized noun as a secondary dependent somewhere within the relative clause.

In our framework, we will assume that a filler licensor acts as a special landing site that establishes a secondary dependency by creating a phonetically empty placeholder (called a *filler*) for another node (called the *filler source*). The governor of the filler is sometimes called a *secondary governor* of the filler source, and the filler source is called its *secondary dependent*. The relationship between the filler, filler licensor, and filler source is encoded by means of two edges: a surface edge with edge type $r_l$ from the filler licensor to the filler, and a source edge with edge type $r_s$ from the filler source to the filler, as shown schematically in Figure 2.19. The edge type $r_l$ must be a subtype of the type fill, and $r_s$ must be a subtype of the type source (cf. Figure 2.20). Fillers must be lexically licensed by a filler frame associated with the filler licensor, as defined below.

**Definition 2.79** A *filler triple* is a triple $f = (f_{\text{type}}, f_{\text{fill}}, f_{\text{src}})$ where $f_{\text{type}}$ is

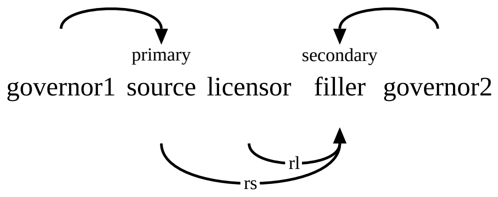


Figure 2.19: The schematic dependency encoding of the relationship between a filler, filler licensor, and filler source in a dependency graph.

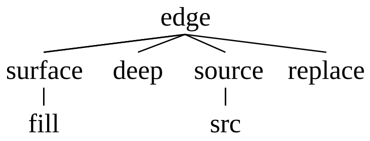


Figure 2.20: Partial hierarchy for edges used in filler constructions.

the filler lexeme assigned to the filler node, $f_{\text{fill}}$ is the fill type assigned to the edge from the filler licensor to the filler, and $f_{\text{src}}$ is the source type assigned to the edge from the filler to the filler source.

**Definition 2.80** A lexical type $t$ can have zero or more associated *filler frames* of the form $(s, f)$ where $s$ is a type specification that identifies all potential nodes in the dependency graph that can function as filler source for the filler, and $f$ is a filler triple.

**Definition 2.81** A filler node $n_f$ in a dependency graph $G$ is called *filler well-formed* if (a) it has exactly one incoming filler edge and one incoming source edge, namely $n_l \xrightarrow{r_l} n_f$ and $n_s \xrightarrow{r_s} n_f$, respectively; (b) there is no other filler in $G$ with the same filler licensor, filler source, and filler licensor and filler source edge types, ie, there is no other filler $n'_f$ in $G$ with edges $n_l \xrightarrow{r_l} n'_f$ and $n_s \xrightarrow{r_s} n'_f$; and (c) $n_l$ has a filler frame $(s, f)$ with $f = (f_{\text{type}}, r_l, r_s)$ such that $f_{\text{type}}$ is the lexeme assigned to $n_f$, and the filler source $n_s$ satisfies the type specification $s$, evaluated at the filler licensor $n_l$.

**Remark 2.82** Fillers are optional by default, ie, the presence of a filler frame does not mean that the lexeme has to generate a filler, only that it can do so. The obligatory presence of a filler must therefore be controlled by means of a cost function that can impose a cost if the lexeme fails to generate a filler.

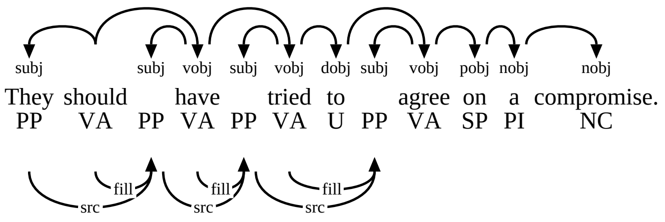

subj
subj
vobj
subj
vobj
dobj
subj
vobj
pobj
nobj
nobj
They should have tried to agree on a compromise.
PP VA PP VA PP VA U PP VA SP PI NC
fill
src
fill
src
fill
src

Figure 2.21: The use of fillers in constructions with verbal complexes and control.

The obligatory absence of a filler can be controlled with cost functions as well.

**Pseudocode 2.83** Filler frames are specified with the method:

$$\text{type}(t)\text{−>fframe}(F,\ s,\ f)$$

where $F$ is a unique identifier for the filler frame, $s$ is a type specification, and $f$ is a filler triple $[ftype, ffill, fsrc]$ where *ftype* is a type name, and *ffill* and *fsrc* are edge types.

**Example 2.84 (verbal complexes and control)** Figure 2.21 shows an analysis involving fillers of a construction with verbal complexes and subject control. All three fillers are created with a filler frame of the form:

$$\text{fframe(subj\_filler, } [\xleftarrow{\text{subj}} \text{this}]\text{, [simple\_filler, fill, src])}$$

This filler frame, which is specified in the lexical entry of the verbs "should", "have", and "tried", licenses the creation of a filler with type simple_filler where the filler licensor's subject acts as the filler source, using the edge labels fill and src for the edges from the filler licensor and filler source to the filler. In the analysis, the verb "should" creates a copy of the subject (ie, a filler), and passes it on to "have", which in turn creates a copy of the copy and passes it on to "tried", which finally creates a copy of the second copy and passes it on to "agree". That is, verbal complexes and control constructions receive the same syntactic analysis.

The filler frame in itself only licenses the creation of the filler — it does not guarantee that the filler is actually created (since fillers are optional by default), nor does it guarantee that the filler will act as subject of the

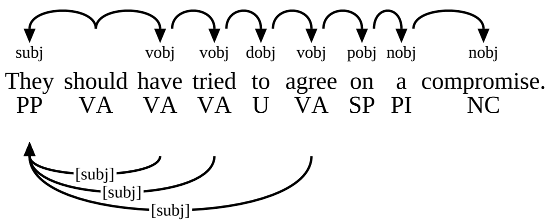

subj
vobj
vobj
dobj
vobj
pobj
nobj
nobj
They should have tried to agree on a compromise.
PP VA VA VA U VA SP PI NC
[subj]
[subj]
[subj]

Figure 2.22: The simplified DG graph corresponding to Figure 2.21.

subordinate verb rather than, say, the direct object. To ensure that these conditions are met, the lexical entries of the filler licensors must specify cost functions that impose penalties on analyses that violate these principles. These cost functions are spelled out in detail in our analyses of verbal complexes, raising, and control in section 4.3, and in our probabilistic language model in section 6.2.

**Remark 2.85** To make DG graphs easier for humans to create and read, the Danish Dependency Treebank (Kromann and Lynge 2004) uses simplified DG graphs where all fillers have been left out. Instead of specifying the filler directly, the secondary dependency between the filler source and the secondary governor is indicated by a dependency label of the form "[*r*]", where *r* is the dependency role that the filler satisfies in the secondary governor. This encoding contains information about the filler source and the dependency role satisfied by the filler, but does not contain any information about the filler licensor or the edge types (which are normally, but not always, easy to reconstruct automatically). The simplified visualization corresponding to Figure 2.21 is shown in Figure 2.22.

**Example 2.86 (simple fillers)** A *simple filler* is a filler that cannot take any complements or adjuncts, but which can satisfy all dependency roles that could have been satisfied by the filler source. Simple fillers are used in most of our analyses involving fillers, including our analyses of verbal complexes, raising and control, relative clauses, parasitic gaps, and peripheral sharing, which are presented in more detail in sections 4.3 and 4.4. One possible lexical entry for simple fillers is shown below. It specifies by means of inheritance that a simple filler is a filler that has an empty complement frame that returns a semantic variable that is coreferent with the filler source (variable_from_source), that copies all adjunct frames from the filler source (aframes_from_source), and which copies its dynamic supertypes from the filler source (super_from_source); the no_dependents cost function specifies that it is ungrammatical for the filler to have any dependents.

```
type(simple_filler)
  −> super(filler, variable_from_source, aframes_from_source,
             super_from_source)
  −> cost(no_dependents, 100 * [$\xleftarrow{\text{deep}}$ this]);
```

The supertype variable_from_source specifies that the filler has a single complement frame with no complements whose semantics is specified as a variable that is coreferent with the filler source (we assume that the function sem_variable($node) returns a semantic variable that is coreferent with $node).

```
type(variable_from_source)
  −> setd(variable_from_source, ’cframe:’, sub {
        my ($G, $n) = @_;
        my $source = $G−>get_node($n, [$\xrightarrow{\text{source}}$ this]);
        return { ’cframe:source’ =>
           cframe_object(sem_variable($source)) };
        });
```

The supertype aframes_from_source specifies that the filler's adjunct frames must be computed dynamically from the filler source by means of a dynamic feature function, implemented as a Perl subroutine that first finds the filler source, and then returns all dynamic adjunct frames associated with the filler source (or an empty hash without any adjunct frames if no filler source was found).

```
type(aframes_from_source)
  −> setd(aframes_from_source, ’aframe:’, sub {
        my ($G, $n) = @_;
        my $source = $G−>get_node($n, [$\xrightarrow{\text{source}}$ this]);
        return $source ? $G−>getd($source, ’/^aframe:/’) : {};
     });
```

Finally, the supertype super_from_source specifies that the filler's dynamic super feature must be computed by prepending the type and supertypes of the filler source to the filler's list of static supertypes.

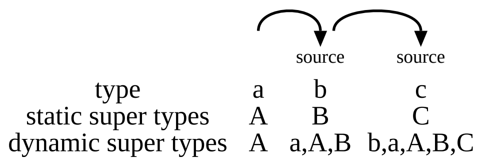


Figure 2.23: The computation of the dynamic super feature in simple fillers.

```
type(super_from_source)
   −> setd(super_from_source, super, sub {
         my ($G, $n) = @_;
         my $source = $G−>get_node($n, [$\xrightarrow{\text{source}}$ this]);
         if ($source) {
            return [$G−>get($source, type),
               @{$G−>get($source, super)},
               @{$G−>get($n, super)} ];
         } else {
            return $G−>get($n, super);
         }
      });
```

The computation of the dynamic super feature in simple fillers is exemplified in Figure 2.23, where a type *a* with static and dynamic supertypes *A* acts as filler source for a simple filler with type *b* and static supertypes *B*, which results in *b* having dynamic supertypes *a*, *A*, *B*. Similarly, the simple filler *c*, which has *b* as its filler source, has dynamic supertypes $b, a, A, B, C$. Consequently, if *a* satisfies the type specification isa$(t)$, then so will *b*, *c*, and all other fillers that are generated from *a* via a chain of simple fillers, unless isa$(t)$ explicitly refers to properties that are affected by the presence of the supertypes *b*, *c*, *B*, or *C*. This ensures that if the filler source is (say) an accusative noun, then the filler will also pose as an accusative noun.

While simple fillers can account for the secondary dependencies in verbal complexes, control constructions, relatives, parasitic gaps, and coordinations with peripheral sharing, they are not powerful enough to deal with cases where the same head has more than one set of dependents, as in gapping coordinations (eg, “We must tutor John, you Alice” or “I had tea and then coffee”) and comparatives (eg, “I love tea more than you coffee”). The gapping mechanism is one of the technically most complicated aspects of

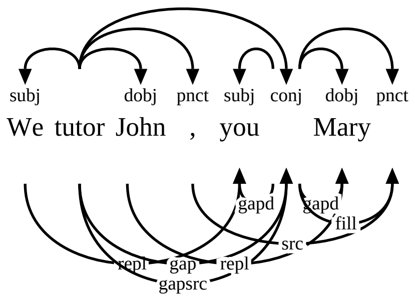


Figure 2.24: DG analysis of a gapping coordination.

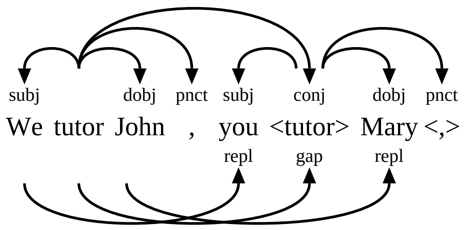


Figure 2.25: Simplified visualization of the DG analysis in Figure 2.24.

the DG syntax formalism. An example of a DG analysis of gapping constructions is shown below.

**Example 2.87** Figure 2.24 shows a DG analysis of the gapping coordination "We tutor John, you Mary". The head "tutor" of the first conjunct creates a gapping filler that takes "you" as its subject (as a replacement for "we") and "Mary" as its direct object (as a replacement for "John"). The gapping filler also generates a filler that represents the adjunct "," in the first conjunct. Instead of drawing all filler licensor edges and filler source edges, it is convenient to introduce the simplified visualization of gapping constructions shown in Figure 2.25, where we have omitted all filler licensor edges and filler source edges, except for the filler licensor edge for the gap, and used the node label <$w$> to encode the filler source $w$. Henceforth, we will use the simplified visualization when drawing DG analyses of gapping coordinations.

In DG, gapping constructions are controlled by means of gapping frames that copy the missing words into the gapped phrase from the filler source tree. Gapping frames are defined formally below.

**Definition 2.88** A lexical type $t$ can have zero or more associated *gapping frames* of the form $(s, f, i, b, l, r)$ where $s$ is a type specification that identifies all potential nodes in the dependency graph that can be licensed as the filler source for the gapping filler, $f$ is the filler triple used to create the gapping filler, $i$ is the filler triple used to create internal fillers, $b$ is the filler triple used to create boundary fillers, $l$ is the landing edge type used for gapping dependents, and $r$ is the replacement edge type.

**Pseudocode 2.89** Gapping frames are implemented as filler frames with additional arguments, and are specified with the method:

```
type(t)−>fframe(F, s, f, i, b, l, r)
```

where *F* is a unique identifier for the filler frame, *s* is a type specification, *r* is a replacement edge type, *l* is a landing edge type, and *f*, *i*, *b* are filler triples of the form [*type, fill, src*] where *type* is a type name, and *fill* and *src* are edge types.

**Example 2.90** The lexical entry below specifies that any word can function as the first conjunct in a gapping coordination. The gapping frame licenses the generation of a gapping filler, using the current node as the gapping source, and specifies that landing edges for gapping dependents must have edge type gapd, replacement edges must have type repl, and specifies the lexical type and edge types associated with the gapping filler, the internal fillers, and the boundary fillers (which are all simple fillers). The lexical entries for gapping fillers and internal fillers are defined in detail in Example 2.96.

```
type(word)
−> fframe(gapping_coord, this,
        [gapping_filler, gap, gapsrc],
        [internal_filler, fill, src],
        [simple_filler, fill, src], gapd, repl);
```

In the following, we let $n_f$ denote a gapping filler generated by a gapping frame $(s, f, i, b, l, r)$ with filler source $n_s$ in a dependency graph $G$. We first define the different kinds of nodes in gapping constructions.

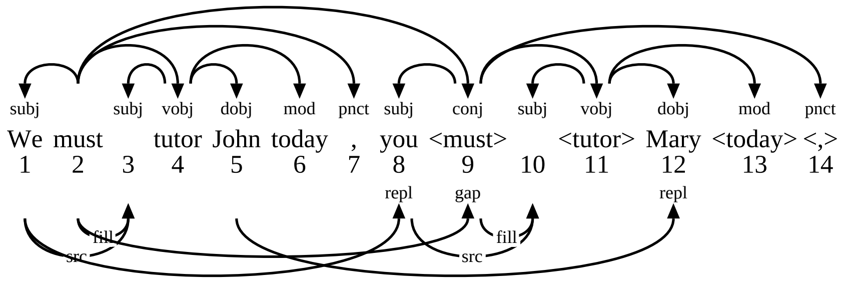


Figure 2.26: A DG analysis of "We must tutor John today, you Mary".

**Definition 2.91** A non-filler node that lands on $n_f$ is called a *gapping dependent* of $n_f$. A node $n$ is called a *subsource* of $n_f$ if $n$ is deeply dominated by $n_s$, $n$ is not contained in the deep yield of any gapping filler with filler source $n_s$, and $G$ contains no *replacement edge* of the form $d' \xleftarrow{r} n$ where $d'$ is a gapping dependent of $n_f$. A subsource $n$ is called a *proper subsource* if neither $n$ itself, nor any of its transitive governors below $n_s$, is a filler whose filler source is a subsource of $n_f$. A proper subsource is called an *internal subsource* if its deep yield contains a gapping dependent of $n_f$. A non-internal proper subsource is called a *boundary subsource* if it is a dependent of either $n_s$ or some internal subsource of $n_s$. A filler $n'$ with filler edge $n' \xleftarrow{i_{\text{fill}}} n_f$ and source edge $n' \xleftarrow{i_{\text{src}}} n$ is called an *internal filler* for $n$ in $n_f$ if $n$ is an internal subsource of $n_f$, and a *boundary filler* for $n$ in $n_f$ if $n$ is a boundary subsource of $n_f$.

**Example 2.92** Figure 2.26 shows (a simplified visualization of) a DG analysis of the gapping coordination "We must tutor John today, you Mary". The table in Figure 2.27 exemplifies how the concepts from Definition 2.91 are used by listing all matching nodes and edges from Figure 2.26.

We can now define the deep source tree and deep gapping tree in a gapping construction, and the important notion of parallelism between them.

**Definition 2.93** The subgraph $T$ of $G$ consisting of $n_s$, its internal subsources and boundary subsources, and all deep edges connecting them, is called the *deep source tree* of $n_f$. The subgraph $T'$ of $G$ consisting of $n_f$, its internal fillers and boundary fillers, and all deep edges connecting them, is called the *deep gapping tree* of $n_f$. $T$ and $T'$ are called *parallel* if $T'$ is the

| Concepts | Matching nodes and edges |
|---|---|
| gapping filler | <must>$_9$ |
| gapping dependent | you$_8$ Mary$_{12}$ |
| subsource | filler$_3$ tutor$_4$ today$_6$ comma$_7$ |
| replacement edge | We$_1$ $\xrightarrow{\text{repl}}$ you$_8$ John$_5$ $\xrightarrow{\text{repl}}$ Mary$_{12}$ |
| proper subsource | tutor$_4$ today$_6$ comma$_7$ |
| internal subsource | tutor$_4$ |
| boundary subsource | today$_6$ comma$_7$ |
| internal filler | <tutor>$_{11}$ |
| boundary filler | <today>$_{13}$ <comma>$_{14}$ |

Figure 2.27: Gapping concepts applied to Figure 2.26.

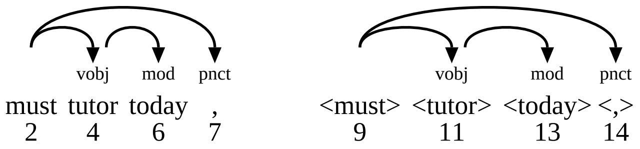


Figure 2.28: The deep source tree (left) and deep gapping tree (right) for the gapping filler <must>$_9$ in Figure 2.26.

result of replacing each node $n$ in $T$ with its corresponding filler node $n'$ in $T'$, ie, if every deep edge $d \xleftarrow{r} g$ in $T$ corresponds to a deep edge $d' \xleftarrow{r} g'$ in $T'$, and vice versa.

**Example 2.94** The deep source tree and the deep gapping tree for Figure 2.26 is shown in Figure 2.28. The two trees are clearly parallel.

Finally, we can define what it means for the fillers in the deep gapping tree to be filler well-formed.

**Definition 2.95** The gapping filler $n_f$ and its internal fillers and boundary fillers are called *filler well-formed* if (a) the filler source of $n_f$ satisfies the type description $s$ evaluated from the filler licensor of $n_f$; (b) $n_f$ is well-formed with respect to the filler triple $f$; (c) the internal fillers of $n_f$ are well-formed with respect to the filler triple $i$; (d) the boundary fillers of $n_f$ are well-formed with respect to the filler triple $b$; (e) the deep gapping tree of $n_f$ is parallel with the deep source tree of $n_f$; (f) each gapping dependent has landing edge $[\xleftarrow{l} n_f]$; and (g) each gapping dependent $n'_d$ of $n_f$ has at most one replacement edge $n'_d \xleftarrow{r} n$, and if this edge exists, $n$ must have the same governor as the subsource $n_d$ for $n'_d$.

In gapping constructions, we have introduced three new kinds of fillers: gapping fillers, internal fillers, and boundary fillers. As boundary fillers, we can use simple fillers (cf. Example 2.86) because they do not allow any complements, adjuncts, or landed nodes, and generate a semantic variable from the filler source — which is what we want for boundary fillers. But for gapping fillers and internal fillers, we need fillers that can take complements and adjuncts, and compute an independent compositional semantic representation, just like ordinary lexemes. Gapping fillers must additionally function as landing sites so that they allow their gapping dependents as landed nodes. In the following example, we will demonstrate one way of implementing gapping fillers and internal fillers in the DG lexicon.

**Example 2.96 (gapping fillers)** Gapping fillers and internal fillers can be viewed as fillers that act as perfect replica of their filler source, apart from two modifications: in addition to the supertypes they inherit from their filler source, they must have a supertype that indicates that they are fillers; and their landing site properties must be restricted so that the only landed nodes licensed by their landing frames are the gapping dependents that land on gapping fillers. The lexical entry for internal fillers is shown below:

```
type(internal_filler)
  -> super(filler, super_from_source, features_from_source,
           no_dynamic_lframes);
```

The filler supertype is used to indicate that internal fillers are fillers. The supertype features_from_source defines a dynamic feature function named ˜features_from_source that copies all feature values from the filler source. The name starts with a tilde (ASCII value 126) so that it comes last in the alphabetical sorting order (cf. Remark 2.26), in order to ensure that the feature values produced by ˜features_from_source are used as defaults that can be overwritten by other dynamic feature functions.

```
type(features_from_source)
    -> setd('~features_from_source', ' ', sub {
            my ($G, $n) = @_;
            my $source = $G->get_node($n, $[\xrightarrow{\text{source}} \text{this}]$);
            return $source ? $G->getd($source, '//') : {};
        });
```

The super_from_source supertype introduced in Example 2.86 ensures that the internal filler also returns the supertypes of its internal source. The supertype no_dynamic_lframes ensures that internal fillers keep their non-dynamic landing frames and ignore all landing frames generated by features_from_source.

```
type(no_dynamic_lframes)
    -> setd(no_dynamic_lframes, 'lframe:', sub {
            my ($G, $n) = @_;
            return $G->getd($n, '/^lframe:/');
        });
```

The lexical entry for gapping fillers is the same as for internal fillers, except that a gapping filler additionally allows any node to land on it, provided the landed node's governor is an internal filler or a gapping filler.

```
type(gapping_filler)
    -> super(filler, super_from_source, features_from_source,
                no_dynamic_lframes)
    -> lframe(gapd => any)
    -> cost(bad_gapd_governor,
            100 * ($[\xrightarrow{\text{deep}} [\xleftarrow{\text{gapd}} \text{this}]]$ - (internal_filler|gapping_filler));
```

In this section, we have shown how secondary dependencies can be encoded by means of fillers that are lexically licensed by a filler licensor. We have introduced simple fillers, which can be used to account for verbal complexes, control constructions, relatives, coordinations with peripheral sharing, and parasitic gaps. We have also introduced gapping fillers, internal fillers, and boundary fillers, which can be used to account for gapping coordinations. We will return to these issues in sections 4.3 and 4.4, where we provide a more detailed analyses of constructions involving simple fillers and gapping fillers.

## 2.6 Summary

In this chapter, we have introduced the most important aspects of the DG formalism, apart from cost functions and probability models. In section 2.1, we defined dependency graphs and their properties. In section 2.2, we described DG's multi-hierarchical inheritance lexicon and its notion of type transformations and dynamic feature functions, and defined a type specification language that can be used to express constraints on DG graphs. In section 2.3, we described DG's notion of complements and adjuncts and how they relate to functor-argument structure, and we explained the difference between complements and adjuncts from a theoretical point of view. In section 2.4, we described how DG controls word order by means of landing sites licensed by landing frames, and how the surface tree is related to the deep tree by means of the deep upwards movement principle; we also briefly sketched how cost functions can be used to control how landing sites order their landed nodes. Finally, in section 2.5, we described how DG accounts for secondary dependencies by means of simple fillers in verbal complexes, relatives, and control constructions, and by means of other specialized fillers in gapping constructions.

We have also defined a notion of well-formedness within DG, which departs from the classical notion of grammaticality in that a well-formed graph can be highly ungrammatical, either because the speaker's utterance is ungrammatical, because it is only a partial analysis of the speaker's utterance, or because it violates grammatical principles that are only expressed by means of cost functions. In the following chapter, we will address how to define cost functions that encode violable constraints with respect to word order, agreement, island constraints, selectional restrictions, etc.

# Chapter 3

# The cost measure: weighted local constraints

We introduce a set of cost operators, which can be used to encode a wide range of weighted syntactic, semantic, and time-dependent linguistic constraints. We describe how costs are related to grammaticality, probability, OT constraints, and neural networks. Finally, we propose a linguistically plausible axiomatization of speaker preferences that entails the existence of a cost measure.

In DG, a grammar is given by a search space that defines the space of all conceivable analyses, and an associated cost measure that quantifies the absolute well-formedness of different analyses. In the model of human communication we have presented in section 1.1, all we know about cost measures is that they are functions from the set $\mathcal{A}$ of analyses into the set $[0, \infty[$ of non-negative real numbers. However, cost measures are only useful if there is an efficient procedure for computing the cost associated with any given analysis. For this reason, we will assume that the cost measure arises as the aggregate sum of a large number of weighted constraints that are associated with particular nodes in the graph. This assumption is not only compatible with the examples of manually written cost functions used as examples in the present chapter, but also compatible with the cost functions in the probabilistic language model presented in section 6.2.

| **Cost operator** | **DTAG notation** | **Definition** |
|---|---|---|
| $\|a\|$ | abs($a$) | Number of objects that match $a$. |
| $\neg$a | not($a$) | 0 if $\|a\| > 0$, 1 if $\|a\| = 0$. |
| $n * a$ | $n * a$ | $n$ multiplied by $\|a\|$. |
| dist($a, b$) | dist($a, b$) | The sum of the distances between all pairs of nodes $(n_1, n_2)$ where $n_1$ matches $a$, and $n_2$ matches $b$ (cf. Definition 2.68). |
| egain($b, r, t_1, t_2, q$) | egain($b, r, t_1, t_2, q$) | The expected unit gain for a resolution time distribution with parameters $(b, r, t_1, t_2, q)$ (cf. Definition 3.7). |
| code($c$) | code($c$) | Cost returned by Perl subroutine c when called as &$c$($G$,$n$). |

Figure 3.1: The cost operators in DG: $a$ and $b$ are type specifications, and $n$ is a non-negative real number.

## 3.1 Cost functions and cost operators

***Summary***. *We describe how cost measures are specified by means of localized weighted constraints that are associated with particular nodes in the graph. The constraints that apply to a particular lexical type are specified in the lexicon, using a small vocabulary of cost operators.*

We will now present a formal language that allows us to specify cost measures in the grammar by means of local weighted constraints. These weighted constraints are called *cost functions*.

**Definition 3.1** A *cost function* is a function $c$ that given a node $n$ and a graph $G$ computes a non-negative cost $c(G, n) \in [0, \infty[$. Cost functions are specified in the lexicon by means of the cost operators shown in Figure 3.1, and by means of the statistical modelling language introduced in chapter 6. The *cost* of a node $n$ in the graph is the sum of all costs induced by cost functions associated with $n$, and the *cost* of a graph $G$ is the sum of the cost of all nodes in $G$.

**Remark 3.2** The set of cost functions associated with a node in the graph can be viewed as a local "penalty code" that specifies what counts as an offense at that node, and how severely it is punished: minor offenses receive a small fine, whereas major offenses receive a large fine. Small fines are used to express weak preferences.

The non-probabilistic cost functions are specified in the DTAG lexicon as described below.

**Pseudocode 3.3** A cost function *costfunc* is added to the type *t* with the method:

```
type(t)−>cost(name, costfunc)
```

using the identifier *name*.[1]

In the following section, we will show how manually written cost functions can be used to encode a wide range of violable linguistic constraints. In section 6.2, we will then present an alternative set of cost functions that is specified indirectly by means of a probabilistic language model.

## 3.2 Examples of cost functions

***Summary.*** *We exemplify how the cost operators can be used to specify a wide variety of hand-written weighted constraints, including constraints on obligatory presence and absence, word order, agreement, island constraints, word distances, selectional restrictions and semantics, and dynamic costs for modelling time-dependent activation levels.*

In the following, we will demonstrate how cost operators can be used to express hand-written cost functions that encode a wide range of syntactic constraints. The hand-written cost functions do not constitute a comprehensive DG grammar of Danish, English, or German, nor do we claim that a good DG grammar must necessarily contain these rules, although they could be used as an inspiration. The examples are merely meant to illustrate how the cost operators can be used, and to show that our cost language is general enough to deal with a wide range of phenomena. The hand-written cost functions are also used as an inspiration in the formulation of a probabilistic language model for DG. For simplicity, we will use the cost 100 to signal that a construction is ungrammatical, and the cost 50 to signal that a construction sounds unnatural.

[1]The method type(*t*)−>cost(*name*, *costfunc*) is defined as an abbreviation for type(*t*)−>set("cost:*name*", *costfunc*).

## Obligatory presence and absence

Complement frames, landing frames, and filler frames license the presence of certain edges, but they do not punish the absence of a licensed complement or filler. Thus, if we want to signal that an edge must be obligatorily present or absent in order to avoid a grammatical error, we must specify this excplicitly by means of a cost function. For example, every word (except for the root word) must obligatorily have a landing site and a governor, so we can punish the absence of these by means of the following two cost functions, defined at the word level.

```
type(word)
    -> cost(missing_gov, 100 $* \neg[\xrightarrow{\text{deep}}\text{this}]$)
    -> cost(missing_lsite, 100 $* \neg[\xrightarrow{\text{surface}}\text{this}]$);
```

The type specification $[\xrightarrow{\text{deep}}\text{this}]$ returns the set of all governors to the current node, so $\neg[\xrightarrow{\text{deep}}\text{this}]$ will return 1 if the current node does not have a governor, and 0 if it has a governor. Likewise, all dependents of a word are optional by default, since no cost is associated with their absence, but can be made obligatory by imposing a cost on their absence.

Landing sites may want to punish the absence of a landed node. For example, in Danish, a finite verb in a non-interrogative clause must always have a landed dependent (such as a subject, an adverbial, or a topicalized phrase) to its left. This can be specified with the following cost function:

```
type(finite_verb)
    -> cost(missing_left_landed, 100 $* \neg([\xleftarrow{\text{surface}}\text{this}]+[<\text{this}])$);
```

It is also possible to punish the presence of a node. For example, an adjective can modify a common noun (as in "Good ideas emerged" and "Cold water is nice"), but in the presence of a determiner (eg, "The cold water is nice"), the adjective cannot attach to the common noun but must modify the determiner (cf. footnote 10 on p. 64). This can be encoded in the lexicon by punishing an adjective if it is placed between two nouns where the second noun is the nominal object of the first (eg, a preceding determiner and a following noun), and it modifies the following rather than the preceding noun. The corresponding cost function is shown below.

type(adjective)
  −> aframe(mod_nattr, mod => noun)
  −> cost(bad_nmod,
    100 * $(\text{noun} + [< \text{this}] + [\xrightarrow{\text{nobj}} [\xrightarrow{\text{mod}} \text{this}] + \text{noun} + [> \text{this}]]))$;

In section 6.2, we will present an even better solution: a probabilistic language model that is sensitive to obligatory presence and absence.

### Word order violations

To control word order, we must be able to specify cost functions that can check the presence and absence of landed nodes, and the relative word order of different landed nodes, both with respect to the landing site and with respect to each other. To this purpose, we can use the cost operators $[< a]$ and $[> a]$, which match all nodes that come before or after a node with type specification $a$, and the cost operators $\text{prev}(a, b)$ and $\text{next}(a, b)$, which match nodes that, within the set of nodes that match $a$, immediately precede (respectively, succeed) any node that matches $b$. For example, we can use the following cost function to state that a preposition should always precede its nominal object:

type(preposition)
  −> cost(bad_order_nobj, 100 $* ([\xleftarrow{\text{nobj}} \text{this}]+[\xleftarrow{\text{surface}} \text{this}]+[< \text{this}]))$;

The same kind of rule can be used to express a weak preference for letting a subject precede its governing verb. Similarly, we can use the following cost function to state that to the right of a verb, a landed indirect object should precede a landed direct object.

type(verb)
  −> cost(bad_order_dobj_iobj, 100 * $([\xleftarrow{\text{dobj}} \text{this}]+[\xleftarrow{\text{surface}} \text{this}]+[> \text{this}]$
    $+[< [\xleftarrow{\text{iobj}} \text{this}]+[\xleftarrow{\text{surface}} \text{this}]]))$;

The probabilistic language model presented in section 6.2 provides an even better way of encoding word order constraints.

### Agreement violations

Agreement can be encoded by means of cost functions that check whether a dependent agrees with its governor. For example, case agreement in English (cf. Hudson 2003) can be encoded by distinguishing beween nominative and accusative pronouns in the type hierarchy, and stipulating that nominative pronouns can only appear as subjects, and accusative pronouns only as non-subjects, as shown in the lexical entry below.

> type(pronoun_nom)
>     −> super(pronoun)
>     −> cost(bad_case, 100 * $\neg[\xrightarrow{\text{subj}}$ this$]$);
> type(pronoun_acc)
>     −> super(pronoun)
>     −> cost(bad_case, 100 * $[\xrightarrow{\text{subj}}$ this$]$);
> type(they:pp) −> super(pronoun_nom);
> type(them:pp) −> super(pronoun_acc);

This rule is obviously a simplification, but a more detailed rule can be specified with the same mechanisms. For example, to stipulate that conjuncts in coordinated subjects can be nominative (as in "Jane and I went to the Zoo today"), we can replace $[\xrightarrow{\text{subj}} \text{this}]$ with the type specification

$$[\xrightarrow{\text{subj}} \text{this}] \mid [\xrightarrow{\text{subj}} [\xrightarrow{\text{conj}} \text{this}]]$$

in pronoun_nom. Similar refinements are required in Danish, where only subjects without a restrictive modifier are allowed to have nominative case (cf. Heltoft and Hansen 2000, p. 25).

Case agreement is highly restricted in Danish and English, but the mechanism sketched above is powerful enough to handle more complicated agreement phenomena as well. For example, German nouns are marked for gender (masculine, feminine, neuter) and must agree with their determiner. In the lexical entry below, the gender of a common noun is indicated by making the noun a subtype of one of the gender types masc, fem, or neut; the determiner induces a cost if the noun's gender is illegal.

> type(der:p1)
>     −> super(pronoun, masc)
>     −> cost(bad_gender, 100 * ($[\xleftarrow{\text{nobj}}$ this$]$ − masc));
> type(mann:n1) −> super(noun, masc);
> type(auto:n1) −> super(noun, neut);

Alternatively, agreement can be encoded by means of an agreement feature, as in the lexical entry below, where the gender of a noun is encoded in the gender feature, and where a determiner induces a cost if it does not have the same gender as its nominal object.

type(pronoun)
  −> cost(bad_gender, 100 * $(\mathrm{val}(\mathrm{gender}, \mathrm{this}) \neq \mathrm{val}(\mathrm{gender}, [\xleftarrow{\mathrm{nobj}} \mathrm{this}])))$;
type(der:p1) −> super(pronoun) −> set(gender, masc);
type(mann:n1) −> super(noun) −> set(gender, masc);
type(auto:n1) −> super(noun) −> set(gender, neut);

The probabilistic language model presented in section 6.2 is capable of modelling agreement as well.

## Island constraints

Island constraints (Ross 1967) place restrictions on the possible extraction paths from a word's governor to its landing site. These restrictions reduce the time complexity of parsing and generation by limiting the number of potential governors a node can have, given its landing site — an issue we will return to in section 7.3. All languages have island constraints, but even though island constraints are generally very similar across languages, there are slight variations from language to language.[2] Figure 3.2 shows a list of island constraints from Borsley (1991, 181–187), and their corresponding formulation in DG.

Island constraints could be encoded as inviolable constraints — an approach taken by many syntactic theories, including GB and HPSG. However, the great variability between island constraints makes it more convenient in DG to treat them as violable constraints that are encoded by means of cost functions that impose a cost whenever they are violated. The lexical entries below shows one way of encoding the island constraints from

[2]For example, the wh-island condition does not always hold in Danish:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| (1) | Denne | forklaring | ved | jeg | ikke | hvorfor | de | godtog. |
| | This | explanation | know | I | not | why | they | accepted. |

*I do not know why they accepted this explanation.*

Similarly, the complex NP constraint is sometimes violated in Danish:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (2) | Denne | mærkelige | forklaring | kender | jeg | ikke | grunden | til | at | de | godtog. |
| | This | strange | explanation | know | I | not | reason-the | for | that | they | accepted. |

*I do not know the reason for (the fact) that they accepted this strange explanation.*

Allwood (1976) discusses a number of violations of the complex NP constraint in Swedish and Norwegian.

| | **Borsley formulation** | **DG formulation** |
|---|---|---|
| **subject condition** | A wh-dependency cannot cross the boundary of a subject. | An extraction path cannot contain a subject edge. |
| **complex NP constraint** | A wh-dependency cannot cross the boundary of a clause and an NP that contains it. | An extraction path cannot contain a verb dominated by a (common) noun.[a] |
| **wh-island condition** | A wh-dependency cannot cross the boundary of a subordinate wh-question. | An extraction path cannot contain a verb that participates in a wh-construction.[b] |
| **adjunct island constraint** | A wh-dependency cannot cross the boundary of an adverbial expression. | An extraction path cannot contain an adjunct edge. |
| **complementizer gap constraint** | A wh-dependency gap cannot appear in subject position in a clause introduced by a complementizer. | A verb governed by a complementizer requires a locally landed subject. |

[a]In our DG analysis, the complementizer "that" is analyzed as a pronoun with an obligatory verbal object, rather than as a complementizer, because of its ability to appear as a subject. For this reason, such complementizer pronouns must be excluded from the DG formulation of the complex NP constraint.

[b]In our DG analysis, subordinate wh-questions are analyzed as relative clauses where the wh-phrase has been relativized (because some Danish subordinate wh-questions include an obligatory relative pronoun, and because the relative clause is usually optional given an appropriate context). Since relative clauses are analyzed as adjuncts, the wh-island condition is a consequence of the adjunct island constraint.

Figure 3.2: A set of island constraints from Borsley (1991, 181–187), and their corresponding formulation in DG.

Figure 3.2 in the DTAG lexicon. The subject condition can be encoded by letting a verb punish any extraction out of one of its subject edges, and the adjunct island constraint can similarly be encoded by letting a word punish any extraction out of one of its adjunct edges.

type(verb)
    −> cost(subject_condition, 100 * extract($\left[\xleftarrow{\text{subj}}\right.$ this$\left.\right]$));
type(word)
    −> cost(adjunct_island_constraint, 100 * extract($\left[\xleftarrow{\text{adjunct}}\right.$ this$\left.\right]$));

In our DG analysis, the wh-island condition is a consequence of the adjunct island constraint and therefore does not need to be encoded explicitly. The complementizer gap constraint can be implemented by letting a finite verb punish any non-locally landed subject, provided the verb functions as the verbal object of a complementizer word like "that", "whether", "if", or "for" (following Hudson (1997), we do not recognize complementizers as an independent word class).

> type(verb) −> cost(complementizer_gap_constraint,
>         100 * ([$\xleftarrow{\text{subj}}$ this + finite + [$\xleftarrow{\text{vobj}}$ that| . . . |for]] − [$\xleftarrow{\text{land}}$ this]));

With respect to the complex NP constraint, Hudson (2003, "Complex NP-island") argues that all complex NPs in English acting as islands are noun phrases containing either a relative clause or an apposition. Since Hudson analyzes these clauses as adjuncts, he sees the complex NP constraint as a consequence of the adjunct island constraint. So if one believes his analysis, there is no reason to encode the complex NP constraint in our grammar. As additional evidence for his analysis, it is easy to find natural-sounding examples of extractions from complex NPs where the clause is not an adjunct (as exemplified in footnote 2 on p. 97), which suggests that such complex NPs do not act as islands, at least in Danish.

**Remark 3.4** Although we believe that complex NPs that contain a VP are non-islands if the VP is a non-adjunct, and that the complex NP constraint is therefore overly restrictive, it is an interesting challenge to try to encode the complex NP constraint exactly as it is formulated in Figure 3.2. The constraint differs from other island constraints by being essentially non-local: other island constraints can be checked by looking at all subpaths of length 1 in the extraction path, but the complex NP constraint requires us to examine whether a common noun dominates a finite verb within this path, ie, it involves looking at a path of potentially unbounded length. Since extract$(a)$ is a local operator, we cannot express the complex NP constraint with extract$(a)$ alone. So to encode this constraint, we must extend our type specification language with a new *dominance operator*, dominated$(a, b)$, which matches any node that is dominated by a node matching $b$ via a path consisting of edges whose edge types match $a$. With this extension, we can make the following encoding:

> type(noun_common) −> cost(complex_NP_constraint, extract(this)
>         + extract(verb + finite + dominated(deep, this)));

From a computational point of view, the dominated$(a, b)$ operator can be implemented efficiently as a *node filter* (ie, we do not allow it to generate all nodes dominated by a given node, but we do allow it to act as a filter that can check whether a given node is dominated by another given node). However, since the only reason we need this operator is an overly restrictive formulation of the complex NP constraint, and since we are not aware of any other phenomena where the dominated$(a, b)$ operator is needed, we prefer to exclude it from our type specification language.

### Word distances

Word distance plays an important role in natural language. For example, in the Danish Dependency Treebank (Kromann et al. 2003), 44% of all dependents are immediately preceded by their governor, and 88% are fewer than 5 words apart from their governor. It is therefore reasonable to assume that human grammars exhibit a preference for minimizing the distance between a word and its governor and landing site. Evidence in psycholinguistics has shown that, anything else being equal, humans have a preference for low attachment in PP-attachment ambiguities (cf. Gernsbacher 1994). This is exactly the behaviour one would expect from a grammar with a preference for minimizing word distance.

Because the preference for minimizing word distance is only a preference, it does not rule out the large distances observed in long-distance dependencies such as topicalizations and relatives. But it ensures that if there are two potential governors, the closest one will be preferred unless there are stronger syntactic and semantic reasons for preferring the more distant governor. In DG, the word distance preference can be encoded with a cost function that induces a small cost (say, 0.5) for every word separating a node from its governor or landing site, as in the lexical entry below:

```
type(word)
    -> cost(dist_gov, 0.5 * dist(this, [$\xrightarrow{\text{deep}}$ this]))
    -> cost(dist_lsite, 0.5 * dist(this, [$\xrightarrow{\text{surface}}$ this]));
```

### Selectional restrictions and semantics

Selectional restrictions can be encoded in our lexicon if we assume the lexicon contains an ontological hiearchy (cf. Cimiano and Handschuh 2003), and that each lexeme has an associated ontotype feature which contains the

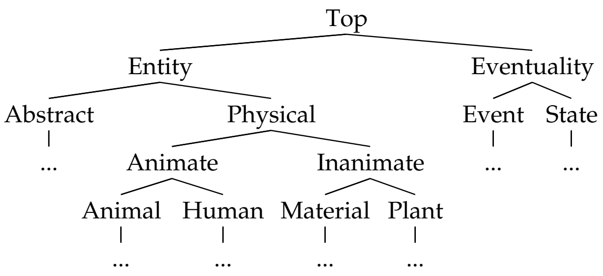


Figure 3.3: An example of a simple ontological hierarchy.

ontological type associated with the phrase headed by the lexeme. For content words like common nouns, the ontotype can be specified in the lexicon, but function words like determiners must compute their ontological type dynamically from their complements, or use a default ontological type in the absence of a complement.

A simple ontological hierarchy is shown in Figure 3.3. Within this hierarchy, the lexeme poet:n could be assigned ontological type Human, the lexeme poem:n could be assigned ontological type Info*Material, and the lexeme write:n could be assigned ontological type Event. The corresponding lexical entries are shown below.

```
type(poet:n) −> set(ontotype, Human);
type(poem:n) −> set(ontotype, Info*Material);
type(write:v) −> set(ontotype, Event);
```

A function word like the determiner "the" copies its ontotype from its nominal object, or uses the default ontotype Entity in the absence of a nominal object. This can be encoded with a dynamic feature function, as shown in the lexical entry below.

```
type(the:pd) −> setd(ontotype, ontotype, sub {
        my ($G, $n);
        my $nobj = $G−>get_node($n, [$\xleftarrow{\text{nobj}}$ this]);
        return $nobj
            ? $G−>getd($nobj, '/^ontotype$/')
            : {ontotype => Entity}; });
```

Using the ontotype feature, lexemes can use cost functions to express preferences for the ontological type of their complements and adjunct governors.

For example, the lexical entry below illustrates how the verb write:v can express a preference for its subject to have ontotype Human, and for its direct object to have ontotype Info.

type(write:v)
  $->$ cost(ontotype_subj, 20 * (val(ontotype, $[\xleftarrow{\text{subj}}$ this$]$) $-$ isa(Human)))
  $->$ cost(ontotype_dobj, 20 * (val(ontotype, $[\xleftarrow{\text{dobj}}$ this$]$) $-$ isa(Info)));

The ontological type system sketched above can be viewed as a highly reduced formal semantics. However, the dynamic feature functions used for implementing the ontotype feature could equally well be used to implement a more sophisticated semantic theory, like DRT (Kamp and Reyle 1993). Given such an implementation of semantics, possibly coupled with an inference system, very general preferences on semantics and pragmatics can be expressed as cost functions and added to the other costs associated with a lexeme. This points to one of the greatest attractions of cost functions: the ability to seamlessly combine preferences from different components of a human language model with each other, in a localized manner so that analysis and generation algorithms can quickly zoom in on the problematic regions in the analysis.

**Dynamic costs**

So far, we have only considered *static costs*, ie, costs that do not change over time. However, it is reasonable to assume that human grammars employ *dynamic costs*, ie, costs that change over time. For example, a cost function in the lexicon may specify that a word should punish the absence of a governor, landing site, or obligatory dependent. However, if there is a low probability that the missing node has appeared in the speech signal at a given time, there is no reason for the parser to trigger an expensive reanalysis operation. But after waiting for some time, there is a good chance that the missing node has appeared, and the parser should then trigger a reanalysis operation that attempts to resolve the violation. On the other hand, if the parser has just attempted an unsuccessful repair operation, it makes sense to wait a while before it makes a new attempt. And after a cost function has indicated a violation for a long time, and the parser has repeatedly tried to find a better analysis, but failed, it makes sense to assume that the input contains a speech error which is impossible to fix. Rather than pondering about the error forever, the parser should stop trying to repair the violation by ignoring the resulting cost. This suggests that cost functions should fall below the threshold for triggering a repair operation after a sufficiently long delay.

These insights can be formalized by assuming that the grammar contains statistical information about the time $T$ when a cost function is resolved, ie, when it falls permanently below a predefined threshold.

**Definition 3.5** We will assume that each node in a graph $G$ has an associated *time span* $[t_1, t_2]$ that encodes its duration in the speech signal. The *temporal subgraph* $G_t$ of $G$ is defined as the subgraph consisting of all nodes in $G$ whose time spans precede $t$, and all edges in $G$ connecting these nodes.

**Definition 3.6** Let $G$ be a graph, let $n$ be a node in $G$ with time span $[t_1, t_2]$, and let $c$ be a cost function associated with $n$. We will say that $c$ is *resolvable* with *resolution time* $t \in [0, \infty]$ with respect to the *threshold* $\tau$ if $t$ is the time measured from $t_2$ where the value of $c$ falls permanently below $\tau$, ie:

$$t = \min\{s - t_2 | c \leq \tau \text{ at node } n \text{ in } G_s\}$$

If $c$ never falls permanently below $\tau$, we say that $c$ is *non-resolvable* with resolution time $t = \infty$.

The resolution time for a cost function varies from graph to graph. Sometimes, the violation will be resolved right away; at other times, there will be a long delay before the violation is resolved; and sometimes, the violation is never resolved. The resolution time can therefore be viewed as a random variable $T$ with underlying distribution function $F\colon [0, \infty] \to [0, 1]$ satisfying $F(t) = P(T \leq t)$ where $F(0) = b$, $F(\infty) = 1$, and $\lim_{t \to \infty} F(t) = r$; $b$ is the probability that a cost function is resolvable right away, and $r$ is the probability that a cost function is resolvable at all. We will decompose $F$ into a finite and an infinite component by defining:

$$F(t) = \begin{cases} b + (r - b)F_0(t) & \text{if } t \text{ is finite} \\ 1 & \text{if } t = \infty \end{cases}$$

where $F_0(t) = P(T \leq t \mid 0 < T < \infty)$ is the probability that $T \leq t$ given that $T$ is a positive real number.

The parameters $r$ and $b$, and the distribution function $F_0$ for a cost function $c$ can be estimated from the actual resolution times observed in a tree-

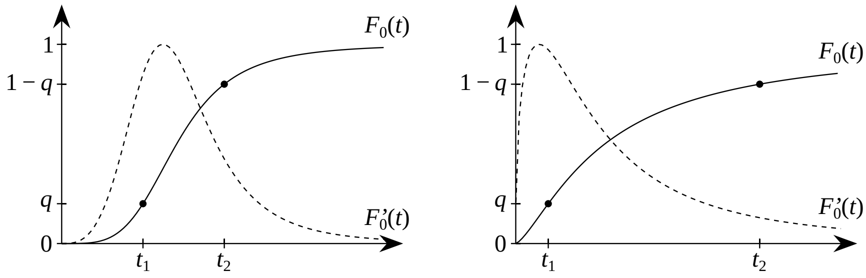


Figure 3.4: Two Champernowne distribution functions $F_0(t)$ (solid lines), and their associated normalized density functions $F_0'(t)$ (dashed lines).

bank.[3] For our purpose, it is not important that the shape of the estimated distribution fits the true distribution perfectly, as long as we get the $q$ and $1-q$ quantiles approximately right. For this reason, we will use a parametric estimator based on the flexible, heavy-tailed,[4] and computationally simple *Champernowne distribution* (cf. Brown 1937; Champernowne 1952), given by the distribution function:

$$F_0(t) = \frac{t^\alpha}{t^\alpha + M^\alpha}$$

where $M > 0$ is the median and $\alpha > 0$ is a tail parameter. If $F_0(t_1) = q$ and $F_0(t_2) = 1-q$, the values of $M$ and $\alpha$ can be computed as:

$$M = \sqrt{t_1 t_2}, \qquad \alpha = 2\frac{\ln q - \ln(1-q)}{\ln t_1 - \ln t_2}$$

Two examples of Champernowne distributions are shown in Figure 3.4. The distribution function $F_{M,\alpha,b,r}$ on $[0,\infty]$ defined by

$$F_{M,\alpha,b,r}(t) = \begin{cases} b + (r-b)t^\alpha/(t^\alpha + M^\alpha) & \text{if } t \in [0,\infty[ \\ 1 & \text{if } t = \infty \end{cases}$$

with $M > 0$, $\alpha > 0$, $b \in [0,1]$, and $r \in [b,1]$ is called the *projective Champernowne distribution* with parameters $M$, $\alpha$, $b$, and $r$.

[3]The resolution time can either be measured in seconds in spoken language, or in characters or tokens in written language.

[4]The Champernowne distribution converges towards the Pareto distribution $G(t) = 1 - (M/t)^\alpha$ (defined on $[M,\infty[$) in the tail, ie, $F_0(t) \to G(t)$ as $t \to \infty$ (cf. Buch-Larsen et al. 2005).

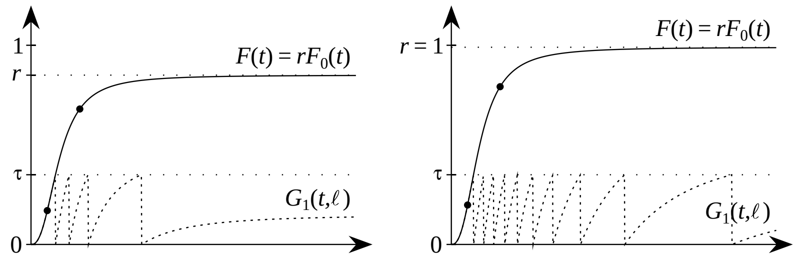


Figure 3.5: The projective Champernowne distribution $F(t)$ with $b = 0$ and the expected gain $G_1(t, \ell)$, with a finite number of reanalyses when $r < 1$ (left), and an infinite number of reanalyses when $r = 1$ (right).

Our intuitions about the dynamic properties of cost functions can now be formalized on a solid probabilistic basis, by replacing our static cost $c$ with the *expected gain* $G_c(t, \ell)$, defined as the gain in cost that a repair operation at time $t$ is likely to produce, given that the last repair operation was attempted at time $\ell(t)$. The quantity $G_c(t, \ell)$ can be calculated as the probability $P(T \in [\ell(t), t] | T \notin [0, \ell(t)])$ times the cost $|c|$, ie, as:

$$G_c(t, \ell) = \frac{P(T \in [\ell(t), t])}{P(T \notin [0, \ell(t)])} |c| = \frac{F(t) - F(\ell(t))}{1 - F(\ell(t))} |c|$$

We will assume the parser makes a reanalysis whenever $G_c(t, \ell)$ reaches a predefined threshold $\tau$. Since a reanalysis at time $t$ results in setting $\ell(t') = t$ for all $t'$ until the next reanalysis, the graph for $G_c(t, \ell)$ will resemble the saw-tooth graph shown in Figure 3.5. The graph shows how the expected gain builds up from 0 until it reaches the threshold $\tau$, thereby triggering the first reanalysis attempt. If the reanalysis attempt is successful, the cost function (and hence the expected gain) will drop permanently below $\tau$. If the reanalysis attempt is unsuccessful, the expected gain will build up from 0 again, until it either converges towards a value below $\tau$, or reaches the threshold $\tau$ again, thereby triggering a new reanalysis attempt.[5]

[5]When a cost function $c$ is non-resolvable (ie, $\tau/|c| < 1$ at all times), three things can happen. (a) If $\tau/|c| \geq r$, the expected gain will never reach the threshold $\tau$, and no reanalysis will ever be attempted. (b) If $\tau/|c| < r$ and $r = 1$, the reanalysis attempts will continue indefinitely at exponentially growing times; an approximation shows that if $t_n$ is the time for the $n$th reanalysis attempt, then $t_{n+1} \simeq t_n(1 - \tau/|c|)^{-1/\alpha}$ for sufficiently large $n$, so that $t_{n+k} \simeq t_n(1 - \tau/|c|)^{-k/\alpha}$. (c) If $\tau/|c| < r$ and $r < 1$, the expected gain will

Thus, the expected gain results in a dynamic cost function that satisfies our earlier list of desiderata: it is initially a fixed value $b$, and grows with the probability that the resolution time falls within the elapsed time interval; after an unsuccessful reanalysis, it falls back to 0; and if there is a non-zero probability that the resolution time is infinite, the expected gain will eventually fall permanently below our predefined threshold.

On the basis of the expected gain, we can define the dynamic cost operator egain.

**Definition 3.7** The dynamic cost operator $\mathsf{egain}(b, r, t_1, t_2, q)$ is called the *expected unit gain* for a projective Champernowne distribution with parameters $(b, r, t_1, t_2, q)$; when evaluated at time $t$ at a node that was created at time $t_c$ and last reanalysed at time $t_r$, it returns the weight:

$$\frac{F_{M,\alpha,b,r}(t - t_c) - F_{M,\alpha,b,r}(t_r - t_c)}{1 - F_{M,\alpha,b,r}(t_r - t_c)}$$

where $M = \sqrt{t_1 t_2}$ and $\alpha = 2\frac{\ln q - \ln(1-q)}{\ln t_1 - \ln t_2}$, and where

$$F_{M,\alpha,b,r}(t) = \begin{cases} b + (r-b)t^\alpha/(t^\alpha + M^\alpha) & \text{if } t \in [0, \infty[ \\ 1 & \text{if } t = \infty \end{cases}$$

The following example shows how the egain operator can be used to encode the dynamic cost of a missing governor.

**Example 3.8** The cost function $100 * \neg[\xrightarrow{\mathsf{deep}} \mathsf{this}]$ checks whether the associated node $n$ has a governor, and returns the cost 100 if the governor is absent. The cost function can be made dynamic by multiplying the cost with the egain operator. We will assume that 5% of all words are resolved right away, that 95% of all words have finite resolution time, and that within the set of all words with resolution time other than 0 or $\infty$, 10% of all words receive their governor within 0.2 seconds, and 90% of all words receive their governor within 2 seconds. This can be specified in the DTAG lexicon with the lexical entry below.

eventually converge towards a value below $\tau$, and no further reanalysis attempts will be made; a calculation shows that the last reanalysis attempt cannot appear earlier than the time $M\sqrt[\alpha]{(r - \tau/|c|)/(1-r)}$.

type(word)
  $\rightarrow$ cost(missing_gov, egain$(0.05, 0.95, 0.2, 2, 0.1)$ * 100 * $\neg[\xrightarrow{\text{deep}}$ this$]$);

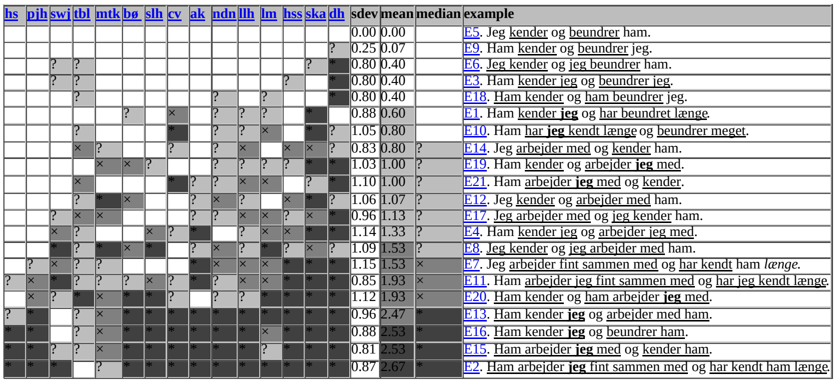
Time-dependent behaviour is seen in a number of psycholinguistically inspired frameworks, as described below. These frameworks have convinced us about the need to build time-dependency into the grammar, although the time-dependency in DG is achieved by different means than in these frameworks.

**Remark 3.9 (time-dependency in competitive inhibition)** In the psycholinguistic parsing model proposed by Vosse and Kempen (2000, 12-13), nodes in the syntax graph have an associated *activation value* which decays exponentially over time. Nodes with a high activation value can be viewed as volatile nodes where dependency links are likely to change, whereas nodes with a low activation value have a more stable dependency structure. Thus, to emphasize the parallels with the model of Vosse and Kempen, we could interpret the dynamic egain operator as an activation value rather than a conditional probability.

**Remark 3.10 (time-dependency in dynamic dependency parsing)** Our dynamic egain cost operator can also be viewed as an alternative to the dynamic weighted constraint dependency grammar proposed by Daum (2004). In Daum's framework, dependency parsing is described as a dynamic constraint optimization problem where the parser is guided towards the "right" analysis by means of incremental changes to constraints and their weights during parsing, and by means of hypothesized dependency edges for unseen words. In our framework, the grammar remains constant, dependency edges to unseen words are not allowed, and the parser determines the most promising repair sites within the analysis by means of the time-dependent conditional probability weights associated with different cost functions. Thus, while the underlying motivation is the same, the means are very different.

## 3.3 Grammaticality, OT, probabilities, and neural networks

***Summary**. We relate costs to other linguistic concepts. First, we argue that ungrammaticality is a local property that corresponds to the violation of a local cost function. We then describe how OT constraints can be interpreted as non-localized polynomial-valued cost*



| hs | pjh | swj | tbl | mtk | bø | slh | cv | ak | ndn | llh | lm | hss | ska | dh | sdev | mean | median | example |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | 0.00 | 0.00 | | E5. Jeg kender og beundrer ham. |
| | | | | | | | | | | | | | | ? | 0.25 | 0.07 | | E9. Ham kender og beundrer jeg. |
| | | ? | ? | | | | | | | | | | ? | * | 0.80 | 0.40 | | E6. Jeg kender og jeg beundrer ham. |
| | | ? | ? | | | | | | | | | ? | | * | 0.80 | 0.40 | | E3. Ham kender jeg og beundrer jeg. |
| | | | ? | | | | | | ? | | ? | | | * | 0.80 | 0.40 | | E18. Ham kender og ham beundrer jeg. |
| | | | | | ? | | × | | ? | ? | ? | | * | | 0.88 | 0.60 | | E1. Ham kender **jeg** og har beundret længe. |
| | | | ? | | | | * | | ? | ? | × | | * | ? | 1.05 | 0.80 | | E10. Ham har **jeg** kendt længe og beundrer meget. |
| | | | × | ? | | | ? | | ? | × | | × | × | ? | 0.83 | 0.80 | ? | E14. Jeg arbejder med og kender ham. |
| | | | | × | × | ? | | | ? | ? | ? | ? | * | * | 1.03 | 1.00 | ? | E19. Ham kender og arbejder **jeg** med. |
| | | | × | | | | * | ? | ? | × | × | | ? | * | 1.10 | 1.00 | ? | E21. Ham arbejder **jeg** med og kender. |
| | | | ? | * | × | | | ? | × | ? | | × | * | ? | 1.06 | 1.07 | ? | E12. Jeg kender og arbejder med ham. |
| | | ? | × | × | | | | ? | ? | × | × | ? | × | * | 0.96 | 1.13 | ? | E17. Jeg arbejder med og jeg kender ham. |
| | | × | ? | | | × | ? | * | | ? | × | × | * | * | 1.14 | 1.33 | ? | E4. Ham kender jeg og arbejder jeg med. |
| | | * | ? | * | × | * | | ? | × | ? | * | ? | × | ? | 1.09 | 1.53 | ? | E8. Jeg kender og jeg arbejder med ham. |
| | ? | × | ? | ? | | | | * | × | × | × | * | * | * | 1.15 | 1.53 | × | E7. Jeg arbejder fint sammen med og har kendt ham *længe*. |
| ? | × | * | ? | ? | ? | × | ? | * | ? | × | × | * | * | * | 0.85 | 1.93 | × | E11. Ham arbejder jeg fint sammen med og har jeg kendt længe. |
| | × | ? | * | × | * | * | ? | | ? | ? | * | * | * | * | 1.12 | 1.93 | × | E20. Ham kender og ham arbejder **jeg** med. |
| ? | * | | ? | × | * | * | * | * | * | * | * | * | * | * | 0.96 | 2.47 | * | E13. Ham kender **jeg** og arbejder med ham. |
| * | * | | ? | × | * | * | * | * | * | * | × | * | * | * | 0.88 | 2.53 | * | E16. Ham kender **jeg** og beundrer ham. |
| * | * | ? | ? | × | * | * | * | * | * | * | ? | * | * | * | 0.81 | 2.53 | * | E15. Ham arbejder **jeg** med og kender ham. |
| * | * | * | | ? | * | * | * | * | * | * | * | * | * | * | 0.87 | 2.67 | * | E2. Ham arbejder **jeg** fint sammen med og har kendt ham længe. |

Figure 3.6: The ratings from a survey of elliptic coordinations in Danish (Kromann 2002a) illustrate the variation in speaker intuitions.

*functions, how cost functions are related to probabilities, and how the parallel distributed computations of cost functions and dynamic types are related to neural networks.*

So far, we have viewed cost measures as an efficient way of encoding a speaker's preference ordering on the set of all conceivable analyses. However, cost measures are related to several other linguistic concepts, as described in the following.

### Grammaticality and acceptability

Speakers of a language have intuitions about how well-formed different utterances are within their language. If they think that an utterance is "wrong," "unnatural," or "awkward," they are usually able to pinpoint the problem and suggest a better alternative. This does not mean that different speakers always have the same intuitions, or that a single speaker's intuitions are always very clear-cut: on the contrary, there can be great variation between speakers, and informants in linguistic surveys often report that they are very uncertain about their judgments of difficult border-line cases. The variation between speakers is illustrated by Figure 3.6, which shows how 15 speakers rated the grammaticality of 21 sentences with elliptic coordinations in Danish, using the linguistic survey tool Linguist-GRID.org (Kromann 2002b); the noisy response pattern is quite typical for linguistic surveys of border-line phenomena.

All syntactic theories must model speaker intuitions in some way or another. As a deliberate simplification, most syntactic theories — including GB, HPSG, LFG, and TAG — assume that speakers classify all utterances as being either “grammatical” or “ungrammatical,” with no degrees of grammaticality in between. However, linguists have recently begun to explore more fine-grained notions of grammaticality, using either OT models with discrete degrees of grammaticality that are assumed to correspond to graded linguistic constraints (Keller 2000; Sorace and Keller 2005), or probabilistic models with continuous degrees of grammaticality that are assumed to arise from the probabilities assigned to different parts of a linguistic analysis (Manning 2003).

Like Manning, we believe that speaker intuitions about grammaticality and acceptability are determined by the probabilities (or costs) that their grammars assign to different parts of the speaker's analyses of the input. More specifically, we believe that if the cost of a cost function in a particular analysis is significantly larger than a certain threshold, then the speaker perceives the node and its relevant neighbourhood to be ill-formed with respect to the linguistic constraint expressed by the cost function; if the cost is significantly below the threshold, the speaker perceives the local structure to be well-formed; and if the cost is near the threshold, the speaker perceives the local structure to be a borderline case with an intermediate degree of well-formedness.[6] These intuitions can be formalized as follows.

**Definition 3.11** Let $n$ be a node in a graph $G$. We say that $G$ is *ungrammatical* (*unacceptable*, *ill-formed*) at $n$ with respect to the cost function $c$ and the threshold $\tau$ if $c(G,n) \geq \tau$, and *grammatical* (*acceptable*, *well-formed*) at $n$ with respect to $c$ and $\tau$ if $c(G,n) < \tau$.

Obvously, this definition can be generalized to $n$ levels of grammaticality by using $n-1$ thresholds. To make the model realistic, we should also assume a certain level of random noise in the judgment of each speaker

[6]It is conceivable that different cost functions have different thresholds so that speakers can encode in their grammars that certain phenomena are grammatical even though they are very rare (eg, idiomatic phrases with non-standard word order), and that other phenomena are ungrammatical even though they are very frequent (eg, common grammatical errors that the speaker has been made aware of in school). However, for simplicity, we will ignore this aspect and assume that all cost functions have the same threshold.

(since different speakers have different cost measures, we do not necessarily assume that the speakers have the same "true" judgment).

It is important to note that we have defined grammaticality as a local rather than global property. An analysis does not become ungrammatical just because it has an aggregate cost that exceeds $\tau$ (or, equivalently, a probability that falls below a certain threshold): it must have a cost function that exceeds $\tau$ at a particular node in order to be ungrammatical. For the speaker, this locality is very important because the information about local ill-formedness can be used to guide repair operations towards the problematic parts of the analysis during parsing and generation.

## Rankings in Optimality Theory

While there are important differences between DG and Optimality Theory (Prince and Smolensky 1993; Tesar 1995; Kager 1999), there are also some interesting similarities. In particular, the rankings used in OT can be viewed as a special case of cost functions, if costs are allowed to be polynomials rather than real numbers. The correspondence between OT rankings and polynomial costs can be defined as follows.

**Definition 3.12** Let $\mathcal{R}$ denote the set of ranked constraints in an OT grammar. Let $r$ denote the *rank function* from $\mathcal{R}$ into the non-negative integers, defined by assigning 0 to the lowest ranked constraints in $\mathcal{R}$, 1 to the next-lowest ranked constraints, etc. Let $V_c(G)$ denote the number of violations of the constraint $c$ in the analysis $G$. Then the *cost polynomial* for $G$ with respect to $c$ is defined as $C_c(G) = V_c(G)X^{r(c)}$ where $X$ is a polynomial variable. The *total cost polynomial* $C(G)$ associated with $G$ is defined as the sum of all the cost polynomials for individual constraints in $\mathcal{R}$, ie, as:

$$C(G) = \sum_{c \in \mathcal{R}} C_c(G) = \sum_{c \in \mathcal{R}} V_c(G)X^{r(c)}$$

With this definition, the coefficient of $X^r$ in the polynomial $C(G)$ encodes the total number of violations of rank $r$. This means that OT's candidate order reduces to the standard order on polynomials (ie, $f \geq g$ if the leading coefficient of $f - g$ is non-negative). Consequently, OT constraints can be viewed as cost functions with polynomial-valued costs. By setting the polynomial variable $X$ to a positive number, the polynomial cost can be reduced to a real-valued cost.

OT's use of polynomial-valued costs means that a single violation of a high-ranked constraint always outweighs an unlimited number of violations of lower-ranked constraints. For example, an OT grammar where word order errors have lower rank than agreement errors stipulates that introducing a thousand word order errors into an analysis is better than introducing a single agreement error into the same analysis. This somewhat counter-intuitive consequence of polynomial costs suggests that OT might be better off by using real-valued costs.

Another difference between OT and DG is that costs in DG are localized — ie, associated with specific nodes in the graph — whereas the global OT cost does not record where the grammatical violations are located. As a consequence, the OT cost cannot be used to guide repair operations during parsing and generation — a crucial property in our proposed parser.[7]

Finally, OT and DG differ in how they interpret costs with respect to grammaticality: OT assumes that an analysis is ungrammatical if it does not have minimal cost within its set of competitors (whatever they are, which is not always clearly defined in OT), whereas DG assumes that ungrammaticality is a property that can be computed directly from the local costs, without any consideration for competing analyses. Thus, while there are interesting similarities between the two theories, and OT has inspired the development of DG, the two theories differ in important respects.

## Probabilities

Probabilities in probabilistic language models can be viewed as special instances of costs. For example, let $G$ be a probabilistic context-free dependency grammar with production rules $\pi_1, \ldots, \pi_n$ where $\pi_i$ has the form $p_i\colon H_i \rightarrow L_i\ h_i\ R_i$ where $p_i$ is a probability, $H_i$ is a phrasal category, $h_i$ is a word, and $L_i, R_i$ are sequences of phrasal categories. Let $k_i(a)$ denote the number of times $\pi_i$ occurs in the analysis $a$. Then $a$ has probability:

$$P(a) = \prod_{i=1}^{n} p_i^{k_i(a)}.$$

Under a minus-log transform, we therefore have:

$$C(a) = -\log P(a) = -\sum_{i=1}^{n} k_i(a) \log(p_i),$$

[7]Creating a localized version of OT would probably not be too difficult.

ie, the multiplicative probability $P(a)$ translates into the additive cost $C(a)$. We can create a DG grammar that corresponds to $G$ as follows: Let the lexical type associated with a word $w$ license all complement frames $L_i + R_i$ where $w$ matches $h_i$. Let $w$ license all its complements (and nothing else) as landed nodes. Let $w$ generate the cost $-\log(p_i)$ if there exists $i$ such that $w$ matches $h_i$, the left landed complements of $w$ match $L_i$, and the right landed complements of $w$ match $R_i$ (in order), and let $w$ generate the cost $\infty$ (the probability 0) if no such $i$ exists.

More generally, dependency-based probabilistic language models can be encoded as DG grammars by interpreting costs as minus-log probabilities. In section 6.2, we will return to the question of how more advanced probabilistic language models can be specified in DG.

### Neural networks

The neurons in the brain act as billions of small processing units that carry out their computations in parallel, ie, independently while interacting with clusters of other neurons (cf. Kandel et al. 1991). Because of this architecture, computations in the brain are massively parallel and highly localized, and neural networks seem to provide a good model of the most important aspects of how the brain works. Little is known about the details of how language is stored and processed in the brain, but a number of sophisticated psycholinguistic models of human parsing have been based on neural networks, including Stevenson (1994), Vosse and Kempen (2000), and Stevenson and Smolensky (2005).

In DG, we try to capture the highly localized, parallel nature of the computations in the brain by viewing each node and cost function as an independent processing unit that is responsible for interacting with neighbouring nodes in the graph. Each node checks its associated local constraints, computes the local compositional semantics and phonological content, and computes an aggregate cost on the basis of the costs returned by the individual cost functions. Each cost function computes a cost that measures the ill-formedness of the local graph structure with respect to a particular phonological, morphological, syntactic, semantic, or pragmatic constraint. That is, DG essentially models language processing by means of a network of processing units organized in a graph structure.

This is not exactly the same thing as a neural network, but the emphasis on parallel, localized computation suggests that the DG model may not be incompatible with a neural model of human language processing. In particular, since general neural networks and Turing machines have the same computational power, it is conceivable that DG can be viewed as a high-level approximation to a neural network where nodes and cost functions correspond to clusters of neurons.

## 3.4 An axiomatization of speaker preferences

***Summary.*** *In section 1.1, we argued that a speaker imposes a complete ordering on the space of all possible analyses, and claimed that it was reasonable to assume that this ordering could be expressed by means of a cost measure. In this section, we propose a linguistically plausible model of speaker preferences and show that it satisfies the axioms of utility theory, so that the von Neumann-Morgenstern theorem forces us to conclude that in any linguistic model that is compatible with our axiomatization, speaker preferences can always be expressed by means of a cost measure.*

In this section, we will present two axiomatizations of speaker preferences that entail the existence of a cost measure: a *probabilistic model*, and a more linguistically appealing *corpus model*. Since our existence proofs rely heavily on utility theory, we introduce the main notions of utility theory.

Utility theory models how rational agents express preferences between different outcomes.[8] We use the notation $a \succ b$ if the agent prefers $a$ to $b$, $a \sim b$ if the agent is indifferent between $a$ and $b$, and $a \succsim b$ if the agent prefers $a$ to $b$ or is indifferent between them. In order to state the conditions under which an agent's preference ordering can be expressed by a measurable utility, we must define the notions of complete ordering, mixture set, and measurable utility (cf. Herstein and Milnor 1953).

**Definition 3.13** A *complete ordering* (or *total ordering*) defined on a set $\mathcal{S}$ is a binary relation $\succsim$ on $\mathcal{S} \times \mathcal{S}$ such that (i) for any $a, b \in \mathcal{S}$, we have $a \succsim b$ or $b \succsim a$, and (ii) for all $a, b, c \in \mathcal{S}$, if $a \succsim b$ and $b \succsim c$, then $a \succsim c$. If $a \succsim b$ and $b \succsim a$, we write $a \sim b$ and say that $a$ and $b$ are *indifferent*. If $a \succsim b$ and $a$ and $b$ are not indifferent, we say that $a$ is *preferred over* $b$ and write $a \succ b$.

**Definition 3.14** Let $\mathcal{S}$ be a set equipped with a function that given any $a, b \in \mathcal{S}$ and $\lambda \in [0, 1]$ returns an element in $\mathcal{S}$, which we denote by $\lambda a +$

[8]For an informal overview of utility theory, see Russell and Norvig (1995, 473–475).

$(1-\lambda)b$ and call a *mixture* of $a$ and $b$. We say that $\mathcal{S}$ is a *mixture set* if $\mathcal{S}$ satisfies the following three conditions for all $a, b \in \mathcal{S}$ and $\lambda, \mu \in [0,1]$:

(i) $1a + 0b = a$
(ii) $\lambda a + (1-\lambda)b = (1-\lambda)b + \lambda a$
(iii) $\lambda(\mu a + (1-\mu)b) + (1-\lambda)b = \lambda\mu a + (1-\lambda\mu)b$

**Definition 3.15** Let $\mathcal{S}$ be a mixture set with a complete ordering $\succsim$. A function $u\colon \mathcal{S} \to \mathbb{R}$ is called a *measurable utility* if $u$ is order-preserving and linear, ie, $u$ satisfies:

(i) $u(a) > u(b)$ if and only if $a \succ b$, for all $a, b \in \mathcal{S}$;
(ii) $u(\lambda a + (1-\lambda)b) = \lambda u(a) + (1-\lambda)u(b)$ for all $a, b \in \mathcal{S}$ and $\lambda \in [0,1]$.

We can now state the von Neumann-Morgenstern theorem (1944), which was discovered independently by Ramsey (1931). Herstein and Milnor (1953) sharpened the result and gave an elegant proof, using a more general set of axioms.[9] Rather than using the original formulation of the theorem, we will use a formulation that is closer to the formulation by Herstein and Milnor.

**Theorem 3.16 (von Neumann and Morgenstern)** *Let $\mathcal{S}$ be a mixture set with a complete ordering $\succsim$. Then a measurable utility can be defined on $\mathcal{S}$ if and only if $\mathcal{S}$ satisfies the following three axioms:*

***Axiom 1 (substitutability)**. If $a \sim a'$, then $\lambda a + (1-\lambda)b \sim \lambda a' + (1-\lambda)b$ for all $b \in \mathcal{S}$ and $\lambda \in [0,1]$.*

***Axiom 2 (continuity)**. If $a \succ b \succ c$, then $b \sim \lambda a + (1-\lambda)c$ for some $\lambda \in [0,1]$.*

***Axiom 3 (monotonicity)**. If $a \succ b$, then $\lambda a + (1-\lambda)b \succ \mu a + (1-\mu)b$ if and only if $\lambda > \mu$.*

[9]Herstein and Milnor (1953) replaces the von Neumann-Morgenstern axiom set with:

**Axiom 1′ (weak substitutability)**. If $a, a' \in \mathcal{S}$ and $a \sim a'$, then $\frac{1}{2}a + \frac{1}{2}b \sim \frac{1}{2}a' + \frac{1}{2}b$ for all $b \in \mathcal{S}$.

**Axiom 2′ (continuity)**. The sets $\{\lambda \mid \lambda a + (1-\lambda)b \succsim c\}$ and $\{\lambda \mid c \succsim \lambda a + (1-\lambda)b\}$ are closed for all $a, b \in \mathcal{S}$.

Intuitively, Axiom 1 says that if an agent is indifferent between $a$ and $a'$, then the agent should also be indifferent when $a$ is replaced by $a'$ in a mixture. Axiom 2 and 3 together roughly state that if the agent prefers $b$ over $c$, and $a$ over $b$, then there is a number $\lambda$ such that given a choice between $b$ and a mixture $\mu a + (1-\mu)c$ of $a$ and $c$, the agent prefers $b$ when $\mu < \lambda$ (ie, the mixture contains too little of $a$ to be better than $b$), is indifferent when $\mu = \lambda$, and prefers the mixture when $\mu > \lambda$ (ie, the mixture contains enough of $a$ to be better than $b$).

**Remark 3.17** The proof given by Herstein and Milnor (1953) shows that the measurable utility is determined uniquely if we pick two elements $r_1 \succ r_0$ that are assigned utility 1 and 0, respectively. Given any $a, b \in \mathcal{S}$ satisfying $a \succsim r_1 \succ r_0 \succsim b$, and any $x \in \mathcal{S}$ with $a \succsim x \succsim b$, we define $\mu_{ab}(x)$ as the unique number in $[0,1]$ that satisfies

$$\mu_{ab}(x)a + (1-\mu_{ab}(x))b \sim x.$$

The measurable utility $u\colon \mathcal{S} \to \mathbb{R}$ can then be defined by

$$u(x) = \frac{\mu_{ab}(x) - \mu_{ab}(r_0)}{\mu_{ab}(r_1) - \mu_{ab}(r_0)}$$

for all $x$ between $a$ and $b$. Herstein and Milnor prove that the value of $u(x)$ is independent of the choice of $a$ and $b$.

More generally, the measurable utility is defined uniquely given any two values $u(r)$ and $u(r')$ with $r \succ r'$. In principle, this allows us to experimentally reconstruct $u(x)$ for any $x \in \mathcal{S}$: pick $a \succ b$ with known utility, and determine $\mu_{ab}(x)$ by asking the agent to compare $x$ with the mixture $\mu a + (1-\mu)b$ for different choices of $\mu$; then compute $u(x)$ from $u(a)$, $u(b)$, and $\mu_{ab}(x)$ using the formula:

$$u(x) = (u(a) - u(b))\mu_{ab}(x) + u(b).$$

As we have defined the space of analyses, it is not a mixture set. Is it reasonable to assume that speakers follow the axioms of utility theory, ie, that a speaker's preference ordering is defined on a mixture set that can be viewed as an extension of the space of analyses, and that this extended preference ordering satisfies the axioms of utility? Obviously, we do not wish to claim that human preferences are always perfectly rational, but we will present two different idealized models of speaker preferences where the axioms of utility apply: a probabilistic model where speakers must compare lotteries consisting of analyses that are drawn randomly according to a known probability distribution, and a corpus model where speakers must compare unordered sequences of analyses. The corpus model is most plausible from a linguistic point of view, but also more complicated than the probabilistic model.

## Probabilistic model of speaker preferences

In the probabilistic model of speaker preferences, we assume that speakers are not only capable of comparing individual analyses, but also lotteries of analyses. By a *lottery*, we mean a random variable over a finite set of analyses with known probabilities, and we use the notation $p_1 a_1 + \cdots + p_n a_n$ with $p_1 + \cdots + p_n = 1$ to denote the lottery in which analysis $a_i$ is drawn with probability $p_i > 0$. We will assume that the speaker's ordering of these lotteries satisfies the following four axioms:

> **Axiom P0 (completeness).** The speaker's ordering of the lotteries is complete.

That is, the speaker either prefers one lottery over another, or is indifferent between them, and the resulting ordering is transitive.

> **Axiom P1 (substitutability).** If the speaker is indifferent between $s$ and $s'$, then the speaker is indifferent between $\frac{1}{2}s + \frac{1}{2}t$ and $\frac{1}{2}s' + \frac{1}{2}t$.

That is, if the speaker is indifferent between $s$ and $s'$, then the speaker is also indifferent between a lottery and an identical lottery where $s$ is replaced by $s'$.

> **Axiom P2 (continuity).** If $a \succ b \succ c$, then $b \sim pa + (1-p)c$ for some $p \in [0, 1]$.

That is, by mixing a good and a bad lottery, we can find a lottery that the speaker will accept as an equivalent substitute for any intermediate lottery.

> **Axiom P3 (monotonicity).** If $a \succ b$, then $pa + (1-p)b \succ p'a + (1-p')b$ if $p > p'$.

That is, if we mix a good and a bad lottery, the more of the good lottery the mixture contains, the more the speaker likes it.

It is easy to verify that the set $\mathcal{S}$ of all lotteries is a mixture set, that Axiom P0 ensures that $\mathcal{S}$ is completely ordered, and that Axioms P1–P3 ensure that $\mathcal{S}$ satisfies the axioms of utility. The von Neumann-Morgenstern theorem then proves the existence of a measurable utility on $\mathcal{S}$, ie, the speaker's preference ordering can be expressed by means of a cost measure. Is it reasonable to assume that speakers are capable of comparing lotteries of analyses, and that these lotteries satisfy Axioms P0–P3? While this is not utterly implausible, it is perhaps not the most intuitive set of axioms for a linguistically motivated model of human communication either, and reasonable people might well reject it. But the probabilistic model of human communication is instructive because it provides a simple model in which cost measures arise as a necessary consequence.

**Corpus model of speaker preferences**

The probabilistic model of speaker preferences may strike some linguists as unintuitive from a linguistic point of view because it overestimates the capabilities of ordinary speakers. After all, it seems unlikely that speakers are capable of comparing complex lotteries with arbitrary probabilities. We therefore propose an alternative model, called the *corpus model*, whose assumptions about speaker capabilities are far weaker, and hopefully more appealing to many linguists.

In the corpus model, we assume that speakers are not only capable of comparing analyses, but that they are also capable of comparing arbitrary unordered collections of isolated analyses, called *corpora*. This corresponds to the situation where a teacher must rank the students in a course on the basis of their linguistic performance in a series of essays, so the assumption is not far from real-world practice. Given two corpora $a$ and $b$, we let $a + b$ denote the disjoint union of the two corpora, and we let $ka$ denote the corpus consisting of $k$ disjoint copies of $a$ where $k$ is a positive integer. Moreover, we assume that the speaker's ordering of the corpora satisfies the axioms C0–C4 below (for all corpora $a, b, c, d$ and positive integers $k$).

**Axiom C0**. The speaker's ordering is complete.

That is, given a choice between two different corpora, the speaker will always prefer one over the other, or be indifferent between them, and the resulting ordering is transitive.

**Axiom C1**. $a \succsim b$ iff $a + c \succsim b + c$.

That is, the speaker's relative ordering of two corpora only depends on the analyses that are not contained in both of them, and we can therefore add or delete a shared subcorpus without affecting the ordering.

**Axiom C2**. $a \succsim b$ iff $ka \succsim kb$.

That is, if the speaker prefers $a$ over $b$, then the speaker also prefers $k$ copies of $a$ over $k$ copies of $b$, and vice versa, ie, the ordering is unaffected by duplication of the two corpora.

**Axiom C3**. If $a \succsim b$ and $c \succsim d$, then $a + c \succsim b + d$.

That is, if the speaker prefers all the parts of a corpus when making a part-wise comparison with another corpus, the speaker also prefers the entire corpus.

**Axiom C4.** If $a \succ a'$ and $b$ is any corpus, then there exists a positive integer $k$ such that $ka \succ ka' + b$.

That is, if $a$ is better than $a'$, then we can always find $k$ such that the difference in goodness between $k$ copies of $a$ and $k$ copies of $a'$ is larger than the goodness of any given $b$. Or, phrased differently, a sufficiently large number of small errors will always outweigh a large error.[10]

Is it a reasonable idealization to assume that human speakers are capable of comparing corpora in a way that satisfies Axioms C0–C4? Obviously, given the limitations of the human mind, it would be surprising if real speakers were capable of following Axioms C0–C4 consistently, or comparing arbitrarily large corpora. However, as an idealized model of speaker behaviour, we think that the axioms are fairly reasonable. Both probabilistic language models and OT naturally lead to an ordering of corpora that satisfies Axioms C0–C3, and we find it hard to imagine a linguistically reasonable ordering of corpora that fails to satisfy them (although this may admittedly be a sign of our poor imagination). Axiom C4 is satisfied by probabilistic language models, but not by OT. However, this is a point where OT has been criticized (cf. Sorace and Keller 2005, §2.3): the

[10]Axiom C4 is violated by OT, since the violation of a high-ranked constraint always outweighs any number of violations of lower-ranked constraints.

psycholinguistic evidence seems to suggest that a large error can always be outweighed by a sufficient number of small errors — a phenomenon known as a *ganging up effect* — ie, Axiom C4 holds.

In the end, we leave it to the reader to decide whether these axioms are plausible or not, and whether it is possible to formulate a better model of speaker preferences that systematically predicts speakers' deviations from our corpus model. Our goal in this section is merely to show that if the reader accepts the corpus model as a plausible model of speaker behaviour, then the reader is also forced to accept that a speaker's preference ordering can be expressed by means of a cost measure. That is, in any linguistic model of human communication that is compatible with the corpus model, cost measures are a necessity, and not just a possibility.

We will do this by proving that we can extend the speaker's ordering of corpora to an ordering on a mixture set that satisfies the axioms of utility. By invoking the von Neumann-Morgenstern theorem, we can then show that the ordering can be expressed by means of a measurable utility, ie, by means of a cost measure. We start by formalizing our notion of a corpus.

**Definition 3.18** Let $\mathcal{A}$ be a set of analyses, and let $S$ be a subset of $\mathbb{R}$ that contains 0 and 1 while being closed under addition and multiplication. A *corpus* over $S$ is a function $c\colon \mathcal{A} \to S$ such that $c(a) = 0$ for all but finitely many $a$. The set of all corpora $c\colon \mathcal{A} \to S$ is called the *corpus set* induced by $\mathcal{A}$ over $S$.

Our corpus model corresponds to the case where $S$ is the set of all non-negative integers. However, we will extend this initial corpus set by replacing $S$ with the set $\mathbb{Q}$ of rational numbers, and the set $\mathbb{R}$ of real numbers.

**Definition 3.19** We define addition and scalar multiplication on a corpus set $\mathcal{C}$ by letting $c + c'$ denote the corpus that maps $a \in \mathcal{A}$ to $c(a) + c'(a)$, and letting $sc$ denote the corpus that maps $a \in \mathcal{A}$ to $sc(a)$, given $c, c' \in \mathcal{C}$ and $s \in S$. We let 0 denote the corpus that maps all $a \in \mathcal{A}$ to 0, and note that $c + 0 = c$ and $1c = c$ for all $c \in \mathcal{C}$.

We can now restate the axioms in our corpus model.

**Definition 3.20** Let $\mathcal{C}$ be a corpus set with a complete ordering $\precsim$. We say that $\mathcal{C}$ is a *corpus space* if $\mathcal{C}$ satisfies the following four axioms for all $a, b, c, d \in \mathcal{C}$ and positive integers $k$:

**Axiom C1.** $a \succsim b$ iff $a + c \succsim b + c$.
**Axiom C2.** $a \succsim b$ iff $ka \succsim kb$.
**Axiom C3.** If $a \succsim b$ and $c \succsim d$, then $a + c \succsim b + d$.
**Axiom C4.** If $a \succ a'$ and $b \in \mathcal{C}$, then there exists a positive integer $k$ such that $ka \succ ka' + b$.

Obviously, the corpus model corresponds to saying that the speaker has an ordering on the corpus set $\mathcal{C}$ induced by $\mathcal{A}$ over $\mathbb{N}_0$ that turns $\mathcal{C}$ into a corpus space. We will now show that $\mathcal{C}$ can be extended to corpus spaces $\mathcal{C}_\mathbb{Q}$ and $\mathcal{C}_\mathbb{R}$ over $\mathbb{Q}$ and $\mathbb{R}$, respectively.

**Definition 3.21** Let $\mathcal{C}_\mathbb{Q}$ be the corpus set induced by $\mathcal{A}$ over $\mathbb{Q}$. Given $c \in \mathcal{C}_\mathbb{Q}$, let $p_c$ denote the positive part of the corpus $c$, ie, the corpus that maps $a \in \mathcal{A}$ to $\max(0, c(a))$, and let $q_c$ denote the smallest positive integer such that $q_c c(a)$ is an integer for all $a \in \mathcal{A}$. Define an ordering $\succsim$ on $\mathcal{C}_\mathbb{Q}$ by

$$a \succsim b \qquad \text{iff} \qquad q_a q_b (p_a + p_{-b}) \succsim' q_a q_b (p_{-a} + p_b)$$

where $\succsim'$ denotes the ordering on $\mathcal{C}$.

**Proposition 3.22** *$\mathcal{C}_\mathbb{Q}$ is a corpus space over $\mathbb{Q}$ that contains $\mathcal{C}$, and the ordering on $\mathcal{C}_\mathbb{Q}$ coincides with the ordering on $\mathcal{C}$.*

*Proof.* Since $p_c = c$, $p_{-c} = 0$, and $q_c = 1$ for all $c \in \mathcal{C}$, it follows immediately that $\succsim$ coincides with $\succsim'$ on $\mathcal{C}$. To show that $\succsim$ is well-ordered, it suffices to show that $a \succsim b$ or $a \precsim b$ for all $a, b \in \mathcal{C}_\mathbb{Q}$. So suppose $a \succ b$ and $a \prec b$, then $q_a q_b (p_a + p_{-b}) \succ' q_a q_b (p_{-a} + p_b)$ and $q_a q_b (p_a + p_{-b}) \prec' q_a q_b (p_{-a} + p_b)$, a contradiction. To show that $\succsim$ is transitive, note that $a \succsim b$ and $b \succsim c$ implies $q_a q_b (p_a + p_{-b}) \succsim' q_a q_b (p_{-a} + p_b)$ and $q_b q_c (p_b + p_{-c}) \succsim' q_b q_c (p_{-b} + p_c)$. Thus, by Axiom C1 we can add $q_a q_b p_{-c}$ to the first inequality and $q_b q_c p_{-a}$ to the second inequality, use Axiom C2 to multiply the first inequality with $q_c$ and the second with $q_a$, and then use transitivity in $\mathcal{C}$ to conclude:

$$\begin{aligned} q_a q_b q_c (p_a + p_{-b} + p_{-c}) &\succsim' q_a q_b q_c (p_{-a} + p_b + p_{-c}) \\ &\succsim' q_a q_b q_c (p_{-a} + p_{-b} + p_c). \end{aligned}$$

By Axiom C1, we can now subtract $q_a q_b q_c p_{-b}$, giving $q_a q_b q_c (p_a + p_{-c}) \succsim' q_a q_b q_c (p_{-a} + p_c)$. By Axiom C2, we can divide with $q_b$, giving $q_a q_c (p_a +$

$p_{-c}) \succsim' q_a q_b(p_{-a} + p_c)$, ie, we have shown $a \succsim c$. This proves transitivity, ie, $\succsim$ is a complete ordering on $\mathcal{C}$.

To prove Axiom C1 on $\mathcal{C}_{\mathbb{Q}}$, note that the identity $a = p_a - p_{-a}$ gives $a - b = p_{a-b} - p_{b-a}$ and $a - b = p_a - p_{-a} - p_b + p_{-b}$, resulting in the identity

$$p_{a-b} + p_{-a} + p_b = p_{b-a} + p_a + p_{-b}.$$

Together with Axiom C1 and C2 and the definition of $\succsim$, this gives

$$\begin{aligned}
a \succsim b \quad &\Leftrightarrow \quad q_a q_b(p_a + p_{-b}) \succsim' q_a q_b(p_{-a} + p_b) \\
&\Leftrightarrow \quad q_a q_b(p_{a-b} + p_a + p_{-b}) \succsim' q_a q_b(p_{a-b} + p_{-a} + p_b) \\
&\Leftrightarrow \quad q_a q_b(p_{a-b} + p_a + p_{-b}) \succsim' q_a q_b(p_{b-a} + p_a + p_{-b}) \\
&\Leftrightarrow \quad q_a q_b p_{a-b} \succsim' q_a q_b p_{b-a} \\
&\Leftrightarrow \quad q_{a-b} p_{a-b} \succsim' q_{a-b} p_{b-a} \\
&\Leftrightarrow \quad a - b \succsim 0.
\end{aligned}$$

From this equivalence, we immediately get $a \succsim b$ iff $a + c \succsim b + c$, ie, $\mathcal{C}_{\mathbb{Q}}$ satisfies Axiom C1.

To prove Axiom C2, it suffices to show that for any $a \in \mathcal{C}_{\mathbb{Q}}$ and positive integer $k$, we have $a \succsim 0$ iff $ka \succsim 0$. We can prove this by combining Axiom C2 on $\mathcal{C}$ with the identity $kp_a = p_{ka}$ for any $k > 0$:

$$\begin{aligned}
a \succsim 0 \quad &\Leftrightarrow \quad q_a p_a \succsim' q_a p_{-a} \quad &&\Leftrightarrow \quad kq_a p_a \succsim' kq_a p_{-a} \\
&\Leftrightarrow \quad q_a p_{ka} \succsim' q_a p_{-ka} \quad &&\Leftrightarrow \quad ka \succsim kb
\end{aligned}$$

To prove Axiom C3, it suffices to show that $a \succsim 0$ and $b \succsim 0$ implies $a + b \succsim 0$. So assume $a \succsim 0$ and $b \succsim 0$, then $q_a p_a \succsim' q_a p_{-a}$ and $q_b p_b \succsim' q_b p_{-b}$, so Axiom C2 and C3 give

$$q_a q_b(p_a + p_b) \succsim' q_a q_b(p_{-a} + p_{-b})$$

The identity $p_{a+b} + p_{-a} + p_{-b} = p_{-a-b} + p_a + p_b$ combined with Axiom C1 and C2 then gives

$$\begin{aligned}
& q_a q_b(p_{a+b} + p_a + p_b) \succsim' q_a q_b(p_{a+b} + p_{-a} + p_{-b}) \\
\Rightarrow \quad & q_a q_b(p_{a+b} + p_a + p_b) \succsim' q_a q_b(p_{-a-b} + p_a + p_b) \\
\Rightarrow \quad & q_a q_b p_{a+b} \succsim' q_a q_b p_{-a-b} \\
\Rightarrow \quad & q_{a+b} p_{a+b} \succsim' q_{a+b} p_{-a-b} \\
\Rightarrow \quad & a + b \succsim 0
\end{aligned}$$

To prove Axiom C4, it suffices to show that $a \succ 0$ and $b \in \mathcal{C}_{\mathbb{Q}}$ implies that there exists a positive integer $k$ such that $ka \succ b$. Suppose this does not hold, then there exists $a, b \in \mathcal{C}_{\mathbb{Q}}$ such that $a \succ 0$ and $ka \prec b$ for all positive integers $k$; since $p_b \succsim b$, we may assume $p_{-b} = 0$ without any loss of generality. We then have $q_a p_a \succ q_a p_{-a}$ and hence $q_a q_b p_a \succ q_a q_b p_{-a}$, but $q_{ka} q_b p_{ka} \prec q_{ka} q_b (p_{-ka} + p_b)$ and hence $k \cdot q_a q_b p_a \prec k \cdot q_a q_b p_{-a} + q_a q_b p_b$ for all $k \in \mathbb{N}$, contradicting Axiom C4 in $\mathcal{C}$.

We have proved that the ordering on $\mathcal{C}_{\mathbb{Q}}$ is complete, satisfies Axioms C1–C4, and coincides with the ordering on $\mathcal{C}$, ie, the proposition holds. □

Since we can replace $a$ with $-a$ if $a \prec 0$, we will assume without loss of generality that $a \succsim 0$ for all $a \in \mathcal{A}$. With this assumption, we will now extend $\mathcal{C}_{\mathbb{Q}}$ to the corpus set $\mathcal{C}_{\mathbb{R}}$.

**Definition 3.23** Let $\mathcal{C}_{\mathbb{R}}$ be the corpus set induced by $\mathcal{A}$ over $\mathbb{R}$, with $a \succsim' 0$ for all $a \in \mathcal{A}$ where $\succsim'$ denotes the ordering on $\mathcal{C}_{\mathbb{Q}}$. Given $c \in \mathcal{C}_{\mathbb{R}}$, let

$$L_n(c) = \frac{1}{n} \sum_{a \in \mathcal{A}} \lfloor nc(a) \rfloor a$$

where $\lfloor r \rfloor$ denotes the largest integer $\leq r$.[11] Define an ordering $\succsim$ on $\mathcal{C}_{\mathbb{R}}$ by defining $c \succsim c'$ iff $c - c' \succsim 0$, and

$$c \succsim 0 \qquad \text{iff} \qquad \forall m\, \exists n > m\colon L_n(c) \succsim' -\frac{u_c}{m}$$

where $u_c$ is the corpus defined by $u_c(a) = 1$ if $c(a) \neq 0$, and $u_c(a) = 0$ if $c(a) = 0$. Equivalently, $c \prec 0$ iff $\exists m\, \forall n > m\colon L_n(c) \prec' -\frac{u_c}{m}$.

Rather than giving a direct proof that $\mathcal{C}_{\mathbb{R}}$ is a corpus space, we prove that $\mathcal{C}_{\mathbb{R}}$ is a mixture set that satisfies Herstein and Milnor's axioms of utility. We start by proving the following lemma.

[11]Our definition of $L_n(c)$ only works under the assumption that $a \succsim 0$ for all $a \in \mathcal{A}$. To deal with general $\mathcal{A}$, it is necessary to replace our definition of $L_n(c)$ with

$$L_n(c) = \frac{1}{n} \sum_{a \in \mathcal{A}} \ell_a(nc(a))a$$

where $\ell_a(r) = \lfloor r \rfloor$ if $a \succ 0$, $\ell_a(r) = \lceil r \rceil = -\lfloor -r \rfloor$ if $a \prec 0$, and $\ell_a(r) = 0$ if $a \sim 0$. Similarly, we must define $u_c(a) = -1$ if $a \prec 0$, and $u_c(a) = 0$ if $a \sim 0$.

**Lemma 3.24** *Let $c \in \mathcal{C}_{\mathbb{R}}$, and let $m, n$ be positive integers. Then*

$$-\frac{u_c}{\min(m,n)} \prec' L_m(c) - L_n(c) \prec' \frac{u_c}{\min(m,n)}.$$

*Proof.* Since $x - 1 < \lfloor x \rfloor \le x$, we have

$$\frac{1}{m}\lfloor ms \rfloor - \frac{1}{n}\lfloor ns \rfloor < \frac{ms}{m} - \frac{ns-1}{n} = \frac{1}{n} \le \frac{1}{\min(m,n)}.$$

Since $a \succsim' 0$ for all $a \in \mathcal{A}$, we therefore have

$$L_m(c) - L_n(c) = \sum_{a \in \mathcal{A}} \left( \frac{1}{m}\lfloor mc(a) \rfloor - \frac{1}{n}\lfloor nc(a) \rfloor \right) a \prec' \frac{u_c}{\min(m,n)}$$

The lemma follows immediately by exchanging $m$ and $n$. □

**Proposition 3.25** *$\mathcal{C}_{\mathbb{R}}$ is a completely ordered mixture space.*

*Proof.* It follows immediately from the definition of $\mathcal{C}_{\mathbb{R}}$ that it is a mixture space. To prove that $\succsim$ is well-ordered, it suffices to prove that $c \succsim 0$ or $c \precsim 0$ for all $c \in \mathcal{C}_{\mathbb{R}}$. Suppose we can find $c \in \mathcal{C}_{\mathbb{R}}$ such that $c \succ 0$ and $c \prec 0$, then $\exists m_1\ \forall n > m_1\colon L_n(c) \succ' u_c/m_1$ and $\exists m_2\ \forall n > m_2\colon L_n(c) \prec' -u_c/m_2$, so for $n > \max(m_1, m_2)$ we have $L_n(c) \succ' u_c/m_1 \succ' 0$ and $L_n(c) \prec' -u_c/m_2 \prec' 0$, a contradiction since $\succsim'$ is well-ordered.

To prove that $\succsim$ is transitive, it suffices to prove that $c \succsim 0$ and $c' \succsim 0$ implies $c + c' \succsim 0$. So suppose $c \succsim 0$ and $c' \succsim 0$, and let $m$ be given. Then we can find $n > 2m$ and $n' > 2m$ such that $L_n(c) \succsim -u_c/2m$ and $L_{n'}(c') \succsim -u_{c'}/2m$. By Lemma 3.24, $L_n(c') - L_{n'}(c') \succ' -u_{c'}/2m$. Adding the three inequalities together, we therefore get:

$$L_n(c) + L_{n'}(c') \succ' -\frac{u_c}{2m} - \frac{u_{c'}}{2m} - \frac{u_{c'}}{2m} \succsim' -\frac{u_c + u_{c'}}{m}$$

Since $u_c + u_{c'} \succsim u_{c+c'}$, and $L_n(c + c') \succsim L_n(c) + L_n(c')$ follows from $\lfloor x \rfloor + \lfloor y \rfloor \le \lfloor x + y \rfloor$, the inequality above shows that

$$L_n(c + c') \succsim' -\frac{u_{c+c'}}{m}.$$

That is, we have proven that $c + c' \succsim 0$. □

**Proposition 3.26** *$\mathcal{C}_{\mathbb{R}}$ is an extension of $\mathcal{C}_{\mathbb{Q}}$, ie, the ordering $\succsim$ on $\mathcal{C}_{\mathbb{R}}$ coincides with the ordering $\succsim'$ on $\mathcal{C}_{\mathbb{Q}}$ when restricted to $\mathcal{C}_{\mathbb{Q}}$.*

*Proof.* It suffices to prove that if $c \in \mathcal{C}_{\mathbb{Q}}$, then $c \succsim' 0$ implies $c \succsim 0$. So suppose $c \succsim' 0$. Then $q_c c(a)$ is an integer for all $a \in \mathcal{A}$, so $L_n(c) = c$ if $n$ is a multiple of $q_c$. It follows immediately that for all $m$, there exists a multiple $n$ of $q_c$ such that $n > m$ and $L_n(c) = c \succsim 0 \succsim -u_c/k$, ie, we have proven that $c \succsim 0$. □

The substitutability axiom follows almost immediately from the following lemma.

**Lemma 3.27** *If $c \succsim 0$ in $\mathcal{C}_{\mathbb{R}}$ and $r > 0$, then $rc \succsim 0$.*

*Proof.* Suppose $c \succsim 0$ and $r \in ]0,1[$, and let $m$ be given. Choose $n > 2m$ such that $L_n(c) \succsim -u_c/2m$. Setting $x = i + s = i'/r + s'$ where $i, i'$ are integers and $s \in [0,1[$, $s' \in [0, 1/r[$, we see that $\lfloor rx \rfloor - r\lfloor x \rfloor = i' - ri = r(s - s') \geq -1$. Using $\lfloor x \rfloor \leq x$, we calculate

$$\frac{\lfloor nrc \rfloor}{n} - \frac{\lfloor nr \rfloor}{n} \cdot \frac{\lfloor nc \rfloor}{n} \geq \frac{\lfloor nrc \rfloor - r\lfloor nc \rfloor}{n} \geq -\frac{1}{n} \geq -\frac{1}{2m}.$$

We therefore have

$$L_n(rc) - \frac{\lfloor nr \rfloor}{n} \cdot L_n(c) \succsim -\frac{u_c}{2m}$$

Multiplying the inequality $L_n(c) \succsim -u_c/2m$ with $\lfloor nr \rfloor / n$, and using that $\lfloor nr \rfloor / n < 1$ when $r \in ]0,1[$, we get

$$\frac{\lfloor nr \rfloor}{n} \cdot L_n(c) \succsim -\frac{u_c}{2m} \cdot \frac{\lfloor nr \rfloor}{n} \succsim -\frac{u_c}{2m}.$$

Adding the two inequalities, we get

$$L_n(rc) \succsim -\frac{u_c}{m},$$

which proves that $rc \succsim 0$. The proof of Lemma 3.25 shows that for all positive integers $k$, $c \succsim 0$ implies $kc = c + \cdots + c \succsim 0$. Writing $r = k + r'$ where $k$ is an integer and $r \in [0,1[$, $c \succsim 0$ implies $kc \succsim 0$ and $r'c \succsim 0$, and hence $kc + r'c = rc \succsim 0$. This proves the lemma. □

**Proposition 3.28** *$\mathcal{C}_{\mathbb{R}}$ satisfies Herstein and Milnor's weak substitutability axiom, ie, if $a \sim a'$, then $\frac{1}{2}a + \frac{1}{2}b \sim \frac{1}{2}a' + \frac{1}{2}b$ for all $a, a', b \in \mathcal{C}_{\mathbb{R}}$.*

*Proof.* To prove the proposition, we note that $a - a' \succsim 0$ implies $\frac{1}{2}(a - a') = \frac{1}{2}a + \frac{1}{2}b - (\frac{1}{2}a' + \frac{1}{2}b) \succsim 0$ by Lemma 3.27. Exchanging $\succsim$ with $\precsim$, the proposition follows immediately. □

Herstein and Milnor's continuity axiom follows almost immediately from the following lemma.

**Lemma 3.29** *$\mathcal{C}_\mathbb{R}$ satisfies Axiom C4, ie, if $a \succ 0$ and $b \in \mathcal{C}_\mathbb{R}$, then there exists a positive integer $k$ such that $ka \succ b$.*

*Proof.* Suppose $a \succ 0$, then there exists $m$ such that $a' = L_m(a) \succ 0$, and since $L_m(a) \precsim a$, we have $0 \prec a' \precsim a$. Similarly, $b' = -L_1(-b)$ satisfies $b' \succsim b$. Since $a', b' \in \mathcal{C}_\mathbb{Q}$ and $a' \succ 0$, it follows from Axiom C4 on $\mathcal{C}_\mathbb{Q}$ that there exists $k$ such that $ka' \succ b'$. Since $a \succsim a'$, we have $ka \succsim ka'$, and combining this with $b' \succsim b$, transitivity therefore gives:

$$ka \succsim ka' \succ b' \succsim b.$$

□

**Proposition 3.30** *$\mathcal{C}_\mathbb{R}$ satisfies Herstein and Milnor's continuity axiom, ie, the sets $\{\lambda \mid \lambda a + (1 - \lambda)b \succsim c\}$ and $\{\lambda \mid \lambda a + (1 - \lambda)b \precsim c\}$ are closed for all $a, b, c \in \mathcal{C}_\mathbb{R}$.*

*Proof.* Define $U_{a,b,c} = \{\lambda \in \mathbb{R} \mid \lambda a + (1 - \lambda)b \succ c\}$. Since closed sets are defined as complements of open sets, Herstein and Milnor's continuity axiom follows if we can prove that $U_{a,b,c}$ is open for all $a, b, c \in \mathcal{C}_\mathbb{R}$. Suppose this is not the case, then we can find $a, b, c \in \mathcal{C}_\mathbb{R}$ such that $U_{a,b,c}$ is not open, ie, there exists $\lambda \in U_{a,b,c}$ such that for all $\epsilon > 0$ we can find $\delta \in ]-\epsilon, \epsilon[$ such that $\lambda + \delta \notin U_{a,b,c}$. Write $s = \lambda a + (1 - \lambda)b - c$ and $s' = a - b$. Then $\lambda \in U_{a,b,c}$ implies $s \succ 0$, and $\lambda + \delta \notin U_{a,b,c}$ implies $s + \delta s' \precsim 0$, so obviously $\delta s' \prec 0$. We may assume without loss of generality that $s' \succ 0$. By Lemma 3.27, we then have $\delta s' \succ -\epsilon s'$, and hence $0 \succsim s + \delta s' \succ s - \epsilon s'$. In particular, setting $\epsilon = \frac{1}{k}$ where $k$ is a positive integer, we have $s - \frac{1}{k}s' \prec 0$, so by Lemma 3.27, we have $ks - s' \prec 0$ for all positive integers $k$. This contradicts that $s \succ 0$ and Axiom C4 implies that $ks \succ s'$ for some $k$. From this contradiction, we conclude that $U_{a,b,c}$ is open for all $a, b, c \in \mathcal{C}_\mathbb{R}$, ie, Herstein and Milnor's continuity axiom holds. □

The interested reader is referred to Munkres (1975) for the definition of open and closed sets, continuity, etc. Propositions 3.25, 3.28, and 3.30 demonstrate that $\mathcal{C}_{\mathbb{R}}$ satisfies the axioms of utility, so from the von Neumann-Morgenstern theorem (Theorem 3.16), we can conclude that the ordering on $\mathcal{C}_{\mathbb{R}}$ can be expressed by means of a measurable utility. From the existence of a measurable utility, it immediately follows that $\mathcal{C}_{\mathbb{R}}$ satisfies Axioms C1–C4, ie, that $\mathcal{C}_{\mathbb{R}}$ is a corpus space. We therefore have:

**Theorem 3.31** *$\mathcal{C}_{\mathbb{R}}$ is a corpus space, and the ordering on $\mathcal{C}_{\mathbb{R}}$ can be expressed by means of a measurable utility, ie, there exists a linear order-preserving mapping $u\colon \mathcal{C}_{\mathbb{R}} \to \mathbb{R}$.*

From Propositions 3.22 and 3.26, we have:

**Theorem 3.32** *Let $\mathcal{C}$ be a corpus space over $\mathbb{N}_0$ with analyses $\mathcal{A}$. Then $\mathcal{C}$ can be extended to corpus spaces $\mathcal{C}_{\mathbb{Q}}$ and $\mathcal{C}_{\mathbb{R}}$, and the ordering on $\mathcal{C}$, $\mathcal{C}_{\mathbb{Q}}$, and $\mathcal{C}_{\mathbb{R}}$ can be expressed by means of a measurable utility $u\colon \mathcal{C}_{\mathbb{R}} \to \mathbb{R}$.*

We have thereby proven that if the reader accepts our corpus model of speaker preferences, then the reader is also forced to accept that these preferences can be encoded by means of a cost measure. From our point of view, the proponents of any language model should therefore ask themselves whether their language model can be extended so that it can be used to compare all corpora, and whether their extension satisfies Axioms C1–C4. If it does, then they should consider whether their theory can be stated by means of cost functions; if it fails to satisfy one or more axioms, they should ask themselves why this is so, and whether this is linguistically plausible. Finding a linguistically plausible model that fails to satisfy Axioms C1–C4 would, in our view, be a highly interesting result.

As observed in section 1.1, our axioms lead to a non-Chomskyan view of language: rather than viewing grammars as devices that specify whether particular analyses are well-formed or ill-formed, we must view grammars as devices that given an analysis computes a real number that quantifies the absolute ill-formedness of the analysis, thereby determining the speaker's ordering of different analyses. And rather than viewing parsing and generation as satisfaction problems, we must view them as optimization problems.

**Remark 3.33** As far as we know, nobody else has proposed an axiomatization of language speakers by means of speaker preferences and used the von Neumann-Morgenstern theorem to prove the existence of a cost measure. However, Parikh (2000, 2001) proposes a game-theoretic model of human communication that also builds on the ideas presented in von Neumann and Morgenstern (1944). Parikh's model describes the game-theoretic aspects of human communication, while our model describes the utility-theoretic aspects, and it therefore seems entirely possible to combine the two models into a single unified model of human communication.

## 3.5 Summary

In this chapter, we have introduced a vocabulary of cost operators and shown how they can be used to encode linguistic constraints on obligatory presence and absence, word order, agreement, island constraints, word distances, selectional restrictions, and time-dependent costs. While the probabilistic language model proposed in section 6.2 is a better and more principled way of specifying cost functions compared to the hand-written cost functions presented in this chapter, our examples demonstrate that our vocabulary of cost operators is expressive enough to encode a wide variety of linguistic constraints. In this chapter, we have also described how the notion of costs relates to grammaticality, OT rankings, probabilistic language models, and parallel distributed processing models based on neural networks. Finally, we have presented a linguistically plausible axiomatization of speaker preferences, and proven that in any model of human communication that is compatible with our axioms, speaker preferences can be expressed by means of a cost measure.

# Chapter 4

# Examples of linguistic analyses

We exemplify how the DG formalism can be used to analyze a wide range of linguistic phenomena from Danish, English, and German. In particular, we show how the subcategorization mechanisms in DG can account for numerals; how the word order mechanisms can account for topicalization, extraposition, and scrambling; how the filler mechanism can account for control constructions, relatives, and parasitic gaps; how simple fillers and gapping fillers can account for peripheral sharing and gapping coordinations; how dependencies and anaphora can account for discourse structure; how constraints on the periphery can account for the placement of punctuation marks; how lexical transformations can account for derivational and inflectional morphology, as well as expletives and passives; and how the theory has been tested on a large scale by creating a dependency treebank for Danish. Many of the analyses introduced in this chapter have previously been presented in (Kromann et al. 2003) in some form before.

The examples in this chapter are meant to illustrate how different linguistic constructions can be analyzed in DG, and to outline how we can write a rule-based lexicon that licenses these analyses. In chapter 6, we will present a probabilistic language model which provides an alternative way of licensing the analyses presented in this chapter.

## 4.1 Dependencies: numerals

***Summary***. *We show how the complement and adjunct mechanisms in DG can be used to account for cardinal numerals in Danish, as an example of how to use the complement and* *adjunct mechanisms in DG.*

Numerals form a small, closed word class, and are therefore well suited to demonstrating the use of complement and adjunct mechanisms in our theory. In this section, we will focus on the numeral system in Danish, which is identical to the numeral system in German, apart from some remnants of a 20-based number system. The numeral system in Danish is also very similar to the numeral system in English, and our analysis extends to German and English with very few changes. Here, we will only look at integers, and ignore the numeral expressions used for real numbers, fractions, times, formulas, etc., which must be described by additional rules. Our analysis has previously been outlined in (Kromann et al. 2003).

**Definition 4.1** A numeral phrase $X$ has a *phrasal value* $\|X\|$, which is the numerical value of the entire phrase, and a *head value* $|X|$, which is the numerical value of the head word.

In our analysis, the head word in a numeral phrase is always the numeral with the largest head value, unless the larger numeral can be analyzed as the second conjunct in a coordination, in which case the numeral that acts as first conjunct is chosen as the head. Consequently, the numeral phrase "tre hundrede og to" ("three hundred and two") is headed by "hundrede" (and therefore has phrasal value 302 and head value 100), whereas the numeral phrase "to og tyve" ("two and twenty" $=$ 22) is headed by "two", although "twenty" has larger head value.

In our analysis, a numeral can take two optional numeral complements: a multiplicative numeral complement with edge type numm on its left, and an additive numeral complement with edge type numa on its right. The additive numeral complement is either a cardinal numeral or a phrase consisting of the coordinator "og" ("and") and a numeral conjunct. A numeral can take a nominal object (just like other determiners[1]), but only when the numeral does not act as numeral complement to another numeral.

**Example 4.2** An example analysis of a numeral phrase involving all four kinds of dependencies is shown in Figure 4.1. The word "tusind" ("thou-

[1]We follow Hudson (2003, 2004) by analyzing determiners as pronouns with a possibly optional noun complement.

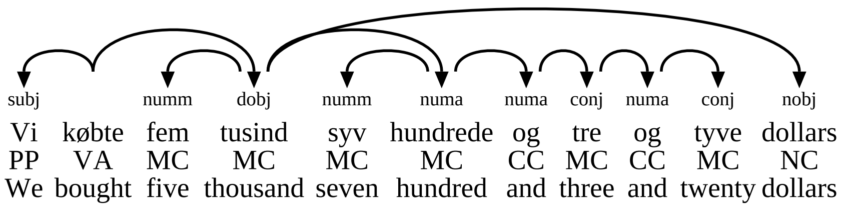

subj
numm
dobj
numm
numa
numa
conj
numa
conj
nobj
Vi købte fem tusind syv hundrede og tre og tyve dollars
PP VA MC MC MC MC CC MC CC MC NC
We bought five thousand seven hundred and three and twenty dollars

Figure 4.1: An analysis of a numeral phrase with additive and multiplicative numeral complements.

sand") has multiplicative numeral complement "fem" ("five"), additive numeral complement "syv hundrede og tre og tyve" ("seven hundred and three and twenty"), and ordinary noun complement "dollars". The word "tre" ("three") has coordinated additive numeral complement "og tyve" ("and twenty").

**Remark 4.3** The phrasal value of any composite numeral phrase can be computed by multiplying the head value of the head with the phrasal value of the multiplicative complement, and then adding the phrasal value of any additive complement. Thus, the value of "tre hundrede og to" ("three hundred and two") is computed as $3 * 100 + 2 = 302$, and the value of "to og tyve tusind tre hundrede og to" ("two and twenty thousand three hundred and two") is computed as $(2 + 20) * 1000 + (3 * 100 + 2) = 22,302$.

**Remark 4.4** Simple numerals in Danish can be grouped into seven subclasses (shown in Figure 4.2), which can in turn be organized in a hierarchy (shown in Figure 4.3), so that we can avoid redundancy in the lexicon by allowing lexemes to inherit the restrictions they place on their multiplicative and additive complements and conjuncts with respect to head value and phrasal value. These restrictions are shown in Figure 4.4.

We will now show how the restrictions in Figure 4.4 can be encoded within our framework.

**Example 4.5** The lexical entry for MC, the class of all cardinal numerals, is shown below. It specifies that cardinal numerals are indefinite pronouns (supertype PI), and introduces a complement frame inherited by all cardinal numerals that specifies that the numeral takes a multiplicative numm

| subclass | head values | words |
|---|---|---|
| MC0 | 0 | nul |
| MC1_9 | 1, . . . , 9 | en/et to tre fire fem seks syv otte ni |
| MC10_19 | 10, . . . , 19 | ti elleve tolv tretten fjorten femten seksten sytten atten nitten |
| MC20_90 | 20, 30, . . . , 90 | tyve tredive fyrre halvtres tres halvfjers firs halvfems |
| MC100 | 100 | hundred/hundrede |
| MC1000 | 1000 | tusind/tusinde |
| MCmillions | $10^6, 10^9, 10^{12}, \ldots$ | million milliard billion billiard . . . |

Figure 4.2: Different subclasses of simple numerals in our analysis of the Danish numeral system for integers.

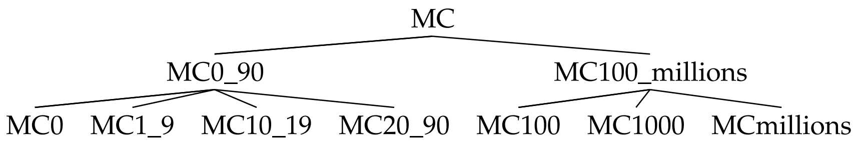


Figure 4.3: The hierarchical organization of different numeral subclasses.

complement and an additive numa complement, and a normal nobj complement; it also specifies cost functions that identify bad word orders, illegal nobj complements, and illegal extractions via a numeral complement edge. The functor \&functor_numeral is a code reference to a Perl subroutine that is used as functor for the complement frame, and which computes the phrasal value of the numeral phrase; for simplicity, we will ignore it.

| **superclass of $H$** | **multiplicative complement $M$** | **additive complement $A$** | **additive conjunct $C$** |
|---|---|---|---|
| MC0 | * | * | * |
| MC1_9 | * | * | $20 \leq \lvert C\rvert \leq 90$ |
| MC10_19 | * | * | * |
| MC20_90 | * | * | * |
| MC100 | $0 < \lvert M\rvert < 20$ | $0 < \lVert A\rVert < \lvert H\rvert$ | $0 < \lVert C\rVert < \lvert H\rvert$ |
| MC1000 | $0 < \lvert M\rvert < \lvert H\rvert$ $M$ is obligatory if $A$ is present | $0 < \lVert A\rVert < \lvert H\rvert$ $A$ must have numm complement | $0 < \lVert C\rVert < \lvert H\rvert$ $C$ must have numm comp. if $\lvert C\rvert \geq 100$ |
| MCmillions | $0 < \lvert M\rvert < \lvert H\rvert$ $M$ is obligatory | $0 < \lVert A\rVert < \lvert H\rvert$ $A$ must have numm complement | $0 < \lVert C\rVert < \lvert H\rvert$ $C$ must have numm comp. if $\lvert C\rvert \geq 100$ |

Figure 4.4: Licensing conditions for Danish numerals (* = ungrammatical).

```
type(MC)
    −> super(PI)
    −> cframe(numm_numa_nobj, \&functor_numeral,
            numm => MC, numa => MC|og:c, nobj => N)
    −> cost(bad_wo_numm, 100 * ([$\xleftarrow{\text{numm}}$ this] − ([$\xleftarrow{\text{land}}$ this] + [< this]))
    −> cost(bad_wo_numa, 100 * ([$\xleftarrow{\text{numa}}$ this] − ([$\xleftarrow{\text{land}}$ this] + [> this]))
    −> cost(bad_wo_nobj_numa, 100 * ([$\xleftarrow{\text{nobj}}$ this] + [$\xleftarrow{\text{land}}$ this]
            +[< this|[$\xleftarrow{\text{numa}}$ this]]))
    −> cost(bad_wo_numm_lmod, 100 * ([$\xleftarrow{\text{adjunct}}$ this] + [$\xleftarrow{\text{land}}$ this]
            +[< this] + [> [$\xleftarrow{\text{numm}}$ this]]))
    −> cost(illegal_nobj, 100 * [$\xrightarrow{\text{numm|numa}}$ (this | (og:c + [$\xrightarrow{\text{conj}}$ this]))] )
    −> cost(illegal_num_extraction, 100 * extract(node([$\xleftarrow{\text{numm|numa}}$ this])));
```

The class MC0_90 inherits all its complement frames and cost functions from MC, but also specifies that all numeral complements are illegal.

```
type(MC0_90)
    −> super(MC)
    −> cost(illegal_numm, 100 * [$\xleftarrow{\text{numm}}$ this])
    −> cost(illegal_numa, 100 * [$\xleftarrow{\text{numa}}$ this]);
```

The class MC1_9 modifies the illegal_numa cost function inherited from the class MC0_90 by allowing additive conjuncts if the conjunct is a numeral in the class MC20_90.

```
type(MC1_9)
    −> super(MC0_90)
    −> cost(illegal_numa, 100 * ([$\xleftarrow{\text{numa}}$ this]−(og:c+[$\xrightarrow{\text{conj}}$ MC20_90])));
```

The restrictions posed by the classes MC100, MC1000, and MCmillions are so similar that they warrant the creation of the subclass MC100_millions, a direct subclass of MC. This subclass specifies that a numm complement must have smaller head value than the head (eg, to avoid "thousand" being a numm complement of "hundred", as in "thousand hundred"), and that a numa complement must have smaller phrasal value than the head value of the head (eg, to avoid "eleven hundred" being a numa complement of "thousand", as in "thousand and eleven hundred"). It also specifies that a numm complement is obligatory, that a "numa" complement without a coordinator must have a "numm" complement, and that a "numa" conjunct in MC100_millions must have a "numm" complement. The type specification val(phrase_value, $n$) is assumed to return the phrasal value associated with the node $n$ (it can be implemented either as a feature that is set as a side effect of the semantic interpretation, or as a dynamic feature function), and val(head_value, $n$) is assumed to return the head value of $n$.

```
type(MC100_millions)
    −> super(MC)
    −> cost(illegal_numm,
        100 * (val(head_value, [$\xleftarrow{\text{numm}}$ this]) > val(head_value, this)))
    −> cost(illegal_numa,
        100 * (val(phrase_value, [$\xleftarrow{\text{numa}}$ this]) > val(head_value, this)))
    −> cost(no_numm, 100 * (this − [$\xrightarrow{\text{numm}}$ any]))
    −> cost(no_numm_in_numa, 100 * ([$\xleftarrow{\text{numa}}$ this] + MC − [$\xrightarrow{\text{numm}}$ any]))
    −> cost(no_numm_in_numac, 100 * ([$\xleftarrow{\text{conj}}$ [$\xleftarrow{\text{numa}}$ this] + og:c]
            +MC100_millions − [$\xrightarrow{\text{numm}}$ MC]));
```

Unlike the classes MC1000 and MCmillions, the class MC100 does not have any obligatory numm or numa complements, and it requires a numm complement to have head-value smaller than 20. This can be specified with the lexical entry:

```
type(MC100)
    -> super(MC100_millions)
    -> cost(no_numm, undef)
    -> cost(no_numm_in_numa, undef)
    -> cost(no_numm_in_numac, undef)
    -> cost(illegal_numm,
        100 * $(\mathsf{val}(\mathsf{head\_value}, [\xleftarrow{\mathsf{numm}} \mathsf{this}]) \geq 20)$);
```

The class MC1000 differs from MC100_millions by only requiring a numm complement in the presence of a numa complement without the "og" coordinator: eg, "one thousand two hundred" is grammatical, but "thousand two hundred" is not. This can be specified with the lexical entry:

```
type(MC1000)
    -> super(MC100_millions)
    -> cost(no_numm, 100 * $(\mathsf{this} + [\xrightarrow{\mathsf{numa}} \mathsf{MC}] - [\xrightarrow{\mathsf{numm}} \mathsf{MC}])$);
```

Finally, the class MCmillions differs from MC100_millions by agreeing in number with the obligatory numm complement. The number agreement can be specified with the following lexical entry:

```
type(MCmillions)
    -> super(MC100_millions)
    -> cost(bad_numm_agree,
            100 * $(\mathsf{val}(\mathsf{num}, [\xleftarrow{\mathsf{numm}} \mathsf{this}]) \neq \mathsf{val}(\mathsf{num}, \mathsf{this}))$);
```

The classes MC0 and MC20_90 inherit all their features from MC0_90:

```
type(MC0) -> super(MC0_90);
type(MC20_90) -> super(MC0_90);
```

Having defined all the numeral classes, we can now specify individual numerals with lexical entries of the form shown below:

```
type(en:mc) -> super(MC1_9) -> set(head_value, 1) -> set(num, sg);
type(et:mc) -> super(MC1_9) -> set(head_value, 1) -> set(num, sg);
type(to:mc) -> super(MC1_9) -> set(head_value, 2);
...
type(million:mc) -> super(MCmillions) -> set(head_value, 1e6);
```

**Remark 4.6** Our model gives a reasonable approximation to what most speakers of Danish recognize as atomic numerals, and how these atomic numerals combine to form compounds. However, speakers do not always form the same analyses, and many words that are analyzed as atomic numerals by ordinary speakers, will undoubtedly be recognized as compounds by etymologically educated speakers. The numbers from 13 to 19 are composites consisting of a number from 3 to 9 followed by the suffix "ten", and the numbers 11 and 12 probably originate from the old Germanic expressions for "one left" and "two left" (after counting to 10). Likewise, the numbers 20, 30, and 40 are remnants of a 10-based system, and the numbers 50, 60, 70, 80, and 90 are remnants of a 20-based system — eg, the word "halvtres" is a short-hand for "halv-tre-sinds-tyve" ("half three times twenty"), ie, the number $2.5 * 20 = 50$. There is also a productive system behind words like "million", "milliard", "billion", "billiard", "trillion", etc. which is shared by most European languages (except English): a latin numeral phrase denoting $n$ followed by "illion" means $10^{6n}$, and $10^{6n+3}$ when followed by "illiard" — although few speakers of modern Danish know the correct meaning of these numerals.

Several things can be learned from this elaborate account of the numeral system within our framework. First of all, our formalism is powerful enough to account for a non-trivial syntactic phenomenon that involves interaction with the compositional semantics (in this case, the calculation of phrasal values). However, even though we have ignored many aspects of the numeral system — including case agreement, negative numbers, fractions, intervals, approximations, times, dates, etc. — the resulting lexicon still involves a relatively large number of complex rules that govern word order, obligatoriness, extraction, and syntactic and semantic restrictions on the possible complements. As in other syntax formalisms, writing a thorough wide-coverage grammar for an entire language is therefore a serious challenge — an observation which motivates our interest in statistical learning techniques in chapters 5 and 6.

## 4.2 Word order: topicalization, extraposition, scrambling

***Summary.*** *We describe how DG deals with word order phenomena such as topicalization, extraposition, scrambling, partial verb phrases, and obligatory extraction.*

There is a wide range of linguistic constructions where the dependency structure is unremarkable, but where the word order is so complicated that the constructions are viewed as a challenge in linguistic theories. For example, topicalizations, extrapositions, scramblings, and partial verb phrases have received a great deal of attention because of their difficult word order. In this section, we will describe how DG accounts for these phenomena.

In DG, the word order is controlled by both hard constraints (the continuous surface tree, the deep upwards movement principle, and the landing rules associated with the landing sites) and soft constraints (word order cost functions at landing sites, cost functions for obligatory presence or absence at governors, and island cost functions at island nodes on the extraction path that can be used to prevent the extraction). In some sense, movement in DG resembles Chomsky's *move $\alpha$* operation in that if there is a suitable landing site, movement is permitted by default, unless the extraction is explicitly blocked by some node on the extraction path. As in other syntactic theories, finding the right word order rules is a delicate balance between *undergeneration* (ie, ruling out word orders that are acceptable in the language we wish to model) and *overgeneration* (ie, permitting word orders that are unacceptable in the modelled language).

## Topicalization

Topicalization has historically been seen as a central phenomenon in virtually all formal syntactic theories, perhaps because it is a relatively frequent word order phenomenon in most languages. We will therefore now show how DG accounts for topicalizations in Danish.

**Example 4.7** Topicalizations in Danish can be explained by assuming that a finite verb allows all nodes other than finite verbs to land in its left field, provided the left field does not contain any other landed node.[2] This can be encoded by means of a lexical entry like:

[2]The landing rule is obviously a crude approximation. For example, vocatives and certain parentheticals are allowed to land in the frontal field, even in the presence of a topicalized phrase, as in examples (1) and (2) below. In (1), the vocative phrase "min elskede" ("my dear") is analyzed as a voc adjunct of the finite verb "har" ("have"), and in (2), the parenthetical sentence "og det indrømmer jeg" ("and that I admit") is analyzed as a modp adjunct of the finite verb "har" ("have").

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| (1) | Én | is, | min | elskede, | har | jeg | givet | Alfred | idag. |
| | One | ice-cream, | my | dear, | have | I | given | Alfred | today. |

'I have given Alfred *one* ice-cream today, my dear.'

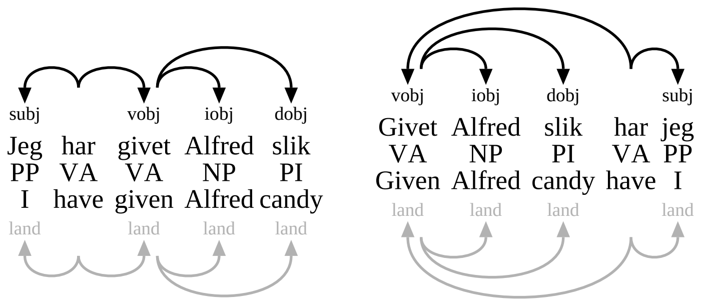

subj
vobj
iobj
dobj
Jeg har givet Alfred slik
PP VA VA NP PI
I have given Alfred candy
land
vobj
iobj
dobj
subj
Givet Alfred slik har jeg
VA NP PI VA PP
Given Alfred candy have I
land

Figure 4.5: Canonical word order (left) and VP topicalization (right).

```
type(verb_finite)
    -> lframe(xland => $-(\mathrm{V} + \mathrm{finite} \mid [\overset{\text{deep}}{\longleftarrow} \text{this}])$)
    -> cost(bad_xland_pos, 100 * $([\overset{\text{xland}}{\longleftarrow} \text{this}] + [> \text{this}])$)
    -> cost(bad_left_land, 100 * $([\overset{\text{surface}}{\longleftarrow} \text{this}] - [\overset{\text{adjunct}}{\longleftarrow} \text{this}] + [< \text{this}]$
            $+[< [\overset{\text{surface}}{\longleftarrow} \text{this}] - [\overset{\text{adjunct}}{\longleftarrow} \text{this}] + [< \text{this}]])$);
```

This rule captures most topicalizations in Danish, including topicalizations where the topicalized phrase is headed by a preposition or an infinite verb; it excludes all topicalizations where the left field contains more than one normal complement or topicalized node.

**Example 4.8** Consider the dependency structure expressed by the Danish sentence "Jeg har givet Alfred slik" ("I have given Alfred candy"). Danish allows the dependency structure of this sentence to be realized in many different ways which express more or less the same meaning, but with slightly different topic-focus structures: Apart from the canonical word order (Figure 4.5 left), Danish also allows topicalization of the verb phrase (Figure 4.5 right), and topicalization of the indirect and direct object (Figure 4.6). In Figure 4.5 and 4.6, the dependency trees are shown above the sentence, and the surface trees below the sentence (fillers are omitted for brevity). Surface edges for extraced nodes are black, whereas surface edges for non-extracted nodes (which are usually omitted in DG visualizations)

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (2) | Én | is, | og | det | indrømmer | jeg, | har | jeg | givet | Alfred | idag. |
| | One | ice-cream, | and | that | admit | I, | have | I | given | Alfred | today. |

'I have given Alfred *one* ice-cream today, and that I admit.'

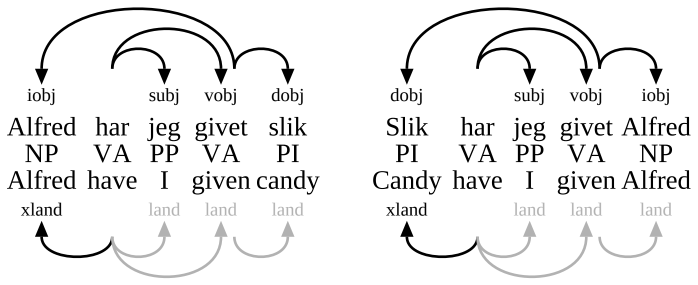

iobj
subj
vobj
dobj
Alfred har jeg givet slik
NP VA PP VA PI
Alfred have I given candy
xland
land
land
land
dobj
subj
vobj
iobj
Slik har jeg givet Alfred
PI VA PP VA NP
Candy have I given Alfred
xland
land
land
land

Figure 4.6: Topicalization of indirect object (left) and direct object (right).

are grey. The four graphs have the same dependency structure, and only differ with respect to their surface trees: the two surface trees in Figure 4.5 have the same tree structure, but differ with respect to the ordering of the landed nodes of the finite verb, and in the two surface trees in Figure 4.6, the topicalized object lands on the finite verb rather than its governor.

**Remark 4.9** In Danish, there is an even greater word order variation in poetry and songs, since speakers can relax the usual word order constraints to ensure rhymes and metric — in DG, this could be modelled by introducing cost functions that impose a cost for violations of the speaker's chosen scheme for rhymes and metric, and by reducing word order violation costs in the presence of a metric. In a probabilistic language model, this can be achieved by introducing genre as a conditional variable. Figure 4.7 shows two word orders for our original dependency tree, which would be acceptable in songs or poetry. In both examples, the requirement that the frontal field must contain only one landed node is relaxed, and the second example even contains two extracted nodes.

### Partial verb phrases

Partial verb phrases (PVPs) in German (Nerbonne et al. 1994, pp. 109–150) is another word order phenomenon where a dependency structure may have several different surface realizations with approximately the same meaning, but slightly different topic-focus structures. DG handles it in the same way as topicalization: by assuming that certain words may function as landing sites for the phrases that have been extracted from the partial verb phrase.

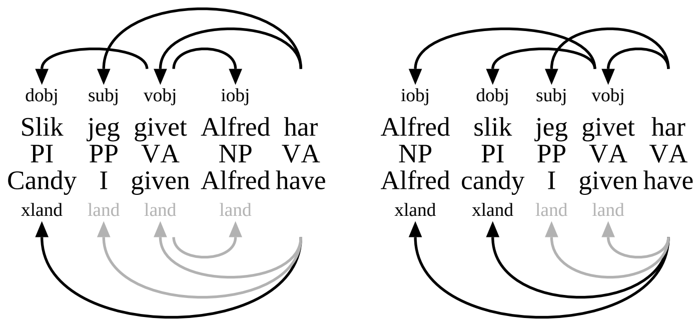


Figure 4.7: More complicated topicalizations and scramblings that could be used in poetry and songs because of their trochaic metric.

**Example 4.10** Consider the German sentences below (from Nerbonne 1994, p. 112).

(1)

| | | | | | | |
|---|---|---|---|---|---|---|
| Er | kann | seiner | Tochter | ein | Märchen | erzählen. |
| He | can | his | daughter | a | fairy-tale | tell. |

‘He can tell his daughter a fairy tale.’

German allows the words in this sentence to be reordered by topicalizing the embedded verb phrase, but letting some of the dependents in the embedded verb phrase remain in their original position, as in (2)–(4) below.

(2)

| | | | | | | |
|---|---|---|---|---|---|---|
| Ein | Märchen | erzählen | kann | er | seiner | Tochter. |
| A | fairy-tale | tell | can | he | his | daughter. |

(3)

| | | | | | | |
|---|---|---|---|---|---|---|
| Seiner | Tochter | erzählen | kann | er | ein | Märchen. |
| His | daughter | tell | can | he | a | fairy-tale. |

(4)

| | | | | | | |
|---|---|---|---|---|---|---|
| Erzählen | kann | er | seiner | Tochter | ein | Märchen. |
| Tell | can | he | his | daughter | a | fairy-tale. |

**Remark 4.11** We can explain PVPs by assuming that the finite verb does not only allow a topicalized constituent (in this case, the inner VP) to land on it, but also allows one or more extracted NPs to land in its middle field. We can encode this in the DG lexicon by modifying the lexical entry for finite verbs from Example 4.7 so that the bad_xland_pos cost function does not punish NPs that land in the right field. In addition, we need a set of word order cost functions that ensures that the extracted NPs appear in the correct position in the middle field, which is normally after the finite

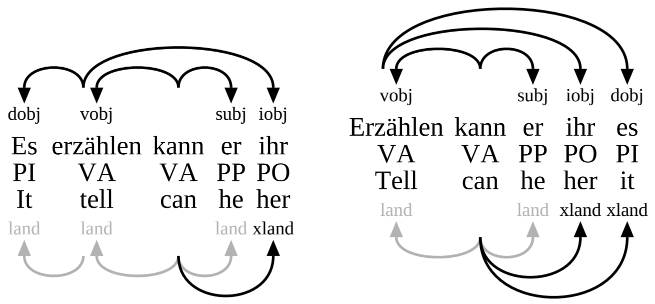


Figure 4.8: DG analyses of the partial verb phrases (2) and (4).

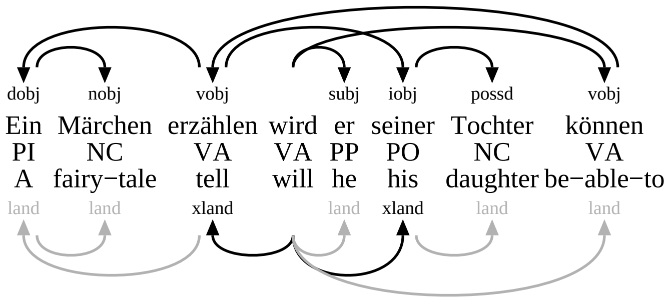
verb's subject, direct and indirect object (bad_order_xland_comp), but before the finite verb's verbal object (bad_order_vobj_xland). The resulting lexical entry is shown below.[3]

```
type(verb_finite)
    −> lframe(xland => −(V + finite | [$\xleftarrow{\text{deep}}$ this]))
    −> cost(bad_xland_pos, 100 * ([$\xleftarrow{\text{xland}}$ this] − noun + [> this]))
    −> cost(bad_left_land, 100 * ([$\xleftarrow{\text{surface}}$ this] − [$\xleftarrow{\text{adjunct}}$ this] + [< this]
            +[< [$\xleftarrow{\text{surface}}$ this] − [$\xleftarrow{\text{adjunct}}$ this] + [< this]]))
    −> cost(bad_order_xland_comp, 100 * ([$\xleftarrow{\text{xland}}$ this] + [> this]
            +[< [$\xleftarrow{\text{subj|dobj|iobj}}$ this] + [$\xleftarrow{\text{land}}$ this] + [> this]]))
    −> cost(bad_order_vobj_xland, 100 * ([$\xleftarrow{\text{xland}}$ this] + [> this]
            +[< [$\xleftarrow{\text{vobj}}$ this] + [$\xleftarrow{\text{land}}$ this] + [> this]]));
```

With this lexical entry, our lexicon permits the analyses of (2) and (4) shown in Figure 4.8 (to save space, we have replaced all nouns with pronouns). In the analyses, the verbal object is topicalized, and one or more dependents of the verbal object are extracted from the topicalized VP so that they land in the middle field of the finite verb.

**Example 4.12** A more complicated example of a partial verb phrase is shown in (5) below. The associated DG analysis is shown in Figure 4.9: the VP

[3]The word order cost functions bad_order_xland_comp and bad_order_vobj_xland are a good first approximation to the word order constraints governing extracted nodes in the middle field, but the full set of constraints is probably quite complex. In practice, we expect that it will be simpler and more reliable if word order constraints are induced by a probabilistic language model, as described in Chapter 6.



dobj
nobj
vobj
subj
iobj
possd
vobj
Ein Märchen erzählen wird er seiner Tochter können
PI NC VA VA PP PO NC VA
A fairy−tale tell will he his daughter be−able−to
land
land
xland
land
xland
land
land

Figure 4.9: DG analysis of the partial verb phrase (5).

headed by "erzählen" ("tell") is topicalized, and the indirect object "seiner Tochter" ("his daughter") is extracted from the topicalized VP to the middle field of the finite verb "wird" ("will").

|  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
| (5) | Erzählen | wird | er | seiner | Tochter | ein | Märchen | können. |
|  | Tell | will | he | his | daughter | a | fairy-tale | be-able-to. |

**Extraposition, scrambling, and cross-serial dependencies**

Extraposition, scrambling, and cross-serial dependencies are other examples of word order phenomena where a dependency structure may have several surface realizations, with slightly different topic-focus structures. In extraposition in German (cf. Rambow and Joshi 1994; Reape 1994), a "zu" infinitive governed by a verb with verb-last order can be extracted from its canonical position before its governing verb to a position after the verb.

**Example 4.13** In German, the phrase "die Walzer zu spielen" ("to play the waltzes") can be moved from a position before "versprochen" to a position after "hat", as shown in (6) and (7).

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| (6) | weil | niemand | dem | Publikum | die | Walzer | zu | spielen | versprach |
|  | because | nobody | the | audience | the | waltzes | to | play | promised |

'because nobody promised the audience to play the waltzes'

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| (7) | weil | niemand | dem | Publikum | versprach | die | Walzer | zu | spielen |
|  | because | nobody | the | audience | promised | the | waltzes | to | play |

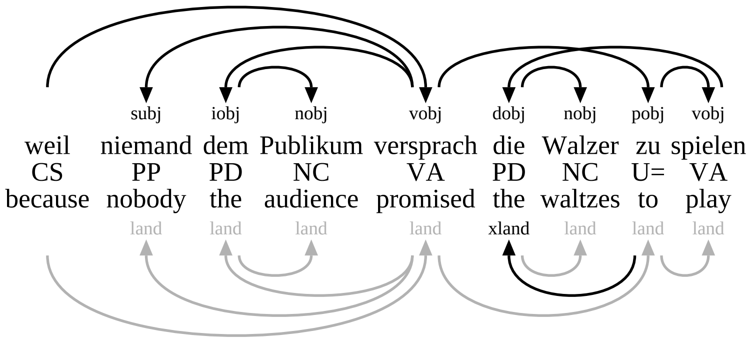

subj
iobj
nobj
vobj
dobj
nobj
pobj
vobj
weil niemand dem Publikum versprach die Walzer zu spielen
CS PP PD NC VA PD NC U= VA
because nobody the audience promised the waltzes to play
land
land
land
land
xland
land
land
land

Figure 4.10: DG analysis of the extraposition (7).

These examples can be explained by assuming that German verbs allow "zu" infinitives to land in their right field when they have verb-last order (ie, when they function as verbal objects). The resulting analysis is shown in Figure 4.10 (with fillers omitted). Note that the infinitive marker "zu" allows all the dependents of its verbal object as non-locally landed nodes, and that an infinitive verb does not allow any left landed nodes when acting as the verbal object of "zu". To control the word order in the left and right field of the verb, we will assume that the left field cannot contain extracted "zu" infinitives, and that the right field cannot contain anything but "zu" infinitives, when the verb is in verb-last order. This can be encoded by means of the lexical entry below.

```
type(verb)
    -> lframe(xland => (zu + [$\xrightarrow{\text{vobj}}$ any] − [$\xleftarrow{\text{deep}}$ this]) if [$\xrightarrow{\text{vobj}}$ this])
    -> cost(bad_right_land, 100 *
            (([$\xleftarrow{\text{surface}}$ this] + [> this] − (zu + [$\xrightarrow{\text{vobj}}$ any])) if [$\xrightarrow{\text{vobj}}$ this]))
    -> cost(bad_left_land, 100 * (([$\xleftarrow{\text{xland}}$ this] + [< this] + zu + [$\xrightarrow{\text{vobj}}$ any])));
```

Scrambling (cf. Rambow and Joshi 1994) is characterized by a deviation from the canonical word order in the middle field.

**Example 4.14** As an example of scrambling, consider (8) which has been derived from (7) by interchanging the order in which the subject "niemand" ("nobody") and the indirect object "dem Publikum" ("the audience") appears. (9) is identical to (8), except that we have inserted the verb

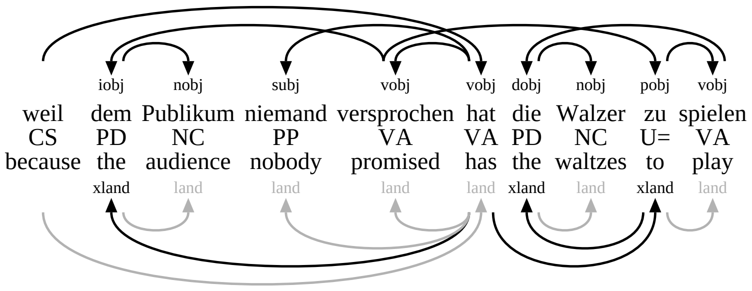


Figure 4.11: DG analysis of the extraposition and scrambling (9).

"hat" ("has") into the verbal complex, to demonstrate the non-locality of the extraction.

(8)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| weil | dem | Publikum | niemand | versprach | die | Walzer | zu | spielen |
| because | nobody | the | audience | promised | the | waltzes | to | play |

(9)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| weil | dem | Publikum | niemand | versprochen | hat | die | Walzer | zu | spielen |
| because | the | audience | nobody | promised | has | the | waltzes | to | play |

Scrambling is handled in the same way as topicalization and extraction: by positing a landing rule that allows scrambled phrases to land on a certain word. In a probabilistic language model, scrambling is detected as the possibility that an extracted node can be placed in a non-canonical position relative to the other landed nodes.

**Example 4.15** In the analysis of (9) shown in Figure 4.11, we assume that the finite verb "hat" ("has") allows the extracted indirect object "dem Publikum" ("the audience") to land on it, and that the scrambled word order is licensed by the finite verb. This can be encoded in our lexicon in the same way as topicalizations and extractions.

### Cross-serial dependencies

The DG analysis of cross-serial dependencies (cf. Steedman 1984; Rambow and Joshi 1994) follows the same pattern as the DG analysis of other word order phenomena, as illustrated by the following example.

**Example 4.16** Consider the Dutch cross-serial dependency (10) (cf. Steedman 1984; Rambow and Joshi 1994). The example is analyzed like other

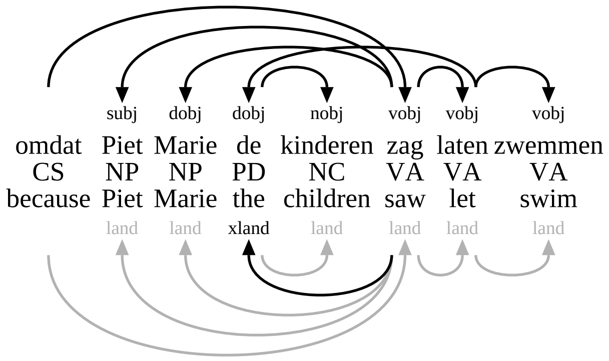


Figure 4.12: DG analysis of the cross-serial dependency (10).

word order phenomena: a dependent (the phrase "de kinderen" ("the children") in (10)) is moved from a subordinate verb ("laten") to the finite verb ("zag"). The resulting DG analysis is shown in Figure 4.12. The verb "zag" is assumed to be a control verb that passes on a filler for "Marie" to "laten", and "laten" is assumed to be a control verb that passes on a filler for "de kinderen" to "zwemmen"; for simplicity, the fillers generated by the control verbs have been omitted from the graph.

(10)

| omdat | Piet | Marie | de | kinderen | zag | laten | zwemmen |
|---|---|---|---|---|---|---|---|
| because | Piet | Marie | the | children | saw | let | swim |

'because Piet saw Marie let the children swim'

### Obligatory extraction

Obligatory extraction in Danish is another interesting word order phenomenon.

**Example 4.17** In Danish, objects that have been negated with the determiner "ingen" ("no") are treated in the word order as if they were sentence adverbials, ie, they cannot remain in the ordinary object position, but must be obligatorily fronted to sentence adverbial position (where objects are not normally permitted).

(11) a.

| Jeg | har | købt | en | bil | idag. |
|---|---|---|---|---|---|
| I | have | bought | a | car | today. |

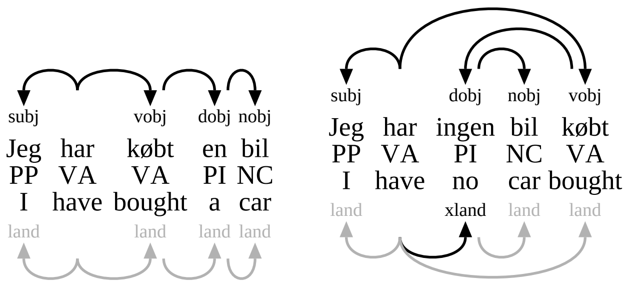


Figure 4.13: DG analysis of (12b) and the obligatory extraction (11a).

b. *Jeg har en bil købt idag.
I have a car bought today.

(12) a. *Jeg har købt ingen bil idag.
I have bought no car today.

b. Jeg har ingen bil købt idag.
I have no car bought today.
'I have not bought any car today.'

A DG analysis of (11a) and (12b) is shown in Figure 4.13. To encode this analysis in the DG lexicon, we must ensure that finite verbs such as "har" ("have") license direct and indirect objects headed by negated quantifier pronouns like "ingen" ("no") as external landed nodes, and that the finite verbs are equipped with word order cost functions that ensure that the extracted object ends up in sentence adverbial position.

**Example 4.18** Another example of obligatory extraction is provided by the Danish verb "gøre" ("do"), which subcategorizes for a direct object or an obligatorily topicalized verbal object. Thus, in the example below, the topicalized word order in (13a) is grammatical, whereas the canonical word order (13b) is ungrammatical. The corresponding DG analyses are shown in Figure 4.14.

(13) a. Synge har jeg ofte gjort.
Sing have I often done.
'I have often been singing.'

b. *Jeg har ofte gjort synge.
I have often done sing.

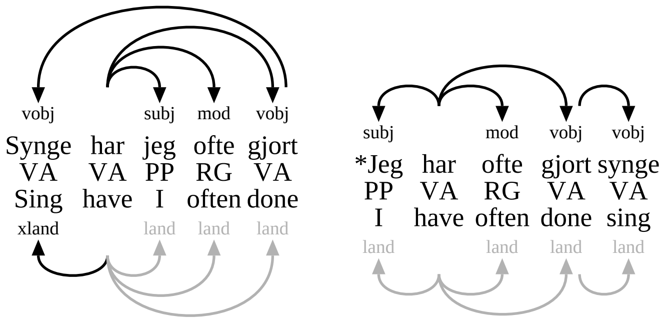


Figure 4.14: DG analysis of the topicalization (13a) and the ungrammatical canonical word order (13b).

We can explain these phenomena by assuming that the verb "gøre" ("do") is equipped with a cost function that induces a cost whenever the verbal object is non-topicalized, ie, whenever it lands in the right field of "gøre".

In this section, we have sketched how DG can be used to explain a wide range of word order phenomena, such as topicalizations, partial verb phrases, extrapositions, scramblings, cross-serial dependencies, and obligatory extractions. The analyses of these phenomena have followed a general pattern: introducing a landing frame in the word class associated with the landing site that will enable the extraction, and positing cost functions that can be used to constrain the possible landing sites and word orders.

## 4.3 Fillers: control, relatives, and parasitic gaps

***Summary***. *We describe how DG deals with phenomena that can be explained by means of simple fillers, such as verbal complexes, raising and control, relatives, and parasitic gaps.*

There is a wide range of linguistic constructions in which a single phrase seems to satisfy more than one dependency. As described in section 2.5, DG accounts for these constructions by assuming that one word (the filler licensor) can pass on a copy of another word (the filler source) to a third word (the filler governor). According to this analysis, an auxiliary or modal verb in a verbal complex passes on a copy of its subject to its subordinate verb, a control or raising verb passes on a copy of one of its dependents to its subordinate verb, and a relative verb passes on a copy of the relativized noun to some governor within the relative clause. Parasitic gaps are analyzed as filler constructions as well. In this section, we will show how DG accounts for these phenomena in detail. Elliptic coordination is also viewed as a filler phenomenon in DG, but since it involves fillers of a more complex kind, it will be dealt with separately in section 4.4.

In DG theory, fillers do not have any place in the linear word order. When drawing DG graphs with fillers, we are forced to draw the fillers somewhere in the linear order (usually in their canonical position if they were phonetically non-empty), but this placement is completely arbitrary and irrelevant for DG theory.

## Verbal complexes

In a verbal complex, a modal or auxiliary verb creates a filler that is used to satisfy the subject role of the verbal object.

**Example 4.19** Using simple fillers, we can use the following lexical entry for modal verbs like "will", "must", "can", "should", etc. that take a subject and an infinitival verbal object, and pass on a simple filler subject to the verbal object. The cost function induces the cost 100 if the filler fails to be present, or if it fails to be analyzed as the subject of the verbal object.

```
type(modal_verb)
    -> super(verb)
    -> cframe(subj_vobj, functor, subj => noun, vobj => 'verb+infinitive')
    -> fframe(modal, simple_filler, fill, src, $[\xleftarrow{\text{subj}} \text{this}]$)
    -> cost(filler_not_subj_of_vobj, 100 * $\neg([\xleftarrow{\text{fill}} \text{this}] + [\xleftarrow{\text{subj}} [\xleftarrow{\text{vobj}} \text{this}]])$);
```

Specific modal verbs can then inherit these features:

```
type('will:v1')->super(modal_verb);
type('may:v1')->super(modal_verb);
...
```

The resulting analysis of verbal complexes with modal and auxiliary verbs is shown in Figure 4.15.

## Raising and control

Raising and control constructions are analyzed like verbal complexes, ie, the verb is assumed to pass on a filler for one of its dependents to some

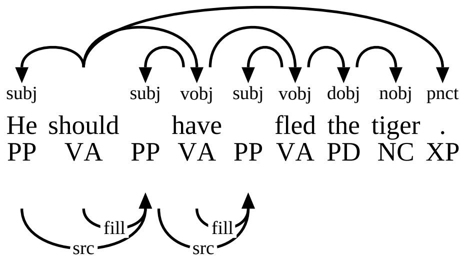


Figure 4.15: The use of fillers in the analysis of verbal complexes.

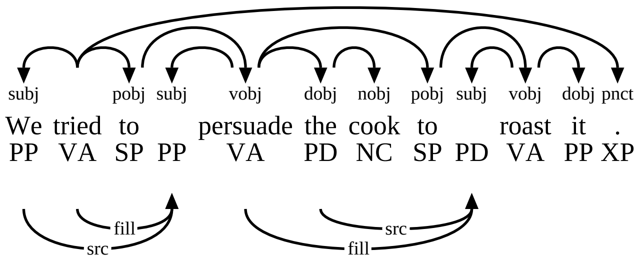

subj
pobj
subj
vobj
dobj
nobj
pobj
subj
vobj
dobj
pnct
We tried to persuade the cook to roast it .
PP VA SP PP VA PD NC SP PD VA PP XP
fill
src
src
fill

Figure 4.16: The use of fillers in the analysis of subject and object control.

subordinate verb.

**Example 4.20** The following lexical entry shows how filler frames and simple fillers can be used to encode subject control verbs like "try" (as in "The cook tried to roast the elephant"):

```
type('try:v1')
    −> super(verb)
    −> cframe(subj_pobj, functor, subj => noun, pobj => 'to:sp1')
    −> fframe(control_subj, simple_filler, fill, src, [$\xleftarrow{\text{subj}}$ this])
    −> cost('filler_not_subj_of_pobj:vobj',
            100 * $\neg$([$\xleftarrow{\text{fill}}$ this] + [$\xleftarrow{\text{subj}}$ [$\xleftarrow{\text{vobj}}$ [$\xleftarrow{\text{pobj}}$ this]]]));
```

The resulting analysis is illustrated in Figure 4.16.

**Example 4.21** The following lexical entry shows how we can encode object control verbs like "persuade" (as in "We persuaded the cook to roast the elephant"):

```
type('persuade:v1')
    −> super(verb)
    −> cframe(subj_dobj_pobj, functor, subj => noun, dobj => noun,
            pobj => 'to:sp1')
    −> fframe(control_dobj, simple_filler, fill, src, $[\xleftarrow{\text{dobj}} \text{this}]$)
    −> cost('filler_not_subj_of_pobj:vobj',
            100 * $\neg([\xleftarrow{\text{fill}} \text{this}] + [\xleftarrow{\text{subj}} [\xleftarrow{\text{vobj}} [\xleftarrow{\text{pobj}} \text{this}]]])$);
```

The resulting analysis is illustrated in Figure 4.16.

Raising is analyzed in the same way as control constructions.

**Example 4.22** In the raising construction "He seems to lie", we analyze "He" as the subject of "seems" (since there is person and number agreement between the subject and "seems"). We assume that "seems" acts like any other control verb by passing on a filler to the infinitival verb "lie".

**Remark 4.23** It is often argued that the subject of "seems" is not an argument of "seems" from a semantic point of view — in DG terms, this means that in the compositional semantics, the functor associated with "seems" does not consume the semantics associated with "He" as if it was a normal argument. However, from a syntactic point of view, the subject of "seems" behaves as any other subject, so even if it is not an argument of "seems" from a semantic point of view, it is convenient to analyze it as a subject of "seems" from a syntactic point of view.

## Relatives

Filler frames and simple fillers can also be used to account for relatives.

**Remark 4.24** In relatives, we will assume that the finite verb heading the relative clause attaches itself as a modifier of the relativized constituent and creates a filler that has the relativized constituent as its filler source. This can be encoded by means of the following lexical entry:

```
type(verb)
    −> aframe(relative, modifier, rel => noun)
    −> fframe(relative, simple_filler, fill, src, $[\xrightarrow{\text{rel}} \text{this}]$);
```

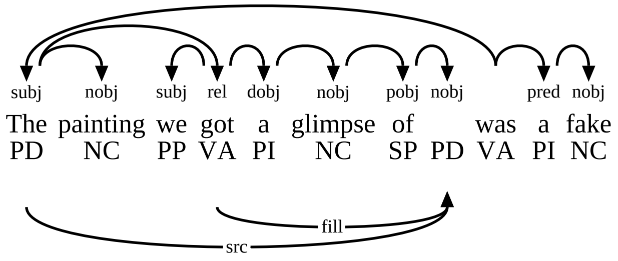

subj
nobj
subj
rel
dobj
nobj
pobj
nobj
pred
nobj
The painting we got a glimpse of was a fake
PD NC PP VA PI NC SP PD VA PI NC
fill
src

Figure 4.17: The analysis of a relative without a relative pronoun.

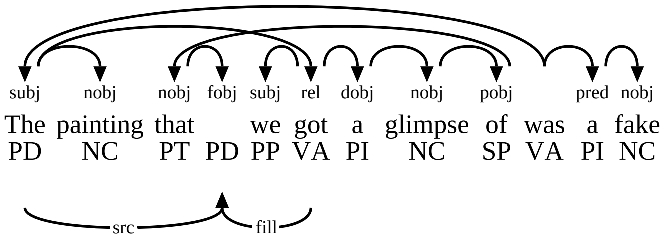

subj
nobj
nobj
fobj
subj
rel
dobj
nobj
pobj
pred
nobj
The painting that we got a glimpse of was a fake
PD NC PT PD PP VA PI NC SP VA PI NC
src
fill

Figure 4.18: The analysis of a relative with a relative pronoun.

Unlike modal verbs and control verbs, the relative verb does not place any restrictions on the governor of the filler, which allows the filler to be analyzed as a dependent of the relative verb itself or of any other word within the relative clause. The resulting analysis is illustrated by Figure 4.17.

**Remark 4.25** The analysis sketched in Remark 4.24 is maintained in the presence of relative pronouns such as "which", "that", or "whose". To avoid a competition between the relative pronoun and the filler with respect to satisfying the secondary dependency, we assume that the relative pronoun obligatorily consumes the filler by taking the filler as its so-called *filler object*. With this analysis, the antecedent of the relative pronoun is syntactically constrained to be identical to the filler object's filler source. Two examples of this analysis are shown in Figure 4.18 and 4.19.

*Free relatives* (or *wh-questions*) are relatives where the relativized phrase is a wh-phrase. Free relatives are analyzed just like other relatives, ie, the relative verb is assumed to create a filler that satisfies some dependency within the relative clause.

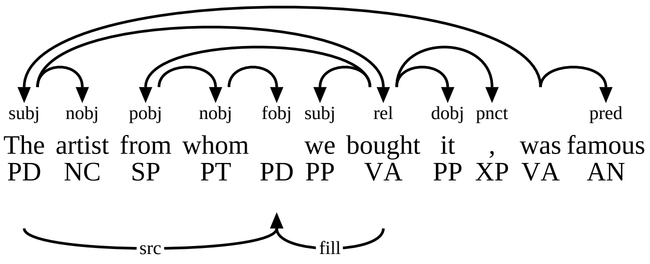

subj nobj pobj nobj fobj subj rel dobj pnct pred
The artist from whom we bought it , was famous
PD NC SP PT PD PP VA PP XP VA AN
src
fill

Figure 4.19: The use of fillers in relatives with a relative pronoun.

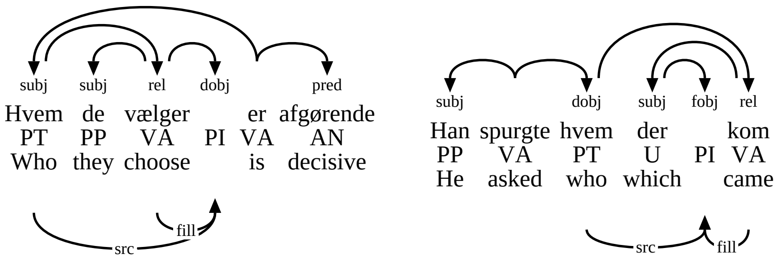


Figure 4.20: An analysis of the free relatives (14) and (15).

**Example 4.26** Three Danish examples of free relatives are shown in (14)–(16) below. The corresponding DG analyses are shown in Figure 4.20 and 4.21 (cf. Kromann et al. 2003, "Verbs").

(14) Hvem de vælger er afgørende.
Who they elect is decisive.
'It is decisive who they elect.'

(15) Han spurgte hvem der kom.
He asked who that came.
'He asked about who was coming.'

(16) Alle ved hvor svært det er.
Everybody knows how difficult it is.

The analysis of wh-questions as relatives is supported by the observation that the relative pronoun "der" is explicitly present in (15).

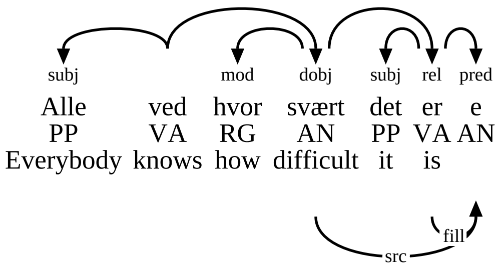

subj
mod
dobj
subj
rel
pred
Alle ved hvor svært det er e
PP VA RG AN PP VA AN
Everybody knows how difficult it is
fill
src

Figure 4.21: An analysis of the free relative (16).

## Parasitic gaps

Filler frames and simple fillers can also be used to account for parasitic gaps. There is a vast literature on parasitic gaps, summarized by Culicover and Postal (2001) and Parker (1999). Many linguistic formalisms provide an account of them, including GB (Chomsky 1982; Engdahl 1982, 1983), Combinatory Categorial Grammar (Steedman 1987), and HPSG (Pollard and Sag 1994). Parasitic gaps owe their name to GB, where the parasitic gap depends on the presence of another gap, called the *licensing gap*. In DG, gaps are used much more sparingly than in GB, and many of the constructions that involve gaps in GB are therefore analyzed as gapless extractions or topicalizations in DG.

**Definition 4.27** In DG, a *parasitic gap* can be described as a simple filler that is generated by a filler licensor (*parasitic licensor*) $l$ using a filler source $s$ that is related to $l$ by means of two intermediary nodes $g$ and $d$, as follows:

- $d$ must equal $l$ or have $l$ as its relative modifier;
- $g$ must be a verbal governor of $d$;
- $s$ must be either (a) a locally landed direct or indirect object of $g$ different from $d$ that precedes a locally landed subject of $g$, or (b) a non-subject noun that is extracted via any dependent edge of $g$ except for the edge containing $d$.

**Remark 4.28** The filler licensing rule for parasitic gaps can be encoded as the following filler rule, which is inherited by every word in the DTAG lexicon:

```
type(word)
−> fframe(parasitic, simple_filler, para, src, noun
    + var(d, this | [$\xrightarrow{\text{rel}}$ this])
    + var(g, verb + [$\xrightarrow{\text{deep}}$ var(d)])
    + ((([$\xleftarrow{\text{dobj|iobj}}$ var(g)] + [$\xleftarrow{\text{surf}}$ var(g)] − var(d)
            +[< [$\xleftarrow{\text{subj}}$ var(g)] + [$\xleftarrow{\text{surf}}$ var(g)]])
      | (extract(var(g)) − [$\xleftarrow{\text{subj}}$ any] − var(d) − extract(var(d)))));
```

The lexical entry proposed in Remark 4.28 can account for most of the examples that have been proposed in the literature on parasitic gaps.

**Example 4.29** The lexical entry for parasitic gaps can explain almost all of the examples of parasitic gaps presented in Pollard and Sag (1994, p. 182–200), including the examples (17)–(22) below (the words acting as *l*, *d*, *g*, and *s* are indicated with subscripts, and the word acting as the governor of the parasitic gap is indicated with the subscript *p*).[4]

(17) That was the$_s$ rebel leader who rivals$_{ld}$ of$_p$ shot$_g$.

(18) Those$_s$ boring old reports, Kim filed$_g$ without$_{ld}$ reading$_p$.

(19) Which$_s$ of our relatives should we send$_g$ snapshots$_{ld}$ of$_p$ to?

(20) Who$_s$ did$_g$ my$_{ld}$ talking to$_p$ bother?

(21) Who$_s$ did you consider$_g$ friends$_{ld}$ of$_p$ angry at?

(22) Here's the jerk that$_s$ I expected$_g$ my$_{ld}$ pictures of$_p$ to bother.

For example, in (17), the word "rivals" acts as parasitic licensor *l* for the parasitic gap and as intermediary node *d*, the word "shot" acts as the governor *g* of *d* (*d* is the direct object of *g*), and the word "who" acts as the filler source *s*. The resulting analysis is shown in Figure 4.22.

**Example 4.30** The lexical entry for parasitic gaps can also account for the examples (23)–(26) below from Haegeman (1994/1991, p. 473–478):

(23) Poirot is a man whom$_s$ you distrust$_g$ when$_{ld}$ you meet$_p$.

[4]Our lexical entry overgenerates with respect to the example "*That was the rebel leader rivals of shot", since it licenses a parasitic gap even in the absence of a relative pronoun. It is not clear to us whether this example really is ungrammatical, but if it is, the problem can be fixed easily by stipulating that a subject can only act as the parasitic licensor of a parasitic gap if the filler source is a non-filler.

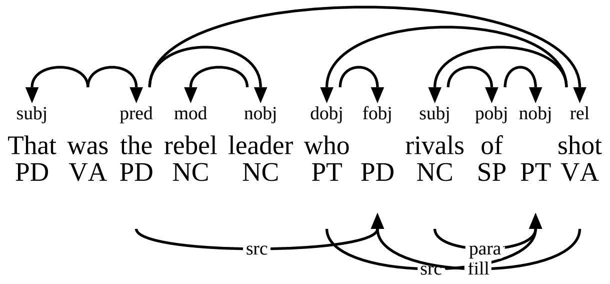

subj
pred
mod
nobj
dobj
fobj
subj
pobj
nobj
rel
That was the rebel leader who rivals of shot
PD VA PD NC NC PT PD NC SP PT VA
src
para
src
fill

Figure 4.22: Analysis of the parasitic gap (17).

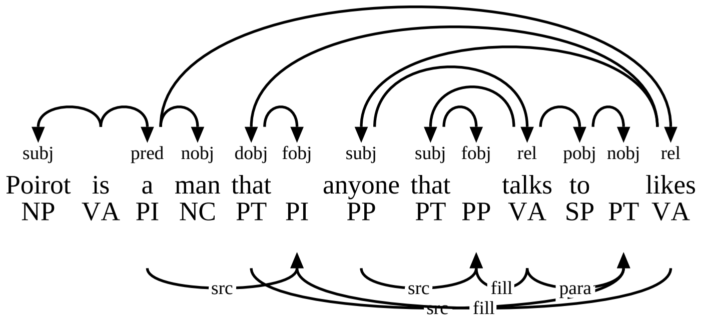

subj
pred
nobj
dobj
fobj
subj
subj
fobj
rel
pobj
nobj
rel
Poirot is a man that anyone that talks to likes
NP VA PI NC PT PI PP PT PP VA SP PT VA
src
src
fill
para
src
fill

Figure 4.23: Analysis of the parasitic gap (24).

(24) Poirot is a man that$_s$ anyone$_d$ that talks$_l$ to$_p$ usually likes$_g$.

(25) Poirot is a man who$_s$ I interviewed$_g$ before$_{ld}$ hiring$_p$.

(26) Poirot is a man who$_s$ I interviewed$_g$ before$_{ld}$ wondering when to hire$_p$.

Example (24) is particularly interesting, because the parasitic licensor $l$ ("talks") does not coincide with the node $d$ ("anyone"); the corresponding DG analysis is shown in Figure 4.23.

**Remark 4.31** In direct and indirect object relatives, such as (24), there is an ambiguity with respect to how we select the filler source for the parasitic gap: we can either select the relative pronoun as filler source, or its filler object, which is a filler licensed by the relative verb. The two readings have the same semantic interpretation when the relative pronoun and the filler are coreferent, but it is also possible to find examples where the two readings differ with respect to their interpretation, as in (27) and (28) below,

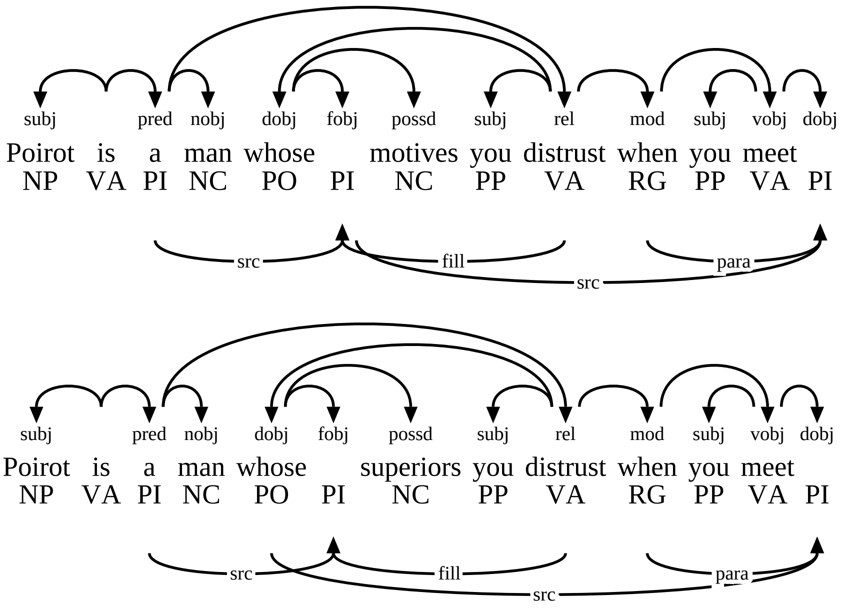

subj pred nobj dobj fobj possd subj rel mod subj vobj dobj
Poirot is a man whose motives you distrust when you meet
NP VA PI NC PO PI NC PP VA RG PP VA PI
src fill para src
subj pred nobj dobj fobj possd subj rel mod subj vobj dobj
Poirot is a man whose superiors you distrust when you meet
NP VA PI NC PO PI NC PP VA RG PP VA PI
src fill para src

Figure 4.24: A relative clause in which the parasitic filler may have two possible sources: the relativized noun, and the fronted direct object in the relative clause.

whose analyses are shown in Figure 4.24. Here, "whose" takes the relative filler for "a man" as its filler object, but "whose motives" is not coreferential with "a man". (27) only allows the interpretation where the direct object of "meet" is coreferential with "Poirot", and (28) only allows the interpretation where the direct object of "meet" is coreferential with Poirot's superiors.

(27) Poirot is a man whose$_s$ e motives you distrust$_g$ when$_{ld}$ you meet$_p$.

(28) Poirot is a man whose e$_s$ superiors you distrust$_g$ when$_{ld}$ you meet$_p$.

**Example 4.32** The lexical entry for parasitic gaps can also account for all the authentic examples of parasitic gaps collected by Potts (2004), except for the example "Please Inspect Before Using". The problem with this example is that the understood argument "it" is not present in the graph, and therefore cannot act as the filler source for the parasitic gap. To account for this example, we could extend our theory by letting an anaphoric filler be used as a default for missing complements. The licensor of the parasitic gap would then be able to use the anaphoric filler as a filler source for the parasitic gap. However, since we do not have a detailed understanding of how default complements should be implemented, we will leave this matter to future research.

In this section, we have shown how simple fillers can be used to account for constructions that involve secondary dependencies, such as verbal complexes, raising and control, relatives, and parasitic gaps. In the following section, we will demonstrate how fillers can be used to account for elliptic coordinations.

## 4.4 Coordination: sharing and gapping

***Summary**. We present a filler-based analysis of elliptic coordination, including peripheral sharing, subject sharing, and gapping.*

In the previous section, we saw how fillers can be used to account for verbal complexes, raising and control, relatives, and parasitic gaps. In this section, we will show that fillers can also be used to account for two kinds of elliptic coordination, peripheral sharing (often called right node raising in the literature) and gapping.

### Peripheral sharing

*Peripheral sharing* (or *right node raising*) is a phenomenon where one or more dependents are shared by several conjuncts. It is a relatively frequent construction. For example, approximately 0.66% of all words in the Danish Dependency Treebank have been marked as heads of a secondary conjunct with peripheral sharing.

**Example 4.33** In the peripheral sharing (29) below, the word "She" functions as the subject of both "was" and "listened", and the word "opera" functions as the nominal object of both "of" and "to" — ie, the words "She" and "opera" are *shared* by the words "was" and "listened", and "of" and "to", respectively. The DG analysis of this example is shown in Figure 4.25.

(29) She was very fond of, and often listened to, opera.

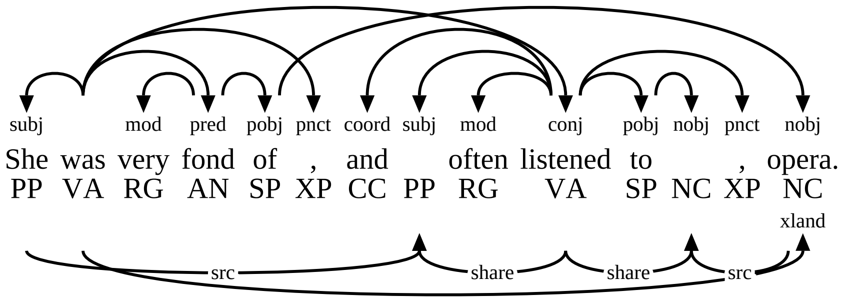

subj mod pred pobj pnct coord subj mod conj pobj nobj pnct nobj
She was very fond of , and often listened to , opera.
PP VA RG AN SP XP CC PP RG VA SP NC XP NC
xland
src share share src

Figure 4.25: Analysis of the peripheral sharing (29).

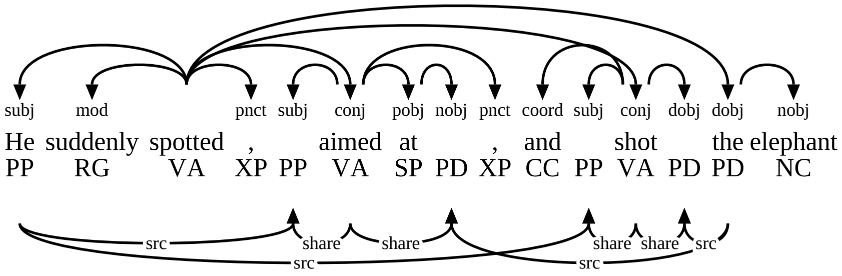

subj mod pnct subj conj pobj nobj pnct coord subj conj dobj dobj nobj
He suddenly spotted , aimed at , and shot the elephant
PP RG VA XP PP VA SP PD XP CC PP VA PD PD NC
src share share share share src
src src

Figure 4.26: Analysis of the peripheral sharing (30).

**Example 4.34** Peripheral sharing may involve any number of conjuncts, as shown by (30), where the word "He" functions as the subject of "spotted", "aimed", and "listened", and the phrase "the elephant" functions as the direct object of "spotted" and "shot", and as nominal object of "at". The DG analysis of this example is shown in Figure 4.26.

(30) He suddenly spotted, aimed at, and shot the elephant.

**Remark 4.35** What is characteristic about these examples of sharing is that the shared dependents are located at the periphery of the phrase, ie, the shared phrases are placed either before all non-shared words within the coordination, or after them. In both Danish and English (and many other European languages, we presume), sharing seems to be overwhelmingly peripheral, with one exception in Danish and German, known as *empty subject coordination* in some theories (cf. Hartmann 2000, p. 43ff): non-peripheral subjects can sometimes be shared as well, as illustrated by (31), where the

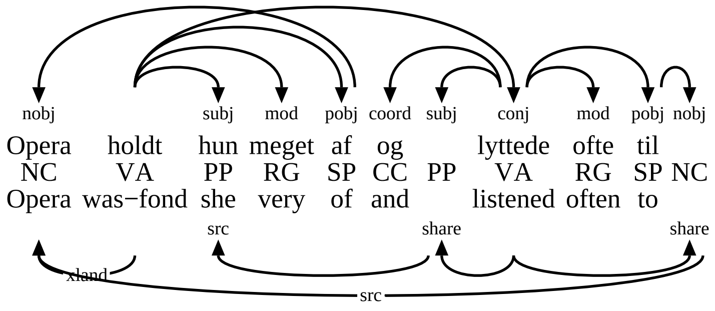

nobj
subj
mod
pobj
coord
subj
conj
mod
pobj
nobj
Opera holdt hun meget af og lyttede ofte til
NC VA PP RG SP CC PP VA RG SP NC
Opera was−fond she very of and listened often to
src
share
share
xland
src

Figure 4.27: Analysis of the peripheral sharing (31) with non-peripheral shared subject.

shared subject "hun" ("she") is placed between the heads of the two conjuncts. The DG analysis of this example is shown in Figure 4.27.

(31) Opera holdt hun meget af og lyttede ofte til.
Opera was-fond she very of and listened often to.
'She was very fond of opera and often listened to it.'

**Remark 4.36** To account for coordination and sharing in DG, we will assume that a coordination is headed by its first conjunct, and that the remaining conjuncts (called *secondary conjuncts*) attach themselves as conj adjuncts of the first conjunct; following Schachter (1977, p. 88) (cf. Hartmann 2000, p. 24), we will assume that coordinators are analyzed as coord adjuncts of the heads of secondary conjuncts.[5] Thus, in our analysis, every word is capable of attaching itself as a conj adjunct to a preceding word, if

[5]The Danish Dependency Treebank has a slightly different analysis of coordinations: in a coordination of the form "*X* and *Y*", DDT assumes that "and *Y*" forms a phrase that is headed by the coordinator "and", whereas we now analyze "and" as an adjunct of *Y*. The main reason for this change of analysis is that the coordinator is often optional (at least in Danish and English), and it is therefore inconvenient to give it a very prominent role in the dependency structure. For example, coordinations rarely have more than one coordinator, even when they have three or more conjuncts, and even in coordinations with only two conjuncts, it is often possible to omit the coordinator (eg, in sentential coordination). Rather than writing down two sets of rules — one for conjuncts with a coordinator and one for conjuncts without a coordinator — the new analysis makes it possible to use one rule to control how a word attaches itself as a secondary conjunct to another word, and a simple adjunct rule to control how the (possibly optional) coordinator attaches itself to the secondary conjunct.

the two conjuncts are sufficiently parallel with respect to word class, syntactic structure, and semantics.[6] The adjunct frame that licenses conj dependencies can be encoded by means of the following lexical specification at the word level:

$$\begin{array}{l}\text{type(word)}\\ -\!> \text{aframe(conjunct, modifier, conj} => [< \text{this]);}\end{array}$$

In our analysis, a shared dependent must be governed by some word within the first conjunct, and the head of the first conjunct must be contained in the shared dependent's extraction path (possibly as the shared dependent's landing site). The heads of all other conjuncts must create a filler with the shared dependent as the filler source; this filler is then used to establish the secondary dependency between the shared dependent and its secondary governors within the other conjuncts. This can be encoded in the DTAG lexicon by means of the filler frame shown below.

$$\begin{array}{l}\text{type(word)}\\ -\!> \text{fframe(sharing, simple\_filler, share, src,}\\ \qquad\quad \text{var}(\text{H}, [\xrightarrow{\text{conj}} \text{this}]) + (([\xleftarrow{\text{surface}} \text{var}(\text{H})] - [\xleftarrow{\text{conj}} \text{var}(\text{H})])\\ \qquad\qquad \mid \text{extract}(\text{node}([\xleftarrow{\text{any}-\text{conj}} \text{var}(\text{H})]))));\end{array}$$

The filler frame licenses the creation of a simple filler (with filler edge type share and source edge type src) at a node that has attached itself as a conj adjunct of the head $H$ of the first conjunct, provided the filler source is either a non-conjunct that lands on $H$, or has been extracted through one of the non-conj edges of $H$.

**Example 4.37** In Figure 4.25, 4.26, and 4.27, all shared dependents land on the head of the first conjunct. Figure 4.28 (an example due to Kahane 1997) shows our analysis of a peripheral coordination where one of the two shared dependents ("whose mother") has been extracted from the first conjunct. The example is complicated by the presence of fillers for relatives and control.

[6]Since parallelism is a matter of degree, it is better expressed as a cost function than as a hard constraint in the adjunct frame. We will therefore assume that the semantic component imposes a cost or probability that quantifies the semantic and pragmatic non-parallelism between the two conjuncts, and that the parallelism with respect to word class and syntactic structure is learned by a probabilistic language model, as described in chapter 6.

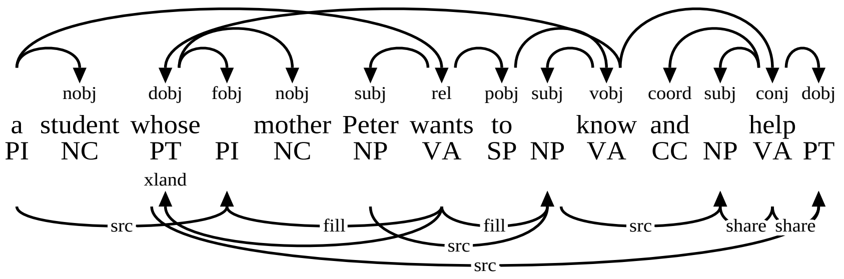


Figure 4.28: Analysis of peripheral sharing with an extracted shared dependent.

**Remark 4.38** Both coordination in general and sharing in particular are highly restricted phenomena, so in a rule-based lexicon we need a set of cost functions that can impose a cost whenever a restriction is violated. For example, a secondary conjunct should impose a cost if a non-shared dependent has been extracted out of the first conjunct, a restriction known as the *coordinate structure constraint* (encoded as the cost function coordinate_structure_constraint in the lexical entry below).

type(word)
$->$ cost(coordinate_structure_constraint,
    $100 * (\text{extract}(\text{node}([\xrightarrow{\text{conj}} \text{this}])) - [\xrightarrow{\text{src}} [\xleftarrow{\text{share}} \text{this}]]))$
$->$ cost(shared_by_sibling,
    $100 * ([\xrightarrow{\text{src}} [\xleftarrow{\text{share}} [\xleftarrow{\text{conj}} [\xrightarrow{\text{conj}} \text{this}]] - \text{this}\,]\,] - [\xrightarrow{\text{src}} [\xleftarrow{\text{share}} \text{this}]]))$
$->$ cost(left_shared_before_nonshared,
    $100 * (\ [\xleftarrow{\text{surface}-\text{filler}} [\xrightarrow{\text{conj}} \text{this}]] + [\xrightarrow{\text{src}} [\xleftarrow{\text{share}} \text{this}]] + [<\ [\xrightarrow{\text{conj}} \text{this}]]$
        $+ [>\ [\xleftarrow{\text{surface}-\text{filler}} [\xrightarrow{\text{conj}} \text{this}]] - [\xrightarrow{\text{src}} [\xleftarrow{\text{share}} \text{this}]]]]))$
$->$ cost(right_shared_after_nonshared,
    $100 * (\ [\xleftarrow{\text{surface}-\text{filler}} [\xrightarrow{\text{conj}} \text{this}]] + [\xrightarrow{\text{src}} [\xleftarrow{\text{share}} \text{this}]] + [>\ [\xrightarrow{\text{conj}} \text{this}]]$
        $+ [<\ [\xleftarrow{\text{surface}-\text{filler}} [\xrightarrow{\text{conj}} \text{this}]] - [\xrightarrow{\text{src}} [\xleftarrow{\text{share}} \text{this}]]]]));$

The cost function shared_by_sibling states that a secondary conjunct should also impose a cost if it fails to generate a filler for a dependent that has been used as a shared dependent by another secondary conjunct, ie, if the secondary conjuncts disagree about what the shared dependents are. Finally, the cost functions left_shared_after_nonshared and right_shared_before_nonshared state that a secondary conjunct should impose a cost if any shared dependent is non-peripheral, ie, the shared dependent lands on the head of the

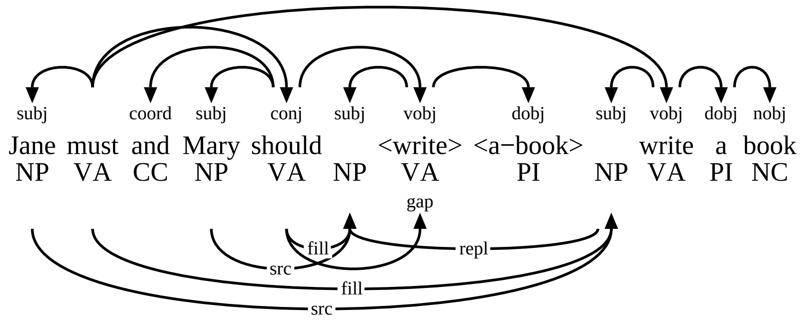

subj
coord
subj
conj
subj
vobj
dobj
subj
vobj
dobj
nobj
Jane must and Mary should <write> <a−book> write a book
NP VA CC NP VA NP VA PI NP VA PI NC
gap
fill
repl
src
fill
src

Figure 4.29: An analysis of the peripheral sharing (32) based on gapping fillers rather than simple fillers.

first conjunct and this landing site has a non-shared landed node that is more peripheral than the shared node.

**Remark 4.39** The account of peripheral sharing in this section may be somewhat simplistic, since it cannot handle examples where the shared dependent must receive different filler arguments from different conjuncts, as in (32) where the shared verbal object "write a book" receives the subject "Jane" from the first conjunct, and the subject "Mary" from the second conjunct. The example (33) from Hartmann (2000, p. 104) shows the same phenomenon in German.

(32) Jane must and Mary should write a book.

(33) Peter sollte es und Maria wollte es Hans erzählen.
Peter should it and Maria wanted it Hans to-tell
'Peter should tell it to Hans, and Maria wanted to tell it to Hans.'

It is possible to account for such examples by assuming that secondary conjuncts copy the secondary dependent by means of a gapping filler rather than a simple filler. In (32), the gapping mechanism allows the secondary conjunct to replace the default subject (a filler for "Jane") with the new subject (a filler for "Mary"). Figure 4.29 shows the resulting DG analysis.

## Gapping coordinations

Gapping is an important syntactic phenomenon which has been analyzed in a wide range of syntactic frameworks. For example, the large litera-

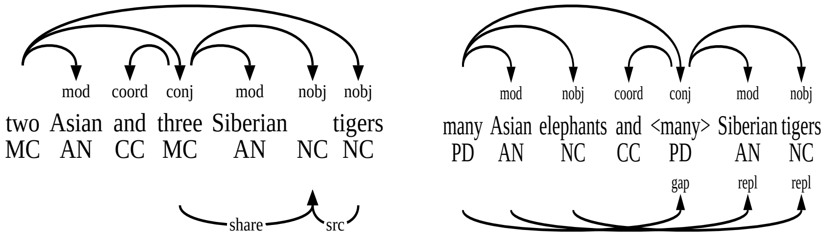


Figure 4.30: Two elliptic DP coordinations: peripheral sharing (left) and gapping (right).

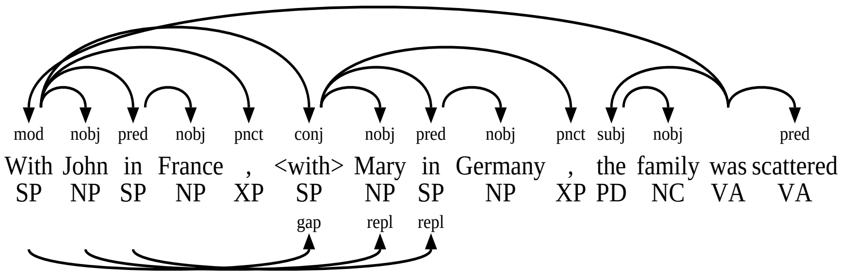

mod
nobj
pred
nobj
pnct
conj
nobj
pred
nobj
pnct
subj
nobj
pred
With John in France , <with> Mary in Germany , the family was scattered
SP NP SP NP XP SP NP SP NP XP PD NC VA VA
gap
repl
repl

Figure 4.31: PP gapping.

ture on gapping includes works within GB (Hartmann 2000; Johnson 2002), GPSG and HPSG (Gazdar 1981; Sag et al. 1985; Pollard and Sag 1994), Combinatory Categorial Grammar (Steedman 1990), Tree Adjoining Grammar (Sarkar and Joshi 1996), and dependency grammar (Melcuk 1988; Kahane 1997; Lombardo and Lesmo 1998; Hudson 2003; Hellwig 2003). In the following, we will show that the mechanisms for gapping coordinations that were introduced in section 2.5 can be used to provide a satisfactory account of gapping coordinations in Discontinuous Grammar.

**Example 4.40** Figure 4.30 shows two elliptic DP coordinations. In the peripheral sharing on the left, the phrase "tigers" is shared between the two conjuncts. In the gapping coordination on the right, a filler for the determiner "many" in the first conjunct functions as the head of the second conjunct. An example of PP gapping is shown in Figure 4.31.

**Remark 4.41** The gapping mechanisms presented in section 2.5 place very few restrictions on what can be elided in a gapping coordination. The only hard-coded restriction is that the head of the second conjuct is obligatorily elided, corresponding to the Finite-First Condition in (Hartmann 2000, p. 144). All other restrictions must be encoded probabilistically or by means of hand-written cost functions. For example, we can use cost functions at the gapping filler to control the word order of the gapping dependents, and to restrict the set of possible gapping dependents — for instance in order to encode the Major Constituent Condition proposed in (Hartmann 2000, p. 147), which roughly states that a gapping dependent must be a dependent of a sentence-heading verb. Likewise, we can use cost functions at the gapping dependent in order to ensure the presence of a replacement edge and compute a cost that quantifies the parallelism between the gapping dependent and the phrase that it replaces — eg, to encode the Maximal Contrast Principle proposed by (Hartmann 2000, p. 165), which states that the number of contrasting remnant-correspondent pairs are maximized in a gapping construction. Cost functions that encode general syntactic constraints (eg, on agreement, word order, obligatory presence or absence, selectional restrictions, island constraints, etc.) naturally also apply to gapping coordinations.

**Remark 4.42** It has been argued by many researchers that the path from the governor of the gapping dependent to the gapping filler is not allowed to contain any islands (cf. Hartmann 2000, p. 152). In DG, the gapping mechanism ensures that gapping dependents land on the gapping filler, which means that the nodes on the extraction path from the governor of the gapping dependent to the gapping filler will check whether the extraction violates any island constraints.[7] Thus, the normal island constraints can be used to explain the ungrammaticality of gapping coordinations such as (34), which violates the wh-island condition.

(34) *John asked what to write to Mary and Peter ~~asked what to write~~ to Sue. (Hartmann 2000, p. 152)

**Remark 4.43** Another consequence of our gapping mechanism is that the

[7]DG does not prevent grammar writers from writing island constraints that distinguish between gapping dependents and ordinary dependents, if they want to do so. For example, to account for examples like Figure 4.32, it might be reasonable to assume that in gapping coordinations, the coordinate structure constraint only prevents extraction out of dependents that have been replaced by gapping dependents.

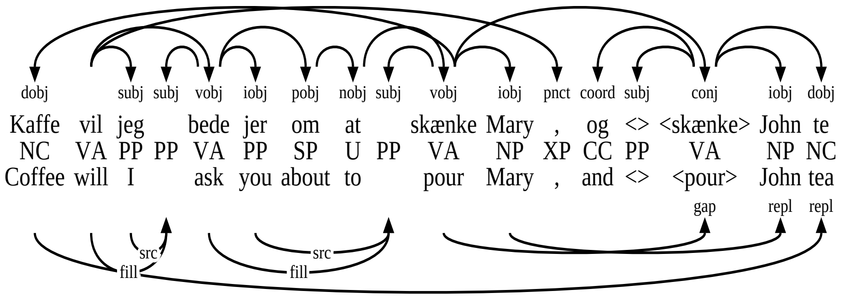

dobj
subj
subj
vobj
iobj
pobj
nobj
subj
vobj
iobj
pnct
coord
subj
conj
iobj
dobj
Kaffe vil jeg bede jer om at skænke Mary , og <> <skænke> John te
NC VA PP PP VA PP SP U PP VA NP XP CC PP VA NP NC
Coffee will I ask you about to pour Mary , and <> <pour> John tea
gap
repl
repl
src
fill
src
fill

Figure 4.32: Extraction out of the first conjunct in the gapping coordination (4.32).

word order of gapping dependents in secondary conjuncts is determined by word order cost functions associated with the gapping filler. That is, in DG, the word order within the gapped conjunct is determined in a very direct way by the gapping filler, and is not in any sense determined by the underlying word order within some reconstructed unelided version of the gapped conjunct, as in the elliptic analyses of Hartmann (2000) and many others.[8] This means that the word order of the gapped dependents is not necessarily parallel with the word order in the first conjunct. For example, consider the Danish gapping coordination (35), whose corresponding analysis is shown in Figure 4.32; in our judgment, it is grammatical but highly marked. In the first conjunct, the topicalized direct object "Kaffe" ("Coffee") precedes the indirect object "Mary", whereas in the second conjunct, the word order is reversed, ie, the indirect object precedes the direct object, which is the canonical word order in Danish.

(35) KAFFE vil jeg nu for syttende gang bede jer om at
COFFEE will I now for seventeenth time ask you about to
skænke MARY, og JOHN TE.
pour MARY, and JOHN TEA.
'I will now ask you, for the umpteenth time, to pour coffee to Mary and tea to John.'

The reversal of word order is almost obligatory, as shown by the the sentence (36) which is ungrammatical in our judgment, and identical to (35)

[8]In DG, all fillers are unordered, so DG analyses do not contain any information about the relative word order of the fillers for elided words.

except for the order of the direct and indirect object.

|  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (36) | * KAFFE | vil | jeg | nu | for | syttende | gang | bede | jer | om | at |
|  | COFFEE | will | I | now | for | seventeenth | time | ask | you | about | to |
|  | skænke | MARY, | og | TE | JOHN. |  |  |  |  |  |  |
|  | pour | MARY, | and | TEA | JOHN. |  |  |  |  |  |  |

The unelided version of the coordination in (36) is fully grammatical, so any ungrammaticality of the elided version of (36) is not caused by the lack of a grammatical unelided reconstruction of the gapped conjunct. This could be interpreted as weak evidence against the elliptic analysis of gapping coordinations. The ungrammaticality of (36) and the grammaticality of (35) follows if we assume that the gapping filler (which functions as the landing site of the gapping dependents) controls the word order, and stipulates that the gapping dependents must follow the canonical word order, ie, that the indirect object should precede the direct object.

**Example 4.44** An example may involve both gapping and sharing simultaneously, as shown by (37) below, where the verb "verlegt" ("lays") has been gapped from the second conjunct, and the noun phrase "den Putz" ("the plaster") has been shared between the two conjuncts. The corresponding DG analysis is shown in Figure 4.33.

|  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (37) | Karl | verlegt | die | Rohre | über | und | Peter | die | Kabel | unter | den |
|  | Karl | lays | the | tubes | above | and | Peter | the | cables | beneath | the |
|  | Putz. |  |  |  |  |  |  |  |  |  |  |
|  | plaster. |  |  |  |  |  |  |  |  |  |  |

In this section, we have shown how simple fillers and gapping fillers can be used to account for elliptic coordinations — including peripheral sharing, subject sharing, and gapping — and how the mechanisms for peripheral sharing and gapping interact with independently motivated cost functions, such as island constraints and word order constraints.

## 4.5 Anaphora: discourse structure

***Summary.*** *We argue that discourse structure can be viewed as a dependency structure for the words within an entire discourse, augmented with anaphoric edges that indicate the*

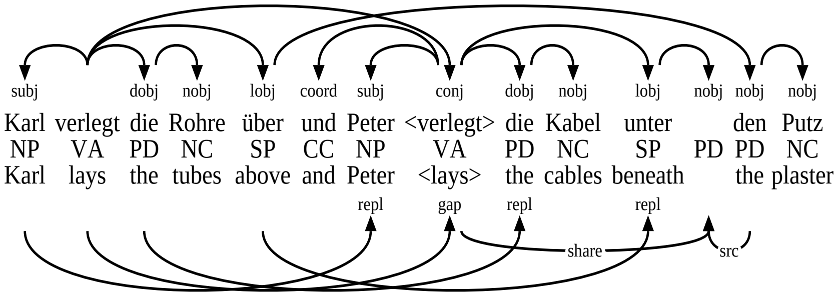

subj
dobj
nobj
lobj
coord
subj
conj
dobj
nobj
lobj
nobj
nobj
nobj
Karl verlegt die Rohre über und Peter <verlegt> die Kabel unter den Putz
NP VA PD NC SP CC NP VA PD NC SP PD PD NC
Karl lays the tubes above and Peter <lays> the cables beneath the plaster
repl
gap
repl
repl
share
src

Figure 4.33: The coordination (37) which involves both gapping and peripheral sharing.

*antecedents for each anaphor. We discuss the relationship between our proposal and the literature on discourse structure.*[9]

Historically, syntax has mainly been concerned with the linguistic relations that hold within sentences. However, in the last few years, linguists have become increasingly interested in discourse phenomena, ie, linguistic relations that hold across sentences. Prominent examples include work based on Rhetorical Structure Theory (Mann and Thompson 1988) such as the English RST treebank (Carlson and Marcu 2001; Carlson et al. 2003) and RST-based work on discourse parsing and summarization (Marcu 2000); work on Segmented Discourse Representation Theory (Asher forthcoming; Lascarides and Asher in press; Baldridge and Lascarides 2005); work on the Linguistic Discourse Model (Polanyi 2003; Polanyi et al. 2003, 2004); and work on the Penn Discourse Treebank (Webber et al. 2003; Webber 2004). In this section, we will sketch how Discontinuous Grammar can be extended to a theory of discourse by viewing discourse structure as a dependency structure augmented with anaphoric edges.

There are basically two ways of accounting for discourse structure. The first approach is to assume two separate levels of analysis, a discourse structure connecting discourse segments and a syntactic structure describing the internal structure within each discourse segment, with some kind of interface linking the two levels. This is essentially the approach taken

[9]This section incorporates insights proposed by my colleagues Daniel Hardt, Bjarne Ørsnes, Peter Juel Henrichsen, Peter Rossen Skadhauge, and Hans Dybkjær during our discussions in the discourse reading group at the Copenhagen Business School.

by the English RST treebank (Carlson et al. 2003), the Penn Discourse Treebank (Webber 2004), and the Linguistic Discourse Model (Polanyi et al. 2003). The second approach, as noted by Webber (2004, p. 755), is to treat discourse relations as lexically anchored dependencies within the syntax — ie, to assume that all words in the discourse are linked into one large dependency structure, using the same mechanisms as in the analysis of intersentential structure. This is not the solution Webber adopts,[10] but it has the obvious advantage that it gives an integrated account of syntax and discourse, thereby avoiding what we view as the central weakness in the two-level approach: the difficulty of drawing a natural boundary between syntax and discourse.

**Example 4.45** Carlson and Marcu (2001, p. 2) have pointed out how difficult it is to give a theoretically satisfactory definition of the notion of elementary discourse unit because the "boundary between discourse and syntax can be very blurry". They illustrate this claim with the examples (38a)–(38d) below (adapted from Carlson and Marcu 2001, p. 2).

(38) a. Xerox's income grew 6.2%. This earned mixed reviews.
b. Xerox's income grew 6.2%, which earned mixed reviews.
c. Xerox's income grew 6.2%, earning mixed reviews.
d. The 6.2% growth of Xerox earned mixed reviews.

Carlson and Marcu note that although (38a)–(38d) are nearly identical with respect to the information they convey, the relation between the two events is expressed by a discourse relation in (38a) and by a syntactic relation in (38d), whereas it seems to be a matter of taste how to classify the relations in (38b) and (38c). In their segmentation scheme, they have decided to segment (38a)–(38c) into two elementary discourse units, and (38d) into only one elementary discourse unit, but they view this as a somewhat arbitrary decision motivated by practical rather than theoretical concerns.

[10] Webber writes: "Such a radical step would not be impossible. However, we have not thought through its many consequences in detail, given that one would not want to lose the generalizations that have been captured over many years of work in lexicalized sentence-level grammars." (Webber 2004, p. 755)

**Example 4.46** Dependency roles can be satisfied by phrases that span several sentences. For example, in (39) below, the phrase "I am tired. Go home" acts as the quotational object of the verb "said".

(39) He said "I am tired. Go home," in a sleepy voice.

The construction has a high frequency in Danish: 50% of the 297 quotational objects in the Danish Dependency Treebank with quotation marks span more than one sentence. The quotation marks are by no means obligatory, and the Danish Dependency Treebank contains many examples of the constructions shown in (40) and (41).

(40) I am tired, he said in a sleepy voice. Go home.

(41) I am tired. Go home, he said in a sleepy voice.

This phenomenon has also been observed by Webber (2004, p. 766), who notes that in (42), the sentential complement of the verb "believed" is an entire discourse which includes the second sentence "Rather, it arose...".

(42) Epigenesists believed that the organism was not yet formed in the fertilized egg. Rather, it arose as a consequence of profound changes in shape and form during the course of embryogenesis.

Examples 4.45 and 4.46 show that arbitrarily complex spans of discourse can function as syntactic units, and that it is in general difficult to maintain a theoretically satisfying distinction between syntactic units and discourse units. In our view, the most natural explanation is that there is no fundamental difference between discourse units and syntactic units, and that it is a historical coincidence that we refer to small units as syntactic units, and to larger units as discourse units. We will therefore assume that a discourse structure is just a syntax graph that spans an entire discourse — ie, in our setting, it is a DG graph that consists of a deep tree that encodes dependencies and a surface tree that encodes word order, augmented by secondary edges that encode filler sources and antecedents for anaphora.

**Example 4.47** Figure 4.34 shows a possible dependency analysis of (39). The quotation "I am tired. Go home" is analyzed as a `qobj` (quotational object) of "said". The phrase "Go home" is analyzed as a `res` (result) adjunct of "I am tired", thereby indicating that the speaker urges the hearer to go home because the speaker is tired (cf. Asher 1999; Baldridge and Lascarides 2005). The `qobj` dependency is licensed by a complement frame

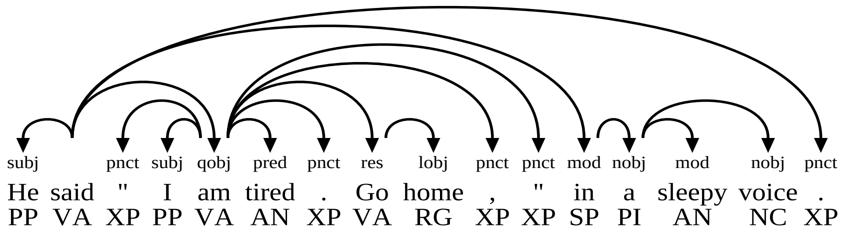

subj
pnct
subj
qobj
pred
pnct
res
lobj
pnct
pnct
mod
nobj
mod
nobj
pnct
He said " I am tired . Go home , " in a sleepy voice .
PP VA XP PP VA AN XP VA RG XP XP SP PI AN NC XP

Figure 4.34: Dependency analysis of the discourse (39).

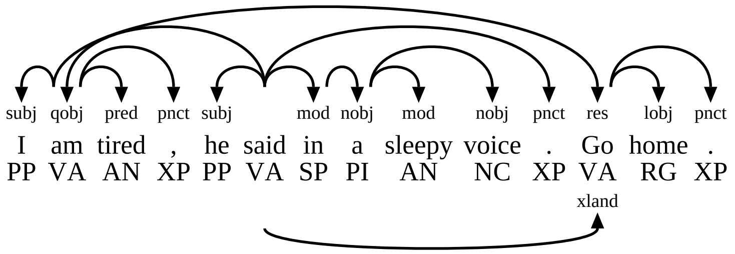

subj
qobj
pred
pnct
subj
mod
nobj
mod
nobj
pnct
res
lobj
pnct
I am tired , he said in a sleepy voice . Go home .
PP VA AN XP PP VA SP PI AN NC XP VA RG XP
xland

Figure 4.35: Discontinuous dependency analysis of the discourse (40).

associated with the lexeme for "said", and the res dependency is licensed by an adjunct frame associated with the lexeme "Go". The functors and modifiers associated with these complement and adjunct frames specify how the semantic representation for the combined phrases are computed.

The dependency structure is maintained in the analysis of (40) shown in Figure 4.35, but the change in word order means that the res dependency becomes discontinuous. The landing site for "Go home" is indicated by means of the xland edge.

**Example 4.48** Figure 4.36 shows a dependency analysis of a small discourse taken from Asher (forthcoming, p. 6), using the SDRT discourse relation names proposed by Baldridge and Lascarides (2005). The discourse dependencies are indicated with dashed arrows. In SDRT, an elab (elaboration) relation indicates that the adjunct elaborates on some aspect of the governor, and a narr (narration) relation indicates that the adjunct event occurs after the governor event. Thus, the analysis shows that "He ate salmon. He devoured lots of cheese" is an elaboration of "He had a fantastic meal", and that the event described by "He devoured lots of cheese" occurs after the event described by "He ate salmon".

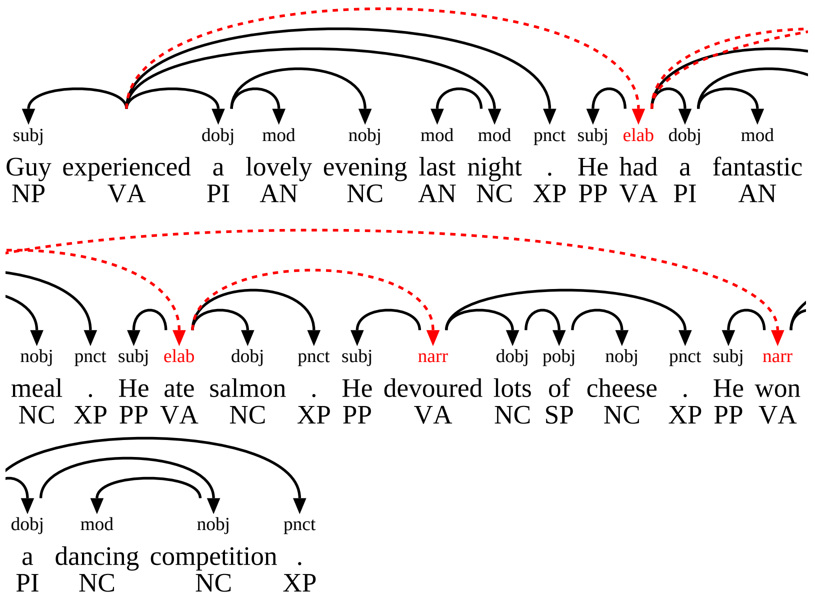

subj dobj mod nobj mod mod pnct subj elab dobj mod
Guy experienced a lovely evening last night . He had a fantastic
NP VA PI AN NC AN NC XP PP VA PI AN
nobj pnct subj elab dobj pnct subj narr dobj pobj nobj pnct subj narr
meal . He ate salmon . He devoured lots of cheese . He won
NC XP PP VA NC XP PP VA NC SP NC XP PP VA
dobj mod nobj pnct
a dancing competition .
PI NC NC XP

Figure 4.36: Dependency analysis of a small discourse.

**Remark 4.49** In the RST annotation scheme proposed by Carlson and Marcu (2001) the notions of "nucleus" and "satellite" closely correspond to the notions of "governor" and "dependent" in dependency theory. This means that RST can largely be viewed as a dependency-based theory. There are some slight incompatibilites, though. For example, RST allows multi-nuclear relations, ie, relations involving two or more heads. But these are not completely symmetric from a dependency theoretic point of view — eg, in coordinations, the first conjunct has different extraction properties compared with secondary conjuncts, and many of the other multi-nuclear relations in RST connect a sentence with an adverbial clause, which would be analyzed as a dependent in most dependency theories.

**Remark 4.50** Is discourse structure a tree structure or a general graph structure? This is perhaps one of the most hotly debated issues in the theory of discourse. Wolf and Gibson (2003) argue that tree structures are a poor representation of discourse structure, but Marcu (2003) defends the use of tree structures, arguing that many of the additional edges proposed by Wolf and Gibson are not discourse edges, but coreference edges.

Most other discourse frameworks have used a tree-based representation as well. For example, Mann and Thompson (1988) require discourse units to be organized in a hierarchical tree structure, although they merely view the tree structure as a convenient methodology for analyzing texts, without any psychological validity. Tree structures are also used to model discourse in the Linguistic Discourse Model (Polanyi 2003), where the discourse structure is licensed by a Context-Free Grammar, and in the English Discourse Treebank (Webber et al. 2003), where the discourse structure is licensed by a Tree-Adjoining Grammar.[11] Within Segmented Discourse Representation Theory, Asher (forthcoming) has argued that there are certain phenomena where a tree structure fails to capture all relevant relations, but Baldridge and Lascarides (2005) have argued that SDRT discourse structure can be viewed as a tree structure augmented by secondary relations.

From our point of view, the discourse tree proposed within these theories corresponds to the dependency tree within our DG analysis — ie, a DG analysis consists of a discourse tree as proposed in these theories, augmented by a surface tree and anaphoric links. Consequently, apart from the elimination of a separate level of discourse, our DG analysis is by no means a radical departure from existing theories of discourse.

**Remark 4.51** Asher (forthcoming) convincingly argues that discourse structure is primarily determined by semantics, and that cue words and sentence-internal syntax only play a minor role. This view, which we fully support, is by no means incompatible with a dependency-based view of dis-

[11]Webber et al. (2003, p. 546) write that they "are not committed to trees as the limiting case of discourse structure". In particular, they assume that the discourse (1) below is a multi-parent structure where (a) and (b) stand in a succession relation to each other, whereas (b) and (c) stand in a manner relation to each other.

(1) (a) The first to do that were the German jewellers. (b) And Morris followed very quickly after, (c) using a lacquetry technique to make the brooch.

However, we disagree with this analysis, and note that if (c) is analyzed as a manner adverbial of (b), then we get a tree structure if the discourse unit (b)+(c) is analyzed as a succession adjunct of (a). The analysis is shown in the dependency tree below.

$$\begin{tikzcd}
(a) \arrow[r, bend left, "\text{succession}"'] & (b) \arrow[r, bend left, "\text{manner}"'] & (c)
\end{tikzcd}$$

course: dependencies encode semantics as much as syntax, and given a modifier scope, the functors and modifiers within the complement and adjunct frames associated with lexemes in the dependency tree uniquely determine the compositional semantics of all phrases in the graph.

Current theories of semantics are far more fragmentary than current theories of syntax, and in particular, their coverage is not large enough to provide detailed semantic analyses for all the sentences in an arbitrary text. For this reason, DG is not committed to a particular semantic theory, but simply provides an interface to semantics by allowing the semantic component (whatever it is) to contribute to the total cost of an analysis. From this perspective, Asher has argued that in discourse relations, the semantic cost contribution is more important than the syntactic cost contribution. Discourse relations are not unique in this respect, since many other adjunct relations — in particular the relations associated with adverbial expressions of time, place, manner, cause, etc. — are also more semantic than syntactic in nature.

**Example 4.52** Webber et al. (2003) convincingly argue that anaphora are involved in the interpretation of discourse adverbials such as "then", "instead", "otherwise", "therefore", etc. — ie, discourse adverbials take two arguments where one argument is provided by the governor of the discourse adverbial, and the other argument is resolved anaphorically. In their analysis, an anaphor has an antecedent which determines the interpretation of the anaphor as being either coreferential to the antecedent, or to some discourse referent that can be inferred from the antecedent by means of a relation provided by either the antecedent, the anaphor, or both (eg, a part-whole relation, a generic relation, etc.).

We will adopt this analysis in DG, ie, we will assume that discourse adverbials are anaphora acting as adjuncts, whose antecedents are annotated by means of an ant (antecedent) edge from the antecedent to the anaphor. Figure 4.37 shows the DG analysis of an example taken from Webber et al. (2003, p. 553). The word "go" is analyzed as a contr (contrast) adjunct of "stop", and the discourse anaphor "otherwise" is analyzed as an adjunct of "go" with the antecedent phrase "the light is red".

In sentences such as "Sue lifted the receiver as Tom darted to the other phone", Webber et al. (2003, p. 555) analyze the word "other" as an anaphor with antecedent "the receiver" which refers to a discourse referent that can be inferred by means of the part-whole relation (receiver-phone) provided

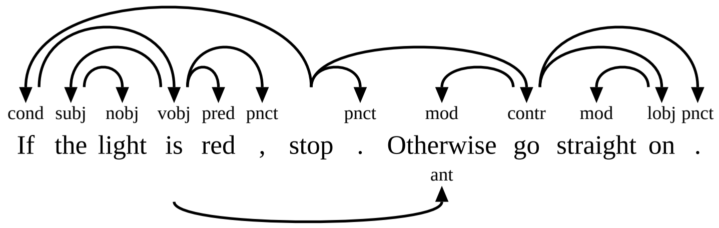


Figure 4.37: DG analysis of the discourse adverbial "otherwise".

by the antecedent and the relation provided by the anaphor "other" (ie, being a phone in the context that is not identical to the inferred phone). However, Webber et al. do not maintain this analysis in their analysis of the construction "on the one hand ... on the other hand". Instead of analyzing "other" as a discourse adverbial, they analyze "on the one hand $X$, on the other hand $Y$" as an idiomatic expression that takes two complements $X$ and $Y$. This analysis is incompatible with the standard syntactic analysis of the phrases "on the one hand" and "on the other hand" as adjuncts that can be very far apart in the dependency structure.

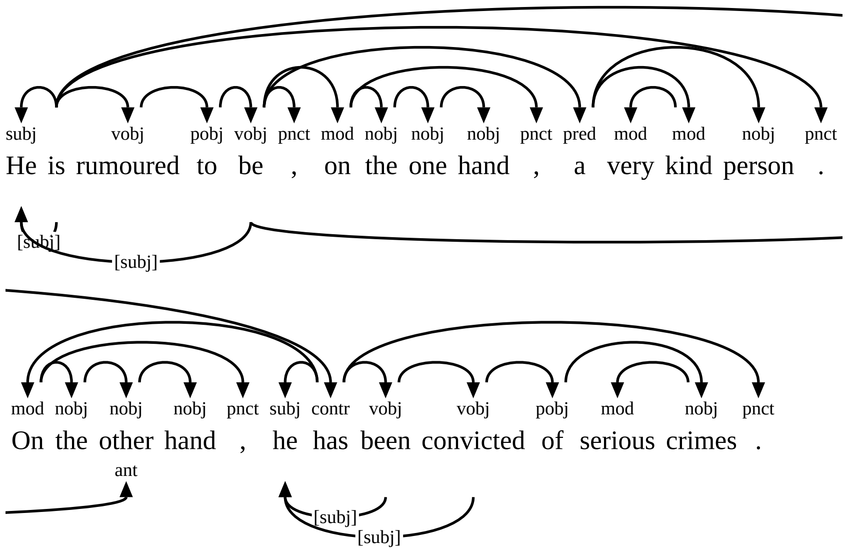
In the DG analysis of (43) shown in Figure 4.38, we will therefore assume that "other" acts as a discourse adverbial with antecedent phrase "(He) be a very kind person" (the attachment point of "on the one hand"), ie, the phrase "on the other hand" can be paraphrased as "arguing against his being a very kind person".

(43) He is said to be, on the one hand, a very kind person. On the other hand, he has been convicted of serious crimes.

The phrase "on the one hand" is optional. It constrains where the antecedent for "on the other hand" is situated, and cannot be used without a discourse adverbial like "on the other hand" which signals contrast.

To understand how discourse adverbials are handled in DG, we need to look briefly at the mechanisms for anaphora in DG, even though anaphora are outside the scope of this dissertation.

**Remark 4.53** We will assume that the antecedent edges connecting an anaphor with its antecedents are licensed by anaphor frames associated with the anaphoric lexeme. Given an antecedent edge, the shortest path within



subj vobj pobj vobj pnct mod nobj nobj nobj pnct pred mod mod nobj pnct
He is rumoured to be , on the one hand , a very kind person .
[subj]
[subj]
mod nobj nobj nobj pnct subj contr vobj vobj pobj mod nobj pnct
On the other hand , he has been convicted of serious crimes .
ant
[subj]
[subj]

Figure 4.38: DG analysis of the construction "on the one hand ... on the other hand".

the deep tree from the anaphor to the antecedent is called the *anaphoric extraction path* for the antecedent edge. To account for structural constraints on antecedents, we must modify the extract operator so that it returns all antecedent edges whose extraction path contains a given node; in this way, nodes on the extraction path can function as islands for anaphoric extraction. Unlike normal extraction paths, which are constrained by the upwards movement principle and therefore only contain upwards edges, anaphoric extraction paths in general consist of a series of upwards edges followed by a series of downwards edges. In Figure 4.39, the anaphoric extraction path for the discourse adverbial "otherwise" in Figure 4.37 is indicated with dotted lines.

With these mechanisms, it is possible to encode the important *right frontier constraint* (cf. Asher 1993) which states that the antecedent of an anaphor must be located at the right frontier associated with the anaphor, ie, anaphoric extraction paths are not allowed to contain any downwards discourse edges. As noted by Asher (1993) and Webber et al. (1999), there are exceptions to this rule, but only with some anaphora. Since our extract operator gives access to both the anaphor and its antecedent, it al-

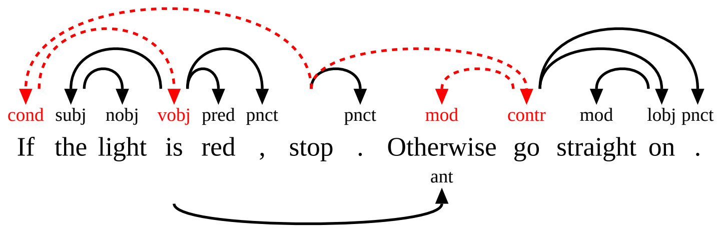


Figure 4.39: The anaphoric extraction path for the discourse adverbial "otherwise" from Figure 4.37.

lows an island to be sensitive to the properties of both the anaphor and the antecedent, ie, our island mechanism may well be powerful enough to account for both the right frontier constraint and its exceptions.

**Example 4.54** In Example 4.52, we saw that the adjunct "on the one hand" is ungrammatical in the absence of a contrastive discourse adverbial like "on the other hand". In our DG analysis, we will assume that "on the one hand" can attach itself as an adjunct to any verbal governor, and that it is equipped with a cost function that triggers a cost if the verbal governor is not a downwards node on the anaphoric extraction path of some contrastive discourse adverbial such as "on the other hand". During incremental processing, the parser will try to minimize the total cost of the graph, ie, it will try to eliminate the cost imposed by "on the one hand" by finding an antecedent for "on the other hand" that has the verbal governor of "on the one hand" on its anaphoric extraction path. In this way, "on the one hand" can be used to constrain the possible antecedents for discourse adverbials such as "on the other hand".

**Remark 4.55** The purpose of the discourse structure is to determine the compositional semantics of the entire discourse. In many cases, two different discourse structures may well lead to the same compositional semantics, ie, there may be several correct analyses of a discourse: if two discourse analyses have the same (or nearly the same) semantic interpretation, the choice between them is of little relevance to speakers, and both analyses can be considered correct.

**Remark 4.56** So far, we have focused on the analysis of written language, but it is also important to consider spoken language, in particular spoken

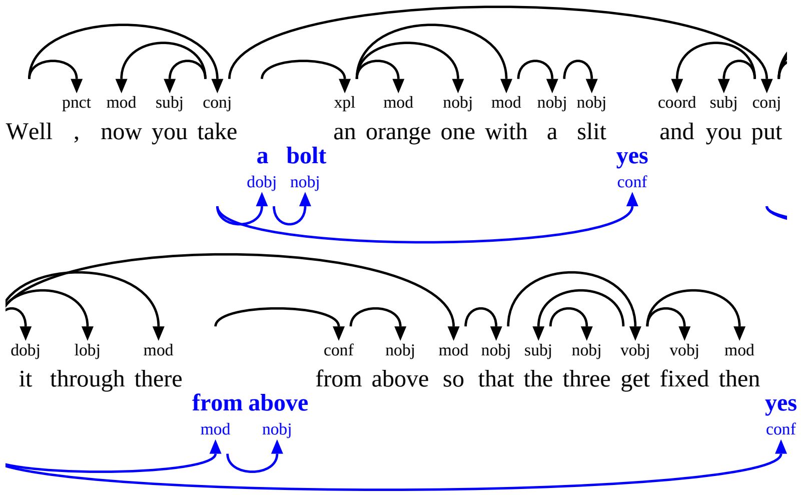


Figure 4.40: Analysis of spoken language dialog.

dialogue with interactions and overlaps between the speakers. Figure 4.40 shows a dependency analysis of a dialogue from the Switchboard corpus. The dialogue is interesting because the speakers interact (and overlap, although this cannot be shown directly in the graph). For example, speaker B provides the direct object "a bolt" for the verb "take" uttered by speaker A, and A then elaborates on the direct object provided by B. Although spoken language obviously contains many phenomena that are rarely observed in written language, and despite the fact that we have not yet tested our ideas on a large corpus, it seems that a dependency analysis which includes both syntactic relations and discourse relations may well be powerful enough to capture most of what is going on in spoken language.

Figure 4.41 shows a dependency analysis of speech repairs, another important phenomenon in spoken language (cf. Meteer et al. 1995; Sampson 1998). In our analysis, which builds on the ideas presented in (Meteer et al. 1995), the speech repair consists of a governor having one or more *reparandum adjuncts* with edge type rep containing the repaired material, an optional *editing adjunct* (or *interregnum*) with edge type edit that signals the presence of a speech repair, and one or more *repair dependents* containing the material that replaces the reparandum. Reparandum adjuncts are viewed as anaphora that take the repair dependent as their repl (re-

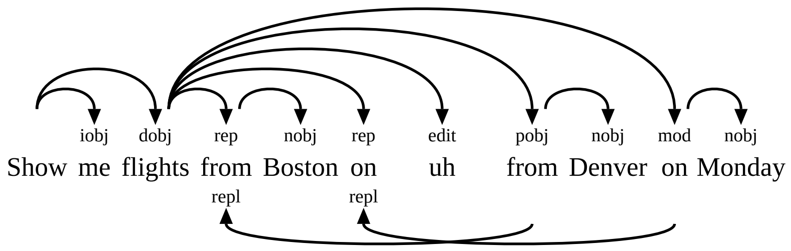


Figure 4.41: Dependency analysis of a speech repair taken from Meteer et al. (1995, p. 23).

placement) antecedent. In the compositional semantics, the reparandum phrases are ignored, although material within the reparandum can function as antecedents for pronouns outside the reparandum.

In this section, we have argued that discourse structure is a dependency tree augmented with a surface tree and a set of anaphoric links, ie, the dependency analysis for an entire discourse is created by linking the dependency analyses of individual sentences by means of a set of discourse dependencies. The dependency tree determines the functor-argument tree and hence the compositional semantics for the entire discourse, ie, how the semantic interpretations of the individual sentences are combined into a semantic interpretation for the entire discourse. The surface tree controls the word order of the entire discourse. The dependency tree and the surface tree also restrict the possible antecedents for anaphora by means of the right frontier constraint. Thus, in our view, we need exactly the same set of mechanisms for accounting for both discourse and sentence-internal syntax.

## 4.6 Periphery: punctuation

***Summary***. *We outline how DG can be used to account for punctuation. After presenting a classification of the different punctuation marks, we describe the principles behind our analysis of punctuation phenomena in DG. We then describe how cost functions can be used to specify where punctuation marks cannot appear and where they must appear; how anaphora and cost functions can be used to control how brackets are matched without any crossing; and how we can account for quote transpositions by assuming that end quotes can be moved under very special circumstances.*

There is relatively little work on punctuation within formal linguistics. Punctuation has been investigated from a linguistic perspective by Nunberg (1990) and Jones (1997), and from a psycholinguistic perspective by Hill and Murray (2000). There have also been some practically motivated attempts to deal with punctuation by computational linguists; a survey of these is given by Say and Akman (1997). However, despite these efforts, most syntactic theories have simply ignored the issue of punctuation. This is unfortunate because punctuation is a frequent and highly complicated phenomenon that stretches the descriptive power of any syntax formalism to its limit. In this section, we will describe how DG accounts for punctuation, and why it is a challenging phenomenon. The section incorporates material from Kromann et al. (2003).

Following Nunberg (1990), we use the following terminology to refer to punctuation marks.

**Definition 4.57** Commas, dashes, semicolons, colons, and periods are called *point marks*; paired symmetrical delimiters such as parentheses and quotation marks are called *bracket marks*; and question marks and exclamation marks are called *tone indicators*. All of these marks are collectively referred to as *punctuation marks*.

In our theory of punctuation, we classify punctuation marks along two dimensions: how they are placed relative to the preceding and following word, and whether they are paired with another mark.

**Definition 4.58** A *begin-mark* attaches as a clitic to a following word or phrase without any intervening spaces (ie, as a proclitic). An *end-mark* attaches as a clitic to a preceding word or phrase without any intervening spaces (ie, as an enclitic). A *separator* is surrounded by spaces.

**Definition 4.59** A mark is called *binary* if it is obligatorily paired with another mark, and *unary* if it is unpaired.

**Example 4.60** Examples of the different kinds of marks are given in Figure 4.42. Note that some punctuation marks can be used as more than one kind of mark. For example, in the ASCII character set, the same single quote is used as both begin-mark and end-mark (eg, "they 'persuaded' her"),

| | begin-marks | end-marks | separators |
|---|---|---|---|
| unary | - | ? ! . , ; : - | — |
| binary | ( [ { < ‘ “ | ) ] } > ’ ” | |

Figure 4.42: Examples of the different kinds of marks.

and a dash can be used as all three kinds of mark; eg, end-mark in coordinations in Danish such as “han- og hunkøn” (“masculine and feminine gender”), begin-mark in coordinations in Danish such as “jernaldermand og -kvinde” (“iron age man and woman”), and separator in parentheticals such as “Bohr — a famous physicist — was born in Copenhagen”.

**Remark 4.61** Within a dependency framework, punctuation marks are naturally viewed as independent lexemes that can attach to other lexemes as either complements or adjuncts. The adjunct analysis is nearly always preferable, for in the complement analysis, punctuation marks must be encoded in all complement frames — an awkward move if we want to use the same grammar for both spoken and written language, make the grammar robust against errors of punctuation, and avoid redundancy in the grammar. For these reasons, we will assume that punctuation marks attach themselves as adjuncts to other words, and that they cannot take any words as their complements or adjuncts.

**Remark 4.62** In our linguistic analysis, we will assume that a begin-mark always precedes its adjunct governor, that an end-mark always succeeds its governor, and that a separator can either precede or succed its governor. Except for ending quotation marks in quote transpositions (cf. Remark 4.77), we will assume that marks can never be extracted, ie, the governor and landing site of a mark must always coincide.

With these assumptions, we can find the adjunct governor associated with a given mark by looking at all possible landing sites in the direction associated with the mark (left for end-marks, right for begin-marks, and both directions for separators), and determine which one of the possible landing sites functions as the adjunct governor of the mark; anything else being equal, we will prefer the highest attachment point. We will assume that binary marks are always attached to the lowest governor that dominates all words between the begin- and end-mark, which makes binary marks even more constrained than unary marks.

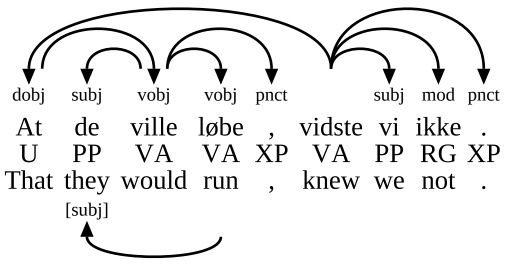

dobj
subj
vobj
vobj
pnct
subj
mod
pnct
At de ville løbe , vidste vi ikke .
U PP VA VA XP VA PP RG XP
That they would run , knew we not .
[subj]

Figure 4.43: Analysis of punctuation in Danish.

**Example 4.63** The comma in Figure 4.43 is an end-mark, so it must attach to a possible landing site on the left (ie, one of the words "run", "would", or "that"). The comma rules for Danish mandate a comma (or a higher mark) at the end of a verb phrase with an overt subject, so the natural adjunct governor for the comma is the finite verb "would". The period is an end-marker which must be attached to a landing site on the left, ie, one of the words "not" or "knew". Since the Danish punctuation rules stipulate that periods mark the end of a sentence, the word "knew" is the most natural choice of governor.

**Example 4.64** In the analysis of (44) shown in Figure 4.44, the two first quotation marks enclose the partial phrase "Are you," whose lowest governor is "Are", so the two first quotation marks are analyzed as adjuncts of "Are". Similarly, the two last quotation marks are also analyzed as adjuncts of "Are" because they enclose the partial phrase "entirely ok now?" whose lowest governor is "Are" as well. Two matching begin- and end-marks are indicated by means of a match edge, as described in Remark 4.76.

(44) "Are you," he said, "entirely ok now?"

So far, we have introduced a set of general principles that guide our dependency analysis of punctuation. We will now address the question of how to formulate rules that can be used to account for the punctuation marks used in languages like Danish or English.

**Remark 4.65** In our rule-based analysis of punctuation, we will assume that punctuation is controlled by means of six different kinds of rules: (i) *adjunct rules* that specify where a punctuation mark can attach itself as an

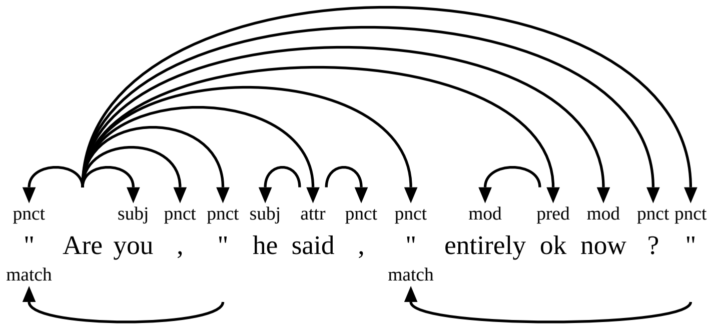


Figure 4.44: Analysis of binary punctuation.

adjunct; (ii) *realization rules* that impose a cost if an obligatory punctuation mark fails to be realized; (iii) *word order rules* that impose a cost if a punctuation mark is placed incorrectly in the linear order; (iv) *adjacency rules* that impose a cost on illegal sequences of punctuation marks; (v) *bracket rules* that impose a cost for unmatched bracket marks; and (vi) *movement rules* that impose a cost on illegal movements of punctuation marks. Since all of these rules are tied to specific types in the inheritance hierarchy, different lexemes can have slightly different sets of punctuation rules. The rules will be described in more detail in the following remarks.

**Remark 4.66 (adjunct rules)** Each punctuation lexeme is equipped with a set of adjunct frames that allow the punctuation lexeme to attach itself as an adjunct to other nodes. Adjunct frames only specify where a punctuation lexeme can appear — they do not ensure that the punctuation adjunct is obligatory or prevent multiple punctuation adjuncts (eg, two adjacent commas) from attaching themselves to the same adjunct governor; this must be achieved by means of realization rules and adjacency rules.

**Example 4.67** Figure 4.45 shows a set of adjunct frames that license the most frequent kinds of punctuation in Danish and English. Following the practice in the Danish Dependency Treebank, the same edge type pnct has been used in all the adjunct frames, although nothing in DG prevents us from using different edge types for different kinds of punctuation.

**Example 4.68** Figure 4.46 shows an analysis of (45) using the adjunct rules in Figure 4.45. The punctuation mark and the begin and end quotes attach

| **Punctuation mark** | **Adjunct governor** | **Adjunct edge type** |
|---|---|---|
| . ? ! ; | head of main sentence | pnct |
| : | governor of xpl or qobj phrase | pnct |
| , | finite verb with overt subject | pnct |
| , ; | head of first conjunct | pnct |
| , | head of topicalized phrase | pnct |
| , – | head/governor of parenthetical phrase | pnct |
| ( ) | head of parenthetical adjunct | pnct |
| " ' " ' | any | pnct |

Figure 4.45: The most important adjunct rules governing punctuation.

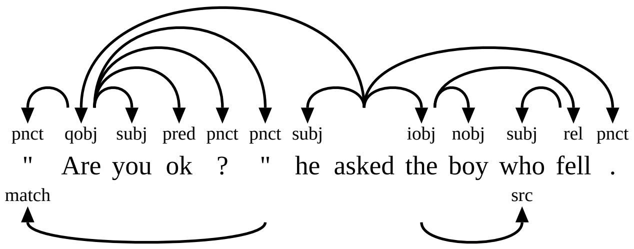


Figure 4.46: DG analysis of punctuation.

to the head of the quoted sentence "Are you ok", and the period attaches to the head of the main sentence.

(45) "Are you ok?" he asked the boy who fell.

**Definition 4.69** The *punctuation periphery* at a position $p$ in a phrase with head $h$ is defined recursively as the set of all punctuation marks that are either adjacent to $p$, or adjacent to some punctuation mark in the punctuation periphery of $h$ at $p$. The *inner punctuation periphery* of $h$ at $p$ is defined as the set of all punctuation marks in the punctuation periphery of $h$ at $p$ that are deeply dominated by $h$, and the *outer punctuation periphery* is defined as the set of all punctuation marks in the punctuation periphery that are not in the inside punctuation periphery.

**Remark 4.70 (realization rules)** Adjunct rules specify where a punctuation mark *can* attach to a given governor, but they do not specify that a

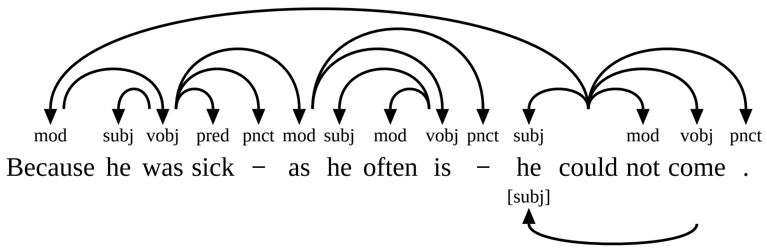


Figure 4.47: Analysis of example where marks must be shared.

punctuation mark *must* attach. To specify that some punctuation mark is obligatory at a given position in a given phrase, the head lexeme must therefore be equipped with a cost function that checks whether the required punctuation mark is realized there and produces a cost if it is not.

Realization rules are complicated by the fact that several heads can share the same punctuation mark, ie, the same punctuation mark can satisfy the realization requirements of different heads.[12] For example, in (46) whose analysis is shown in Figure 4.47, Danish punctuation rules require a comma (or a more prominent mark) at the end of the phrases headed by "was" and "is" because they are finite verbs with an overt subject. This requirement is satisfied by the second dash, which is not an adjunct of either "was" or "is", but which is nevertheless contained in the right punctuation periphery of both phrases (cf. Definition 2.73).

(46) Because he was sick — as he often is — he could not come.

In general, a requirement for a comma at the end of a phrase can be satisfied by any comma, dash, semicolon, colon, period, question mark, or exclamation mark at the right punctuation periphery of the phrase, or by any end-bracket which is either a local adjunct or located in the outer right punctuation periphery — ie, an end-bracket in the inner punctuation pe-

[12]Nunberg (1990) refers to this phenomenon as *point absorption* and *bracket absorption*, since he adopts a transformational view where the observed punctuation is generated from a base representation where each phrase contains all the punctuation marks that it requires. A set of absorption rules then specify how punctuation marks with high rank can absorb adjacent punctuation marks with identical or lower rank, according to a hierarchy where commas have lowest rank, and periods highest rank. His insights are retained in our analysis, but from a non-transformational point of view (sharing) rather than a transformational point of view (absorption).

| **Unary mark** | , | – | ; | : | . | ? | ! |
|---|---|---|---|---|---|---|---|
| **Unary substitutes** | , – ; : . ? ! | ; : . ? ! | | | . ? ! | ? | ! |
| **Binary substitutes** | ) ] ″ ′ | ) ] ″ ′ | | | ) ] ″ ′ | | |

Figure 4.48: Possible substitutes for the unary punctuation marks.

riphery cannot meet the requirement. Requirements for other unary marks can be satisfied by other marks in the same way. Figure 4.48 shows the possible substitutes for each unary mark. As with commas, binary substitutes are only valid if they are local adjuncts or located within the outside punctuation periphery.

For these reasons, realization rules must be sensitive not only to punctuation marks that have attached themselves to the head as adjuncts, but also to punctuation marks at the inside and outside punctuation periphery of the phrase, which can be retrieved with the leftp and rightp operators (cf. Definition 2.73).

The realization rules for Danish and English punctuation are complicated, and specifying them in detail would lead us too far. Indeed, rather than specifying the cost functions for punctuation manually, we believe it is better to account for punctuation by means of a probabilistic language model, as proposed in section 6.2. The following examples are therefore only intended as illustrations of how the leftp and rightp operators can be used to formulate realization rules — they are not meant as complete, detailed proposals in their own right.

**Example 4.71** The cost function below can be used to stipulate that a head must have an obligatory comma (or a valid substitute) at its right punctuation periphery. The cost function checks whether there is a valid unary substitute in the right punctuation periphery of the head (identified by the cost function $\text{rightp}(\text{this}, [\xleftarrow{\text{pnct}} \text{any}], [\xleftarrow{\text{pnct}} \text{any}])$), or a valid binary substitute at the head or in its outside right punctuation periphery (identified by the cost function $\text{rightp}(\text{this}, [\xleftarrow{\text{pnct}} \text{this}], [\xleftarrow{\text{pnct}} \text{any}])$), assuming the type hierarchy for punctuation marks given in Figure 4.49. If no such punctuation mark exists, the cost 100 is returned.

```
100 * not(
    (rightp(this, $[\xleftarrow{\text{pnct}} \text{any}]$, $[\xleftarrow{\text{pnct}} \text{any}]$) + unarymark)
    | (rightp(this, $[\xleftarrow{\text{pnct}} \text{this}]$, $[\xleftarrow{\text{pnct}} \text{any}]$) + endbracket))
```

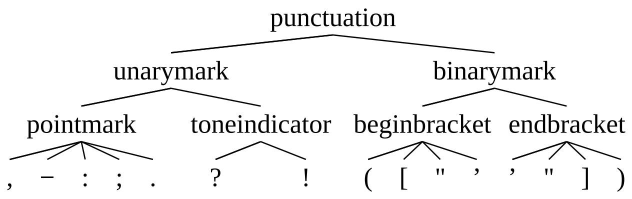


Figure 4.49: Type hierarchy for punctuation lexemes.

At node "was" in Figure 4.47, the operator rightp(this, [$\xleftarrow{\text{pnct}}$ any], [$\xleftarrow{\text{pnct}}$ any]) will return the node corresponding to the second dash, which matches the type specification unarymark; after negation, the cost function will therefore return the cost 0, indicating that the cost function is not violated. If the second dash had been deleted, the two rightp operators would return the empty set, thereby returning the cost 100 after negation and multiplication.

**Example 4.72** The cost function below can be used at the head of a first conjunct to ensure that each secondary conjunct in the coordination is preceded by a comma or a valid substitute if it lacks a coordinator.

> 100 * (([$\xleftarrow{\text{conj}}$ this] − [$\xrightarrow{\text{coord}}$ any])
> +where($n$, not(leftp(var(n), [$\xleftarrow{\text{pnct}}$ any], [$\xleftarrow{\text{pnct}}$ any])+unarymark)))

Note that this cost function is also satisfied in (47)–(49) below, where the conjuncts are separated by periods and question marks, respectively. But it rules out examples like (50).

(47) They saw many animals. A cow. A horse. A mule. And a rabbit.

(48) They saw a cow and a horse. And a rabbit.

(49) What did they see: A cow? A horse? A mule? Or a rabbit?

(50) *What did they see: A cow a horse a mule or a rabbit?

The cost function below can be used to ensure that the same unary mark is used to separate all the conjuncts in the coordination. The cost function identifies a conjunct $C1 that precedes another conjunct $C2, and computes the first unary mark preceding each conjunct within the outside left punctuation periphery of the conjunct; if the marks are non-identical, then the cost function returns the cost 100 for each non-matching pair of marks.

$$\begin{aligned}
&100 * (\text{var}(\$\text{C1}, [\xleftarrow{\text{conj}} \text{this}]) + \text{var}(\$\text{C2}, [\xleftarrow{\text{conj}} \text{this}] + [> \$\text{C1}])\\
&\quad + (\text{val}(\text{lexeme}, \text{prev}(\text{leftp}(\$\text{C1}, -\text{any}, [\xleftarrow{\text{pnct}} \text{any}]), \$\text{C1}))\\
&\qquad \neq \text{val}(\text{lexeme}, \text{prev}(\text{leftp}(\$\text{C2}, -\text{any}, [\xleftarrow{\text{pnct}} \text{any}]), \$\text{C2}))))
\end{aligned}$$

This cost function will prevent examples like (51).

(51) *They saw a cow, a horse; a mule. And a rabbit.

**Remark 4.73 (word order rules)** Adjunct rules specify which governors a punctuation adjunct may attach to, and realization rules specify which punctuation marks must obligatorily be present at a head. However, these rules cannot prevent punctuation marks from occurring in the wrong position within the phrase that they attach to — eg, they cannot prevent a period from preceding its landing site in the linear order. For this purpose, we need a set of *word order rules* which specify where punctuation marks cannot occur within the linear word order. This is achieved by means of cost functions that impose a cost for every word order violation they detect, as illustrated by the following example.

**Example 4.74** The word order rules for periods are rather complicated. For example, (52) and (54) show that periods can be inserted before sentence adverbials, but (53) shows that they cannot be inserted before direct objects. The periods in (52) and (54) correspond to particular pause and intonation patterns in spoken language, which probably indicate either emphasis or some kind of afterthought.

(52) Please leave this room. Instantly.

(53) *Please leave. This room. Instantly.

(54) She presented the poem. To a group of critics. Yesterday.

The situation becomes even more complicated if we analyze discourse relations as dependency relations, as we have argued for in section 4.5. For then the surface phrase associated with the head of a sentence includes all discourse segments that have been attached to the head of the phrase, not just the sentential parts, so instead of saying that a period must be placed at the end of a sentence, we must say that a period must be placed after all landed nodes that have been analyzed as syntactic dependents, but before all right landed nodes that have been analyzed as discourse dependents. For example, in (55) (analyzed in Figure 4.36 on page 171), the period in the first sentence must be placed before the direct object “a fantastic meal”, but before the two discourse units “He ate salmon. He devoured lots of cheese.” and “He won a dancing competition”.

(55) He had a fantastic meal. He ate salmon. He devoured lots of cheese. He won a dancing competition.

As a tentative generalization from these examples, we propose that periods are subject to the following word order rules: (a) periods must always succeed their governors; (b) periods can only appear immediately before a discourse adjunct or (with the purpose of indicating emphasis or afterthought) another adjunct. The first rule can be encoded by means of the cost function below, which imposes a cost of 100 every time a period precedes its governor.

$$100 * ([\xleftarrow{\text{pnct}} \text{this}] + \text{period} + [< \text{this}])$$

The second rule can be encoded by means of the cost function below, which imposes a cost of 100 every time the landed node immediately after a period fails to be an adjunct.

$$100 * (\text{next}([\xleftarrow{\text{land}} \text{this}], [\xleftarrow{\text{pnct}} \text{this}] + \text{period}) - [\xleftarrow{\text{adjunct}} \text{this}])$$

**Remark 4.75 (adjacency rules)** Word order rules prevent bad orderings of punctuation marks locally at a landing site, but cannot weed out ungrammatical sequences of punctuation marks. For example, (56) contains the ungrammatical punctuation sequence “,—.” which cannot violate any word order rules because the three punctuation marks are attached to different nodes, and hence appear within different word order domains, as shown in Figure 4.50.

(56) *I like tiramisu — but I avoid pasta, if I can, — .

To avoid such sequences, we need a set of cost functions that impose a cost whenever an invalid sequence is encountered. For example, the following cost function will impose a cost of 100*n* if a unary punctuation mark is immediately followed by a sequence of *n* unary punctuation marks.

$$100 * \text{rightp}(\text{this}, \text{-any}, [\xleftarrow{\text{pnct}} \text{any}] + \text{unary})$$

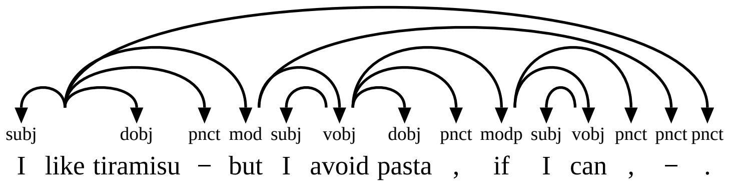


Figure 4.50: Example of ungrammatical punctuation.

**Remark 4.76 (bracket rules)** We also need a set of bracket rules that impose a cost if binary punctuation marks fail to be paired with a matching punctuation mark. To account for this matching condition, we will make the following assumptions: (a) begin brackets are lexically encoded as anaphora whose antecedent must be an end bracket of the same kind — ie, '(' must be matched by ')', and similarly with quotation marks (cf. Remark 4.53); (b) cost functions at the begin bracket ensure that the begin bracket must have an antecedent that succeeds the begin bracket and has the same landing site as the begin bracket; (c) a cost function associated with the end bracket ensures that the end bracket must function as an antecedent for exactly one begin bracket; and (d) a cost function associated with begin brackets prevents crossing brackets, ie, if a begin bracket $B_1$ occurs between two matching brackets $A_1$ and $A_2$, then the matching end bracket $B_2$ for $B_1$ must occur between $A_1$ and $A_2$ as well.

Condition (a) is encoded in anaphor frames, and the encoding of conditions (b) and (c) using cost functions is straight-forward. The cost function below can be used to encode condition (d) at a begin bracket. The leftp operator returns all nodes that are contained within the two matching brackets; for each begin bracket among these, the cost function then imposes a cost if the matching end bracket occurs after the end bracket of the active node.

$$\begin{aligned}&100 * ([\xrightarrow{\text{match}} \text{leftp}([\xrightarrow{\text{match}} \text{this}], -\text{any}, \text{any}-\text{this})+\text{beginbracket}]\\ &\quad +[> [\xrightarrow{\text{match}} \text{this}]])\end{aligned}$$

**Remark 4.77 (movement rules)** Punctuation marks are assumed to be non-extractable (ie, a punctuation mark must land on its governor), with a single exception: quote transposition (cf. Nunberg 1990, p. 64), which is the convention that if an end quote is immediately followed by a period or a comma, then the end quote must be moved to a position right after this

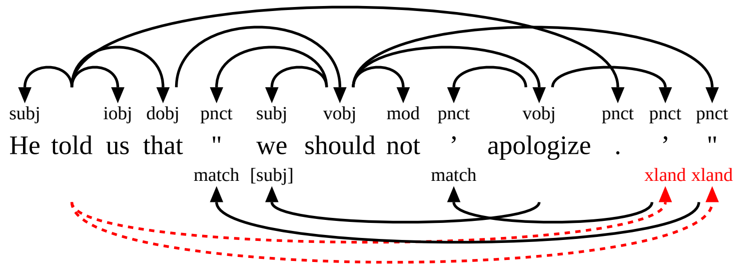


Figure 4.51: Movement analysis of quote transposition.

mark.[13] We can account for this phenomenon in DG by assuming that the end quote is moved upwards to a position where it can succeed the comma or period, as illustrated by Figure 4.51.

This analysis can be formalized as follows: (a) all words must be equipped with a cost function that blocks the extraction of any punctuation mark other than an end quote; (b) end quotes must be equipped with a cost function that ensures the correct placement of the end quote, ie, any nodes between the end quote and the surface phrase it modifies must be either commas, periods, or other end quotes. The condition (a) can be expressed by means of the cost function below, which punishes any extracted punctuation mark that fails to be an end quote.

$$100 * (\mathsf{extract}(\mathsf{this}) + \mathsf{punctuation} - \mathsf{endquote})$$

The condition (b) can be expressed by means of the cost function below. The rightp operator finds all nodes that succed the right boundary of the governor of the end bracket while preceding the end bracket. The remaining part of the cost function imposes a cost of 100 for every such node that fails to be either a comma, a period, or an end bracket.

$$100 * (\mathsf{rightp}([\xrightarrow{\mathsf{pnct}} \mathsf{this}], -\mathsf{any}, [< \mathsf{this}]) - (\mathsf{comma}|\mathsf{period}|\mathsf{endbracket}))$$

[13]The “Frequently Asked Questions” file of alt.english.usage provides the following historical explanation of quote transposition: “According to William F. Phillips (wfp@world.std.com), in the days when printing used raised bits of metal, ‘.’ and ‘,’ were the most delicate, and were in danger of damage (the face of the piece of type might break off from the body, or be bent or dented from above) if they had a ‘”’ on one side and a blank space on the other. Hence the convention arose of always using ‘.”’ and ‘,”’ rather than ‘”.’ and ‘”,’, regardless of logic.”

In this section, we have argued that punctuation is a challenging linguistic phenomenon which should be addressed by any syntactic theory. We have also sketched how cost functions can be used to give a rule-based account of punctuation. In section 6.2, we will present an alternative probabilistic account of punctuation.

## 4.7 Lexical transformations: morphology

***Summary**. We outline how DG can be used to account for morphology. After explaining why lexical transformations are needed to account for morphology, we show how lexical transformations can be used to encode inflectional and derivational morphology in the DG lexicon, as well as lexical alternations such as expletives and passives.*

So far, we have seen how dependency analyses can be used to account for syntax and discourse. Can dependency analyses also be applied to morphology, ie, can we use a single dependency tree for all levels of linguistic analysis from morphology to discourse? In this section, we will argue that this extension is possible, but that there are some morphological phenomena where we need an additional mechanism: lexical transformations.

**Remark 4.78** Many morphological phenomena are concatenative, ie, complex morphological units are formed by concatenating simpler morphological units, as in compounds (cf. Fabb 1998). For example, the Danish compound "haglskadeforsikringsselskabet" in (57), which is an authentic example from the Danish Dependency Treebank, can be analyzed as a concatenation of words and morphemes; the equal signs are used to separate morphemes within a single word.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| (57) | hagl= | skade= | forsikr= | ing= | s= | selskab= | et |
| | hail | damage | insur | ance | GLUE | company | the |

'the insurance company that covers damage caused by hail'

A possible dependency analysis of (57) is shown in Figure 4.52. The word "selskab" is analyzed as the head of the entire compound, while the remaining words have been analyzed as `comp` (compound) adjuncts in a hierarchical structure. The "ing" morpheme is a derivational morpheme that turns a verb into a noun, the "s" morpheme is a glue morpheme without independent meaning, and "et" is a definiteness morpheme; unlike the other

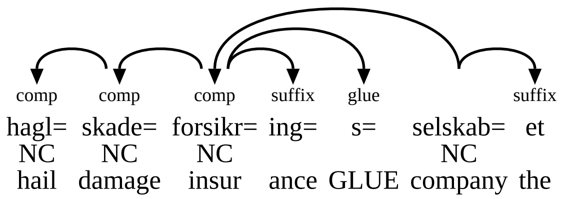

comp
comp
comp
suffix
glue
suffix
hagl= skade= forsikr= ing= s= selskab= et
NC NC NC NC
hail damage insur ance GLUE company the

Figure 4.52: Dependency analysis of the compound (57).

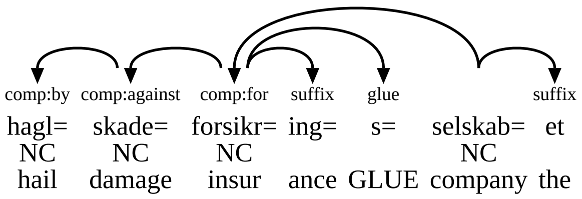

comp:by
comp:against
comp:for
suffix
glue
suffix
hagl= skade= forsikr= ing= s= selskab= et
NC NC NC NC
hail damage insur ance GLUE company the

Figure 4.53: Alternative dependency analysis of the compound (57).

morphemes, these three morphemes are not analyzed as independent lexemes in DG, as we will see in Remark 4.91.

Other dependency analyses are possible. For example, instead of using a single edge label comp for all compound adjuncts, we could have used a more fine-grained label that specifies how the compound adjunct modifies the head, eg, by means of a preposition that indicates the relationship between the compound noun and the head noun, as in the dependency analysis of (57) in Figure 4.53 which specifies that "the hail damage insurance company" can be paraphrased as "the company for insurance against damage by hail". Instead of using prepositions as labels, we could equally well have used semantic roles, or we could have analysed some of the modifiers as complements (eg, "damage" as a prepositional object of "insure").

**Remark 4.79** Obviously, if we allow nodes in a dependency graph to represent morphemes, we must explicitly encode spaces in an analysis of written language. One idea is to assume that spaces function as punctuation marks that attach themselves as adjuncts to the head of the preceding word, before any syntactic dependents but after all morphological dependents. Figure 4.54 shows how the phrase "Vi gennemanalyserede forsikringsselskabet Codan" in (58) can be given a dependency analysis that includes spaces and morphology.

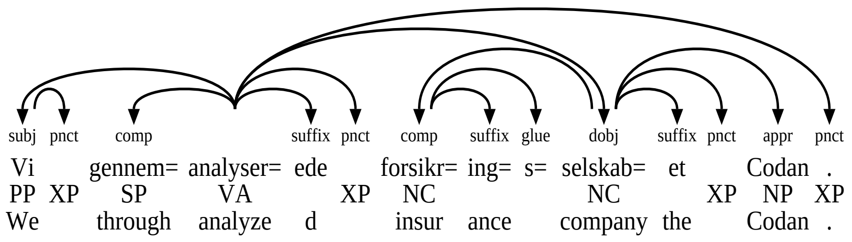

subj pnct comp suffix pnct comp suffix glue dobj suffix pnct appr pnct
Vi gennem= analyser= ede forsikr= ing= s= selskab= et Codan .
PP XP SP VA XP NC NC XP NP XP
We through analyze d insur ance company the Codan .

Figure 4.54: Dependency analysis of (58) with spaces.

(58) Vi gennem= analyser= ede forsikr= ing= s= selskab= et Codan.
We through analyze d insur ance company the Codan.
'We thoroughly analyzed the insurance company Codan.'

When nodes represent morphemes, we must ensure that word order constraints prevent syntactic dependents and word external punctuation signs from being placed between a lexeme and its morphological dependents.

**Remark 4.80** By using the same dependency structure to analyze both syntax and morphology, we simplify the problem of specifying the interface between morphology and syntax. For example, Danish allows coordinations where two morphemes are coordinated as in syntax (ie, the conjuncts are separated by spaces), although the coordinated structure serves as a morphological dependent of another word. An example is given by (59) "Hvad betaler du for ind- og udlandstelefoni?" which can be analyzed as shown in Figure 4.55.

(59) Hvad betaler du for ind- og ud= land= s= telefon= i?
What pay you for in- and out country telephon y?
'What do you pay for national and international phone calls?'

Examples like (59), where a syntactic phrase functions as a morphological dependent, make it difficult to maintain a strict separation between syntax and morphology. A similar phenomenon can be observed in English sentences like (60), where the compound adjunct "shoot-anything-that-moves" is a syntactic phrase which has been coerced into a morphological unit by replacing all spaces with hyphens.

(60) He was in his shoot-anything-that-moves mode.

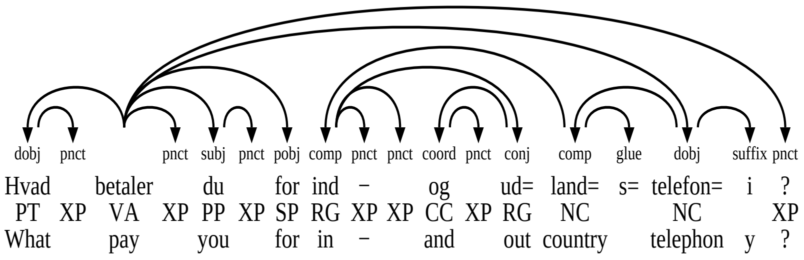

dobj
pnct
pnct
subj
pnct
pobj
comp
pnct
pnct
coord
pnct
conj
comp
glue
dobj
suffix
pnct
Hvad betaler du for ind – og ud= land= s= telefon= i ?
PT XP VA XP PP XP SP RG XP XP CC XP RG NC NC XP
What pay you for in – and out country telephon y ?

Figure 4.55: Dependency analysis of (59) with morphological coordination.

Morphology would be simple if it was always concatenative. Unfortunately, it is not.

**Remark 4.81** In morphology, there are many non-concatenative phenomena. For example, the past of regular verbs in English is formed by adding the suffix "ed" or "d" to the end, eg, "walk/walked" and "die/died"; but for irregular verbs, we observe word pairs such as "see/saw", "sing/sang", and "catch/caught" where the entire stem is modified. Other languages, such as Arabic, exhibit even more complicated patterns of non-concatenative morphology — for an overview of these, see Spencer (1998). Since DG's complement and adjunct mechanisms are concatenative, we need some kind of additional formal machinery to account for non-concatenative morphology. In DG, this machinery is based on the lexical transformations introduced in Definition 2.17 and on the non-concatenative morphological operators described below.

**Remark 4.82** In order to account for non-concatenative morphology in DG, we will allow spellings and pronunciations to be specified as stems followed by sequences of morphological operators. A morphological operator is called *concatenative* if it does not affect the stem (ie, it encodes a possibly context-sensitive suffix or prefix), and *non-concatenative* if it affects the stem (ie, it encodes a vocal shift, a shortening, or some other replacement operation within the stem). We assume that the inventory of morphological operators is partly language-dependent — ie, some morphological operators (such as suffixation, prefixation, and ending replacement) may be genetically encoded, whereas other more specialized operators are learned during language acquisition.

| Ops. | Definition | Example |
|---|---|---|
| $\alpha$ | attach suffix $\alpha$ | låse + r $\mapsto$ låser |
| $\{\#\alpha\}$ | attach prefix $\alpha$ | se + {#be} $\mapsto$ bese |
| $\{[\alpha]\}$ | ensure ending $\alpha$ | låse + {[e]} $\mapsto$ låse |
| $\{?\}$ | duplicate ending consonant | skat + {?} $\mapsto$ skatt |
| $\{\alpha/\alpha'\}$ | replace ending $\alpha$ with $\alpha'$ | viser + {r/s} $\mapsto$ vises |
| $\{v\hat{}v'\}$ | replace last vocal $v$ with $v'$ | syng + {y^a} $\mapsto$ sang |
| $\{\sim v\}$ | delete ending vocal $v$ with consonant undoubling (ie, $ccvc' \mapsto cc'$ and $vc' \mapsto c'$ otherwise) | varsel + {∼e} $\mapsto$ varsl<br>pukkel + {∼e} $\mapsto$ pukl |

Figure 4.56: A set of concatenative (top and middle) and non-concatenative (bottom) morphological operators suitable for Danish.

**Example 4.83** Figure 4.56 shows a set of concatenative and non-concatenative morphological operators suitable for Danish morphology. In other languages, a slightly different set of morphological operators may be more appropriate. The context-free concatenative operators (top) can be used to add a given suffix or prefix, whereas the context-sensitive concatenative operators (middle) can be used to ensure a given ending by adding a minimal suffix, or to duplicate an ending consonant. The non-concatenative operators (bottom) can be used to replace one ending with another, to replace the most recent vocal with another, or to delete the most recent vocal while undoubling any double consonant that immediately precedes it.

**Remark 4.84** In order to ensure that morphological operators are computationally efficient, we will limit their power by assuming that they are *local*, ie, that they cannot affect any part of the morpheme sequence that precedes the last suffix.[14] With this locality assumption, a morpheme sequence such as "vær + e + {være/er}" is ill-formed because the operation "{være/er}" is only allowed to apply to the preceding suffix "e", not to

[14]We conjecture that local morphological operators are powerful enough to give a natural and computationally efficient account of the morphology of most languages, in particular Danish, English, and German. It is unclear to us whether there are languages where the locality condition is unsatisfactory. From a formal point of view, the locality condition can always be upheld by encoding a suffix $\beta$ by means of a non-concatenative operator $\alpha/\alpha\beta$. However, this solution is unsatisfactory if it results in more than a small number of exceptional rules.

"vær + e". To encode the derivation of "er" from "vær" via "være", we must instead use a non-concatenative morpheme sequence such as "vær + {vær/være} + {være/er}".

**Remark 4.85** Concatenative and non-concatenative morphological operators must be treated differently in the lexicon. If a transformed lexeme $t'$ is derived from a base lexeme $t$ by means of a sequence of concatenative transformations, then the speech signal for $t$ is part of the speech signal for $t'$, so $t$ is automatically retrieved whenever we hear $t'$; but if the derivation of $t'$ involves non-concatenative transformations, the lexicon must be compiled so that we know all the transformed roots associated with a given base lexeme. Whenever a transformed root associated with a base lexeme is encountered in the speech signal, the set of all matching transformed lexemes is then computed from the base lexeme.

For example, the lexeme "go" has the transformed root "went", so hearing the speech signal "went" must somehow trigger the retrieval of the lexeme "go". We will assume that after retrieving "go", the human lexicon applies the set of all available transformations to "go" in order to find all associated transformations matching "went". In that way, it will end up returning the transformed lexeme "go.PAST" which matches "go + {go/went} = went".

**Example 4.86** We have made an analysis of a computational morphological dictionary for Danish by Peter Molbæk containing 36,525 base forms of Danish verbs, nouns, and adjectives. Figure 4.57 shows the number of distinct sequences of morphological operators needed to account for the most important inflections in Danish, using the morphological operators in Figure 4.56. The table shows that each inflection involves a relatively small number of distinct morphological transformations, often not more than a handful and never more than fifty. Figure 4.58 and 4.59 show the ten most frequent morphological operations for computing a past verb from an imperative verb, and an indefinite plural noun from an indefinite singular noun, respectively (the absence of an operator indicates unchanged spelling, ie, a "zero" morpheme). These examples also show that even if we only look at the most frequent paradigms in Danish, the entire inventory of morphological operators is needed.

So far, we have merely introduced the morphological operators that are

| verbs | |
|---|---|
| imperative $\mapsto$ past | 37 |
| imperative $\mapsto$ perfect | 28 |
| imperative $\mapsto$ infinitive | 3 |
| infinitive $\mapsto$ present | 7 |
| infinitive $\mapsto$ gerund | 2 |
| infinitive $\mapsto$ participle | 2 |
| present $\mapsto$ present passive | 3 |
| past $\mapsto$ past passive | 3 |

| nouns | |
|---|---|
| sg.indef $\mapsto$ sg.def | 16 |
| sg.indef $\mapsto$ pl.indef | 43 |
| pl.indef $\mapsto$ pl.def | 11 |
| **adjectives** | |
| pos.com $\mapsto$ pos.neut | 4 |
| pos.com $\mapsto$ pos.def/pl | 8 |
| pos.com $\mapsto$ comparative | 12 |
| comparative $\mapsto$ superlative | 4 |

Figure 4.57: Inflections in Danish and the number of distinct sequences of morphological operators needed to account for all inflected words in the Molbæk dictionary.

| Count | Operations | Example roots |
|---|---|---|
| 4516 | ede | tro lav prøv arbejd bo |
| 830 | {?} ede | spil stil byg støt slut |
| 603 | te | men ske brug begynd hør |
| 104 | {æ^a} {?} e | sæt besæt afsæt tilsæt udsæt |
| 100 | {i^e} | bliv grib skriv driv stig |
| 69 | {ør/jorde} | gør afgør omgør opgør udgør |
| 67 | {y^ø} | lyd bryd frys snyd kryb |
| 62 | {a^o} | far jag lad tag drag |
| 59 | | kom hed løb sov græd |
| 53 | {i^a} | gid giv sid drik kling |

Figure 4.58: The most frequent operations for computing past verbs from imperative verbs in Danish.

responsible for producing inflected spellings or pronunciations from base forms. We will now describe how the DG lexicon can be used to encode morphology by coupling morphological operators with the lexical transformations defined in Definition 2.17.

**Remark 4.87** The intuition behind lexical transformations is that some lexemes are more basic than others: *base lexemes* (representing the base forms of words) are permanently listed in the lexicon, whereas *transformed lexemes* (representing inflected or derived forms) are generated during lexical

| Count | Operations | Example roots |
|---|---|---|
| 11975 | {[er]} | krone tid sted uge procent |
| 2635 | {[e]} | gang dag del vej tilfælde |
| 984 | | år folk par liv spørgsmål |
| 852 | {?} er | ven minut klub medlem hotel |
| 276 | {∼e} er | verden eksempel regel aften artikel |
| 128 | {∼e} e | forælder teater nummer meter alder |
| 84 | {?} e | krop blik forskel drøm ryg |
| 81 | s | ton show dollar cup interview |
| 67 | er | te le ide ske entre |
| 40 | {um/a} | minimum faktum visum kuriosum votum |

Figure 4.59: The most frequent operations for computing plural indefinite nouns from singular indefinite nouns in Danish.

lookup from a base lexeme by means of the transformation rules associated with the base lexeme and its transformed lexemes. The transformation rule specifies the phonology, syntax, and semantics of the target lexeme by computing a set of local feature functions that must be applied to the source lexeme in order to produce the target lexeme. Since transformations can be inherited and dynamic lexemes can function as source lexemes for new transformations, lexical transformations enable a highly compact encoding of the lexicon.

**Example 4.88** Figure 4.60 shows how Danish verbs are inflected.[15] The imperative verb functions as the base lexeme, and is equipped with three transformations that produce the infinitive, past, and perfect forms of the verb, respectively. The infinitive is in turn equipped with three transformations that produce the present, present participle, and gerund forms. The infinitive, present, past, and perfect forms (boldfaced) are in addition equipped with transformations for computing the passive and expletive

[15]In Danish grammar, it is often assumed that the imperative, past, and perfect are derived from the infinitive. However, since the infinitive is mophologically more complex than the imperative, we believe that choosing the imperative as the base form gives a simpler account of verbal morphology. If this is correct, then one might expect Danish children to use imperative forms before they start using infinitive forms. However, I do not know whether this prediction is empirically valid or not.

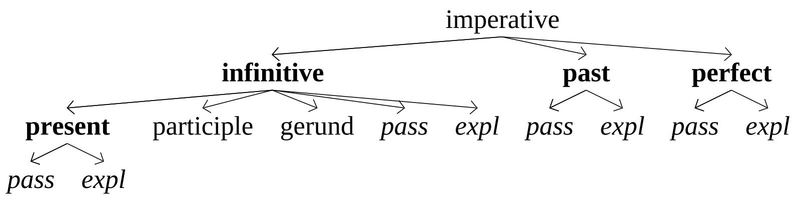


Figure 4.60: The inflectional transformations for Danish verbs (passives and expletives apply to boldfaced forms).

form of the verb. That is, no passive or expletive form is associated with imperatives, participles, and gerunds.

We will now sketch how these transformations can be encoded in the DG lexicon. All transformation types will be encoded as subtypes of the type transform shown below. The function &append_tphon ensures that the phon feature of a transformed type is computed as the phon feature of the source type concatenated with the tphon feature of the transformation type; the function &delete_transforms ensures that all features in the source type starting with "transform:" are set to null; and the function &copy_features copies all features in the transformation type starting with "#" to the transformed type after deleting the initial "#" in the feature name.

```
type(transform)
    −>tfunc(tfunc:phon, phon, &append_tphon)
    −>tfunc(tfunc:transform, transform, &delete_transforms)
    −>tfunc(tfunc:xfeatures, '', &copy_features);
```

For each inflection, we specify an abstract transformation type (eg, infinitive_transform below) that specifies that the transformed complement frames, and hence the semantics of the inflected forms, are computed from the source lexeme by the function &transform_cframes_infinitive. Moreover, we list all the transformations that are licensed by the transformed infinitival lexeme by default (present, participle, gerund, expletive, passives, etc.).

```
type(infinitive_transform)
    −>super(transform)
    −>tfunc(tfunc:cframes, 'cframe', &transform_cframes_infinitive)
    −>set('#transform:present', 'present_transform_[e]r')
    −>set('#transform:participle', 'participle_transform_[e]nde')
    . . .
```

For each inflection pattern, we must define a transformation type that links an abstract transformation type to a specific morphological operation, overriding any unwanted defaults. For example, the transformation type infinitive_transform_?e defines an infinitive transform with the associated morphological operation "{?} e".

```
type(infinitive_transform_?e)
    −>super(infinitive_transform)
    −>set(tphon, ['{?}', e]);
type(infinitive_transform_0)
    −>super(infinitive_transform)
    −>set(tphon, []);
    −>set('#transform:participle', 'participle_transform_ende');
```

We can now specify that an imperative is by default a verb which licenses three different transformations: an infinitive inflection with morphological operation "e", a past inflection with operation "ede", and a perfect inflection with operation "et".

```
type(imperative)
    −>super(verb)
    −>transform(infinitive, infinitive_transform_e)
    −>transform(past, past_transform_ede)
    −>transform(perfect, perfect_transform_et);
```

For particular verbs, we can now specify the lexical entry for the imperative form of the verb. Regularly inflected verbs like "leg" ("play") inherit all the default inflections provided by the supertype imperative, whereas irregularly inflected verbs like "se" ("see") must overwrite the inherited defaults if they are incorrect.

```
type(leg:v)
    −>super(imperative)
    −>phon(leg)
    −>cframe(. . .);
type(se:v)
    −>super(imperative)
    −>phon(se)
    −>cframe(. . .)
    −>transform(infinitive, 'infinitive_transform_0')
    −>transform(past, 'past_transform_eˆå')
    −>transform(perf, 'perf_transform_t');
```

The DG lexicon allows other ways of encoding morphology (eg, we could have stored the entire inflectional paradigm in features associated with the base lexeme). But the example we have given demonstrates the transformational machinery needed to encode morphology efficiently in DG.

**Example 4.89 (expletives and passives in Danish)** Lexical transformations can also be used to account for expletives and passives in Danish. Figure 4.61 lists the expletive and passive transformations that are responsible for producing the expletive (61), the expletive passive (62), the direct object passive (63), the indirect object passive (64), the sentential passive (65), and the prepositional passive (66). We assume that the passive and expletive transformations do not affect the lexical meaning directly, but can affect the lexical meaning indirectly by changing the topic-focus structure. The lexical transformations responsible for expletives and passives can be encoded in the same way as inflections, as sketched in Example 4.88.

(61) En gæst vil komme. Der vil komme en gæst.
A guest will come. There will come a guest.

(62) Han kører forsigtigt. Der køres forsigtigt af ham.
He drives gently. There is-driven gently by him.

(63) Han vækker et barn. Et barn vækkes af ham.
He awakens a child. A child is-awoken by him.

(64) Han giver os den. Vi gives den af ham.
He gives us it. We are-given it by him.

(65) Vi frygter hun synger. Hun frygtes af os at synge.
We fear she sings. She is-feared by us to sing.

| **transform** | **source frame** | **target frame** |
|---|---|---|
| expletive (61) | subj=$X$ | subj=null |
| | dobj=null | dobj=$X$ |
| | expl=null | expl=any expletive |
| | *no prep. objects* | *dobj must be discourse-new* |
| expletive passive (62) | subj=$X$ | subj=null |
| | pobj:af=null | pobj:af=af+$[\xrightarrow{\textsf{nobj}} X]$ |
| | | expl=any expletive |
| | | *dobj must be discourse-new if present* |
| direct object passive (63) | subj=$X$ | subj=$Y$ |
| | dobj=$Y$ | dobj=null |
| | pobj:af=null | pobj:af=af+$[\xrightarrow{\textsf{nobj}} X]$ |
| indirect object passive (64) | subj=$X$ | subj=$Y$ |
| | iobj=$Y$ | iobj=null |
| | pobj:af=null | pobj=af+$[\xrightarrow{\textsf{nobj}} X]$ |
| sentential passive (65) | subj=$X$ | subj=noun |
| | vobj=$Y$+finite | vobj=$Y$+infinitive |
| | pobj:af=null | pobj:af=af+$[\xrightarrow{\textsf{nobj}} \mathrm{X}]$ |
| | | *subj must be filler subj of vobj* |
| prepositional passive (66) | subj=$X$ | subj=$Z$ |
| | pobj:$Y$=$Y$+$[\xrightarrow{\textsf{nobj}} Z]$ | pobj:$Y$=null |
| | pobj:af=null | pobj:af=af+$[\xrightarrow{\textsf{nobj}} X]$ |
| | dobj=null | |
| | vobj=null | |

Figure 4.61: Expletive and passive transformations in Danish.

(66) Han lytter til hjertet. Hjertet lyttes til af ham.
He listens to heart-the. Heart-the is-listened to by him.

**Remark 4.90** Derivational morphology is encoded in the DG lexicon in the same way as inflectional morphology — ie, each derivation that a particular lexeme can undergo is encoded by means of a lexical transformation that is either specified in the local feature function of the lexeme or inherited from a supertype.

**Remark 4.91** So far, we have introduced two different mechanisms for encoding morphology in DG: morphological complements and adjuncts as described in Remark 4.78 for compounding, and lexical transformations with morphological operators as described in Remarks 4.82 and 4.87 for inflectional and derivational morphology and lexical alternations. The first mechanism only works for concatenative morphology, whereas the latter mechanism works for both concatenative and non-concatenative morphology. This means that there is a choice with respect to how concatenative morphological phenomena are encoded. So what determines this choice?

Obviously, noun compounds of the form "$X$ $Y$" are best treated by means of morphological complements and adjuncts, for if we treat $X$ as a prefix or a suffix produced by a lexical transformation of $Y$, then we need a different transformation function for each of the many thousand choices of $X$. Conversely, the third person singular suffix "s" in English (or the glue morpheme "s" in Danish) is best encoded by means of a lexical transformation, for if we encode it as an independent lexeme, then the parser will be forced to treat each occurence of the letter "s" in the speech signal as a potential instance of the suffix, thereby resulting in an unnecessary amount of ambiguity in the grammar. In general, we will prefer the mechanism that results in the most efficient encoding of the grammar in terms of space and processing complexity. However, like with the complement-adjunct distinction, there may be borderline cases where both mechanisms can sensibly be used because the difference in economy is small.

As a guiding principle, we will assume that morphemes are encoded as independent lexemes when they function as words in their own right or can be separated from the stem by other dependents (as in compounding, incorporation, and clitics), and that they are encoded by means of lexical transformations when the morpheme is closely coupled with the root (as in inflectional and derivational morphology).

**Remark 4.92** There are two distinct research traditions in morphology (cf. Spencer 1998). The *Item-and-Arrangement tradition* (Hockett 1958; Chomsky and Halle 1968; Lieber 1980; Wiese 1996) assumes that an inflected form is produced by concatenating a stem with an abstract, lexically listed morpheme that has its own phonology, syntax, and semantics. In contrast, the *Item-and-Process tradition* (Sapir 1921; Hockett 1958; Matthews 1991) assumes that an inflected form is produced by an inflectional process that computes the inflected form from the stem — ie, there is no principled difference between concatenative and non-concatenative morphology, they just represent different phonological operations (concatenation and stem modification).

DG is in some sense indebted to both traditions. On the one hand, DG adheres to the Item-and-Process view by assuming that inflectional and derivational morphemes are not independent lexemes, but rather the result of lexical transformations. On the other hand, the locality condition proposed in Remark 4.84 gives suffixation a special status, and DG's notion of transformation types and concatenative and non-concatenative morphological operators is perhaps not all that far from the Item-and-Arrangement view. In this way, DG morphology combines aspects of both traditions.

In this section, we have outlined how DG can be extended to morphology by using complement and adjunct mechanisms to account for compounds and by using lexical transformations to account for inflections, derivations, and lexical alternations such as expletives and passives.

## 4.8 The Danish Dependency Treebank

***Summary***. *We briefly describe the Danish Dependency Treebank, and how the analyses in the treebank relate to the analyses presented in this dissertation. We provide an estimate for the inter-annotator agreement with the DDT scheme and briefly describe the DTAG treebank tool that has been used to create the DDT annotations.*

In this chapter and the two preceding chapters, a wide range of phenomena have been analyzed within the framework of DG. These analyses — or, rather, earlier versions of these analyses — have formed the basis of our work on the *Danish Dependency Treebank* (*DDT*) and the accompanying annotation manual (Kromann et al. 2003; Kromann and Lynge 2004). In this section, which incorporates material from (Kromann 2003), we briefly describe our work on the DDT and the few differences between the analyses there and the analyses proposed in this dissertation.

**Remark 4.93** The Danish Dependency Treebank is a dependency treebank with 5,540 sentences totalling 100,200 tokens. DDT consists of 536 randomly selected texts from the morphosyntactically tagged Danish PAROLE corpus (Keson and Norling-Christensen 1998), a balanced corpus of written text consisting of 1,553 text samples containing approximately 150-250 tokens each and covering a wide range of different text mediums, genres, and topics. DDT annotations have been created manually by a research assistant at a rate of approximately 1,000 words per day, using the DTAG

| | | |
|---|---|---|
| **A**: adjective | **P**: pronoun | **V**: verb |
| **AN**: normal | **PP**: personal | **VA**: main |
| **AC**: cardinal | **PD**: demonstrative | **VE**: medial |
| **AO**: ordinal | **PI**: indefinite | **X**: extra-linguistic unit |
| **C**: conjunction | **PT**: interrog./relative | **XA**: abbreviation |
| **CC**: coordinating | **PC**: reciprocal | **XF**: foreign word |
| **CS**: subordinating | **PO**: possessive | **XP**: punctuation |
| **I**: interjection | **RG**: adverb | **XR**: formulae |
| **N**: noun | **SP**: preposition | **XS**: symbol |
| **NP**: proper | **U**: unique | **XX**: other |
| **NC**: common | | |

Figure 4.62: The main PAROLE word class tags for Danish.

treebank tool (Kromann 2005). Approximately 25% of the texts have been double-checked by the author of this dissertation.

**Remark 4.94** DDT is based on the Danish PAROLE corpus (Keson and Norling-Christensen 1998), where all tokens are tagged with a lemma and a morphosyntactic PAROLE tag. The main PAROLE word classes for Danish are shown in Figure 4.62. DDT extends the PAROLE corpus by encoding relations between words, such as primary dependencies, secondary dependencies, and syntactically determined coreference in relative clauses, resumptive pronouns, etc. The filler and coreference annotations in DDT are necessary for recovering the underlying functor-argument structure from the annotations. The most important edge labels for primary dependencies in DDT are shown in Figure 4.63.

Secondary dependencies are encoded in DDT by means of the simplified representation described in Remark 2.85 — ie, in order to avoid filler nodes, DDT uses a scheme where the relation between the filler source and the filler governor is indicated by means of a secondary edge from the filler governor to the filler source, with square brackets around the dependency label in order to indicate that the dependency involves a hidden filler, as shown in Figure 4.64. Similarly, syntactically determined coreference is encoded in DDT by means of a ref edge from the antecedent to the anaphor, as shown in Figure 4.65. Finally, dependencies between a word and an elided node in a gapping coordination is encoded by means of a gapping dependency, represented by an edge whose edge label indicates the path (represented as a colon-separated sequence of dependency types) from the

| Complement edges | | Adjunct edges | |
|---|---|---|---|
| **aobj** | adjectival object | **appa** | parenthetical apposition |
| **avobj** | adverbial object | **appr** | restrictive apposition |
| **conj** | conjunct of coordinator | **coord** | coordination |
| **dobj** | direct object | **list** | unanalyzed sequence |
| **expl** | expletive subject | **mod** | modifier |
| **iobj** | indirect object | **modo** | dobj-oriented modifier |
| **lobj** | locative-directional object | **modp** | parenthetical modifier |
| **nobj** | nominal object | **modr** | restrictive modifier |
| **numa** | additive numeral | **mods** | subject-oriented modifier |
| **numm** | multiplicative numeral | **name** | additional proper name |
| **part** | verbal particle | **namef** | additional first name |
| **pobj** | prepositional object | **namel** | additional last name |
| **possd** | possessed in genitives | **pnct** | punctuation modifier |
| **pred** | subject or object predicate | **rel** | relative clause |
| **qobj** | quotation object | **title** | title of person |
| **subj** | subject | **xpl** | explification (colon) |
| **vobj** | verbal object | | |

Figure 4.63: The most important edge labels in DDT.

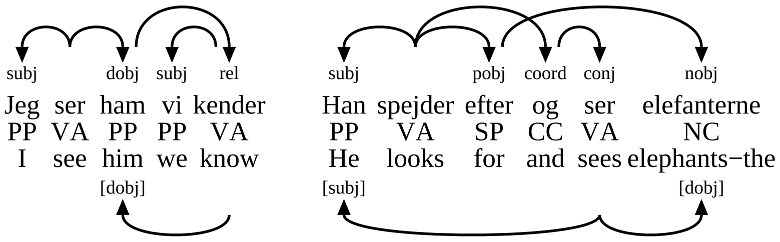


Figure 4.64: Annotation of secondary dependencies in DDT.

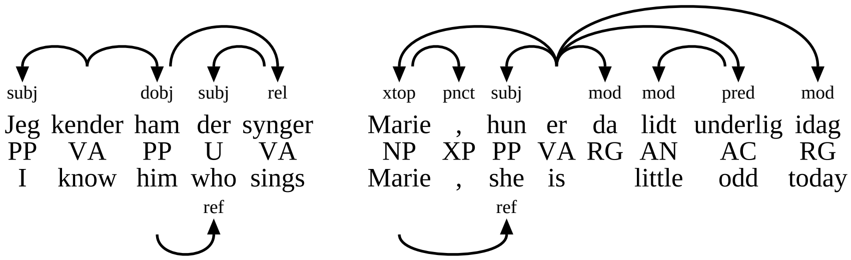


Figure 4.65: Annotation of syntactically determined coreference in DDT.

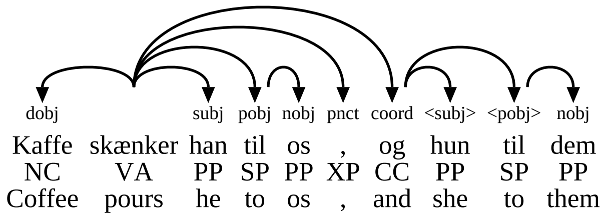

dobj
subj
pobj
nobj
pnct
coord
<subj>
<pobj>
nobj
Kaffe skænker han til os , og hun til dem
NC VA PP SP PP XP CC PP SP PP
Coffee pours he to os , and she to them

Figure 4.66: Annotation of gapping dependencies in DDT.

gapping source to the governor source in the source tree for the gap, as shown in Figure 4.66; the angular brackets are used to distinguish gapping dependencies from primary dependencies.[16]

**Remark 4.95** Penn Treebank II (Marcus et al. 1994) also encodes filler and syntactic coreference dependencies, but most other treebanks leave them unspecified, including the Prague Dependency Treebank (Böhmová et al. 2001), the Tiger/Negra Treebank (Brants et al. 2002) which only encodes secondary fillers in coordinations, and the Danish Arboretum (Bick 2003).

**Remark 4.96** DDT has been based on a 110 page annotation manual (Kromann et al. 2003), which is unfortunately still incomplete. The manual describes the general principles behind the DDT annotation and the specific annotations of a wide range of phenomena. The annotation manual provides analyses for a wide range of phenomena that have not been addressed in this dissertation, such as proper names (with first names, last names, and titles), parenthetical and restrictive appositions, possessives, expletives, vocatives, external topics (resumptive pronouns), tag questions, clefts, and many others.

There are no significant differences between the analyses in the annotation manual and the analyses in the present dissertation, except for coordinations where we have changed our analysis: in DDT, the second conjunct in a coordination is headed by the coordinator, whereas in this dissertation, the coordinator is analyzed as a modifier of the second conjunct.

[16]The Prague Dependency Treebank (Böhmová et al. 2001) and the Tiger/Negra Treebank (Brants et al. 2002) analyze coordinators as heads of the coordination. However, since the coordinator and the second conjunct form a unit in discontinuous coordinations such as "I will serve Mary coffee, and John", our analysis of the first conjunct as the head seems preferable.

| **Label** | **A1** | **A2** | **Agreed** | **Agreement** |
|---|---|---|---|---|
| nobj | 93 | 94 | 93 | 99.5 |
| mod | 83 | 82 | 69 | 83.6 |
| pnct | 60 | 60 | 59 | 98.3 |
| subj | 46 | 46 | 46 | 100.0 |
| vobj | 28 | 27 | 27 | 98.2 |
| pobj | 28 | 21 | 20 | 81.6 |
| dobj | 22 | 22 | 18 | 81.8 |
| [subj] | 23 | 17 | 17 | 85.0 |
| conj | 13 | 14 | 13 | 96.3 |
| possd | 13 | 13 | 13 | 100.0 |
| ref | 11 | 11 | 11 | 100.0 |
| pred | 11 | 10 | 10 | 95.2 |
| coord | 10 | 10 | 9 | 90.0 |
| rel | 10 | 10 | 10 | 100.0 |
| lobj | 5 | 14 | 5 | 52.6 |
| namef | 7 | 7 | 7 | 100.0 |
| part | 6 | 3 | 3 | 66.7 |
| qobj | 3 | 3 | 3 | 100.0 |
| **total labelled** | 483 | 474 | 440 | 92.0 |
| **total unlabelled** | 483 | 474 | 463 | 96.8 |

Figure 4.67: Inter-annotator agreement for the DDT annotation scheme, with statistics for the most frequent edge types.

**Remark 4.97** On the basis of three independently annotated text samples with a total of 483 tokens, the labelled inter-annotator agreement with the DDT scheme is estimated to be around 92%, and the unlabelled inter-annotator agreement is estimated to be around 97% (the estimates are somewhat unreliable because of the small size of the test corpus). Figure 4.67 shows the detailed statistics for the most frequent edge types, and the total sums. The "Label" column shows the edge label, "A1" and "A2" the number of edges produced by the two annotators with the given label, "Agreed" the number of edges they agreed on, and "Agreement" the inter-annotator agreement, calculated by means of the formula:

$$\text{Agreement} = \frac{2 \cdot \text{Agreed}}{\text{A1} + \text{A2}}.$$

Unsurprisingly, the inter-annotator agreement is lowest for dependency

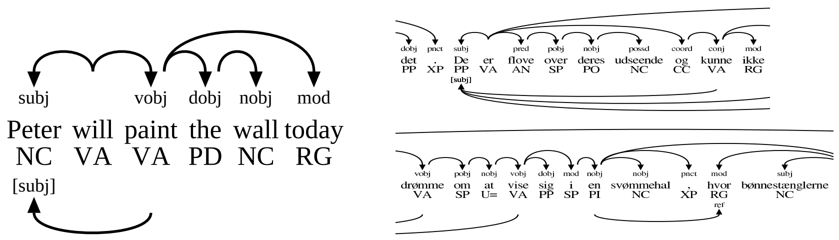

subj vobj dobj nobj mod
Peter will paint the wall today
NC VA VA PD NC RG
[subj]
dobj pnct subj pred pobj nobj possd coord conj mod
det , De er flove over deres udseende og kunne ikke
PP XP PP VA AN SP PO NC CC VA RG
[subj]
vobj pobj nobj vobj dobj mod nobj nobj pnct mod subj
drømme om at vise sig i en svømmehal , hvor bønnestænglerne
VA SP U= VA PP SP PI NC XP RG NC
ref

Figure 4.68: Visualizations of graphs with DTAG.

types such as prepositional objects (pobj), locative-directional objects (lobj), and modifiers (mod) where it is difficult to make a principled distinction between complement and adjunct constructions, and where attachment ambiguity is the rule rather than the exception.

**Remark 4.98** The DDT annotations have been created with the DTAG treebank tool (Kromann 2005). DTAG can handle any syntax graph consisting of nodes connected by directed labeled edges. The nodes may correspond to words, phrases, or fillers, and may have any number of associated attributes for encoding word class, lemma, etc. The labeled edges may form any graph structure, including trees, discontinuous graphs, cyclic graphs, and graphs where nodes may have any number of in-coming or out-going edges. In this respect, DTAG allows a wider range of graphs than most other treebank tools, including the tools from the Tiger/Negra treebank project (König et al. 2003).

DTAG displays syntax graphs continuously by generating a PostScript file which is viewed by a PostScript viewer, and which can also be included in text files. Nodes are drawn in linear order, with different attributes shown on separate lines. Directed edges are drawn above or below the nodes, with the edge label shown at the arrow head, as in Figure 4.68 left. DTAG can print graphs corresponding to entire texts by splitting the graphs into several lines and pages, as in the excerpt in Figure 4.68 right. DTAG can also be used to print multi-speaker dialogue and discourse annotations. The precise formatting (colors, dashes, placement of edges and attributes, etc.) can be configured by the user.

DTAG is command-line based. For example, to specify a directed edge from node 34 to node 45 with label ″subj″, the user must type the command ″34 subj 45″ (DTAG allows any string without white-space to be used as a label). DTAG also has commands for comparing two graphs and visualizing the differences, and for performing search-and-replace within all texts in the treebank. A search is specified as a constraint on a set of node variables. Simple constraints specify conditions on a node's associated attributes, or the relative node order or connecting edges of two nodes, and can be combined into complex constraints by the logical operators 'and', 'or', 'not', 'exist', and 'all'; the use of 'exist' and 'all' means that DTAG's query language is slightly more powerful than TIGERSearch (König et al. 2003). For example, the following DTAG command is used to find all instances in the treebank where a verb $V has a "der" ("there") expletive $E, but no direct object $D:

```
find -corpus ($E expl $V) & ($E[lemma] =~ /^der$/)
     & !  exist($D, $D dobj $V)
```

The entire set of DTAG commands is described in (Kromann 2005). The file format used by DTAG is described in (Kromann et al. 2003).

In this section, we have described the Danish Dependency Treebank and its relationship to DG and DTAG. The creation of DDT shows that it is possible to use DG to analyze a large corpus, even though there are many areas where the DDT analyses can be refined (eg, by providing more detailed classifications of modifiers, by annotating discourse dependencies and functor-argument structures, and by extending the annotation manual to cover a larger number of phenomena). The DDT has been a major motivation for our work on statistical language learning, since it provides rich statistical training material, a topic we will return to in chapter 6.

## 4.9 Summary

In this chapter, we have proposed specific DG analyses for a wide range of natural language phenomena, including word order phenomena such as topicalizations, extrapositions, and scramblings; filler phenomena such as control constructions, relatives, and parasitic gaps; coordination phenomena such as sharing and gapping; punctuation phenomena; morphological phenomena such as inflections, derivations, and lexical alternations; and the structure of discourse, spoken language, and dialogue. We have also argued that the DG formalism is in principle expressive enough to give a rule-based account of these phenomena, and we have briefly outlined some of the lexical rules and cost functions needed in a rule-based account.

Given the small number of pages we have been able to devote to each phenomenon, there are undoubtedly many important aspects that we have overlooked, particularly with respect to the manually written cost functions we have proposed to account for the phenomena. However, the chapter has at least shown that a rule-based account of a language requires a highly complex grammar with a large number of rules. For that reason, we will argue in chapter 6 that it is better to reduce the complexity by using a probabilistic language model where the rules are induced by a probabilistic learning algorithm, rather than specified manually.

# Part III

# Statistical language learning

# Chapter 5

# Statistical estimation with hierarchical partition models

Statistical estimation lies at the heart of probabilistic language learning, and it is therefore essential to develop estimation algorithms that can take advantage of the classification hierarchies associated with language data. In this chapter, which is based on (Kromann 2004), we propose a statistical estimation algorithm that can be used to estimate probability distributions for hierarchically organized data. In section 5.1, we introduce some prerequisites from probability theory and statistics. In section 5.2, we define our notion of hierarchies and show how hierarchies can be combined by means of product and intersection hierarchies. In section 5.3, we propose HPM and XHPM models as a general family of probability distributions that can be used to model random variables with hierarchically organized data, including random vectors where we want to calculate conditional probalities with non-hierarchical components. In section 5.4, we propose an estimation algorithm based on local search that constructs an HPM or XHPM model that maximizes the posterior Bayesian probability of the model, using a prior model probability based on the Minimum Description Length principle. Finally, in section 5.5, we present a simulation study that shows that the algorithm performs well in a wide range of situations.

## 5.1 Prerequisites from probability theory and statistics

***Summary***. *We introduce the most important concepts from probability theory and statis-*

*tics: probability measures, random variables, distributions, densities, maximum-likelihood estimation, and Bayesian estimation.*

Probability theory is concerned with modelling repeatable random processes, such as the throwing of a dice or the random selection of a word in a treebank. In probability theory, a random process is modelled by means of a probability measure, which quantifies how often the random process can be expected to produce a given outcome. Probability measures are defined formally below (cf. Rice 1988; Halmos 1974/1950):

**Definition 5.1** Let $S$ be a set, and let $\mathcal{A}$ be a set of subsets of $S$. Then $S$ is called a *measurable space* with *measurable sets* $\mathcal{A}$ provided (a) $\mathcal{A}$ contains $\emptyset$ and $S$; (b) $\mathcal{A}$ is closed under set difference, ie, if $A, B \in \mathcal{A}$, then $A - B \in \mathcal{A}$; and (c) $\mathcal{A}$ is closed under countable unions, ie, if $A_i \in \mathcal{A}$, then $\bigcup_{i=1}^{\infty} A_i \in \mathcal{A}$.

**Definition 5.2** Let $S, S'$ be measurable spaces. A function $f\colon S \rightarrow S'$ is called *measurable* if the inverse image $f^{-1}(A')$ of any measurable set $A'$ in $S'$ is a measurable set in $S$.

**Definition 5.3** Let $S$ be a measurable space with measurable sets $\mathcal{A}$. A function $\mu\colon \mathcal{A} \rightarrow [0, \infty]$ is called a *measure* on $S$ if $\mu(\emptyset) = 0$ and $\mu(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} \mu(A_i)$ for any sequence of mutually disjoint $A_i \in \mathcal{A}$. If $\Omega$ is a measurable space with measure $P$ satisfying $P(\Omega) = 1$, we will say that $\Omega$ is a *probability space* with *probability measure* (or *distribution*) $P$, and we will refer to $P(A)$ as the *probability* of the measurable set $A$.

**Example 5.4** In this dissertation, $\Omega$ will usually be a finite set, and $\mathcal{A}$ will usually be the set of all subsets of $\Omega$. In that case, we can write:

$$P(A) = \sum_{\omega \in A} P(\{\omega\})$$

for any subset $A$ of $\Omega$. We will say that the distribution $P$ is *uniform* if $P(\{\omega\}) = 1/|\Omega|$ where $|\Omega|$ denotes the number of elements in $\Omega$, and *non-zero* if $P(\{\omega\})$ is non-zero for all $\omega \in \Omega$.

Random variables can be viewed as functions that classify outcomes in terms of some classification, ie, they map outcomes to classifications. We follow standard practice by defining the notation and terminology below.

**Definition 5.5** A *random variable* is a measurable function $X\colon \Omega \to \Omega'$ from a probability space $\Omega$ with distribution $P$ into a measurable space $\Omega'$. $X$ induces a distribution $P_X$ on $\Omega'$ given by:

$$P_X(A') = P(X^{-1}(A')) = P(\{\omega \in \Omega \mid X(\omega) \in A'\})$$

for every measurable set $A'$ in $\Omega'$. $P_X$ is called the *distribution* associated with $X$, and the set $\Omega'$ is called the *possible outcomes* of $X$. As a notational convention, we often write $P(X)$ instead of $P_X$, $P(X = x)$ instead of $P_X(\{x\})$, $P(X \in A')$ instead of $P_X(A')$, and $\Omega(X)$ instead of $\Omega'$.

**Definition 5.6** Let $\Omega$ be a probability space with probability measure $P$. If $X_1, \ldots, X_n$ are random variables $X_i\colon \Omega \to \Omega_i'$, we let $X = (X_1, \ldots, X_n)$ denote the random variable $X\colon \Omega \to \Omega_1 \times \cdots \times \Omega_n$ defined by $X(\omega) = (X_1(\omega), \ldots, X_n(\omega))$. The composite random variable $X$ is often called a *random vector*. By convention, we write $P(X_1, \ldots, X_n)$ instead of $P_{(X_1,\ldots,X_n)}$, and $P(X_1 = x_1, \ldots, X_n = x_n)$ instead of $P_{(X_1,\ldots,X_n)}(\{(x_1, \ldots, x_n)\})$.

**Example 5.7** A fair dice with the six equally probable outcomes $1, \ldots, 6$ can be modelled by a probability space $\Omega = \{1, 2, 3, 4, 5, 6\}$ with distribution $P$ satisfying $P(\{\omega\}) = 1/6$ for all outcomes $\omega \in \Omega$ (ie, $P$ is the uniform distribution on $\Omega$). Let $X$ be the random variable that maps $\Omega$ into the set $\Omega' = \{\text{odd}, \text{even}\}$ according to whether the outcome is odd or even. We can then compute $P(X = \text{odd}) = P(X^{-1}(\{\text{odd}\})) = P(\{1, 3, 5\}) = P(\{1\}) + P(\{3\}) + P(\{5\}) = 1/2$.

**Example 5.8** The process of randomly selecting a word instance from a treebank can be modelled by means of a probability space $\Omega$ with probability measure $P$, where $\Omega$ consists of all word instances in the treebank, and $P$ assigns probability $P(\{\omega\}) = 1/|\Omega|$ to each word instance $\omega$ in $\Omega$. The word, word class, and ontological class associated with each word instance can be modelled by means of three random variables $W\colon \Omega \to \mathcal{W}$, $C\colon \Omega \to \mathcal{C}$, and $O\colon \Omega \to \mathcal{O}$ that map each word instance to its associated word, word class, or ontological class. We can then express the probability of observing the word $w$ as $P(W = w)$, and the probability of observing a word with word class $c$ and ontological class $o$ as $P(C = c, O = o)$.

It is often convenient to define distributions by means of densities.

**Definition 5.9** Let $S$ be a measurable space with measure $\mu$, and let $f\colon S \rightarrow [0,\infty[$ be a function with $\int f\, d\mu = 1$. Then $f$ is called the *density* for the distribution $F$ on $S$ defined by $F(A) = \int_A f\, d\mu$. As a notational convenience, we sometimes write $P_f$ instead of $F$ for the distribution induced by $f$.

For the definition of integration within measure theory, we refer the interested reader to (Halmos 1974/1950). As an illustration, we will consider two special cases of densities.

**Remark 5.10** Let $\mathbb{R}$ be the set of real numbers, and let $\mu$ be the standard measure on $\mathbb{R}$ defined by $\mu(]a,b[) = b-a$. Then any function $f\colon \mathbb{R} \rightarrow [0,\infty[$ with $\int f(x)\, dx = 1$ is a density with distribution $F(A) = \int_A f(x)\, dx$. Moreover, $f$ is the derivative of $F$, ie, $f(x) = F'(x)$.

**Remark 5.11** Let $S$ be a countable set with the *point measure*, ie, $\mu(\{s\}) = 1$ for all $s \in S$. Any function $f\colon S \rightarrow [0,\infty[$ with $\int_S f\, d\mu = \sum_{s\in S} f(s) = 1$ then defines a density with distribution $F$ given by

$$F(A) = \int_A f\, d\mu = \sum_{s\in A} f(s).$$

Since $f$ can be recovered from $F$ by means of the identity $f(a) = F(\{a\})$, any distribution on $S$ can be expressed by means of a density.

In statistical analysis, we are given the observed outcomes of a random variable $X$ where the underlying distribution $P(X)$ is unknown, and must reconstruct $P(X)$ by making informed estimates of $P(X)$ on the basis of the observed outcomes. That is, we must construct estimates of $P(X)$ that could plausibly have generated the observed outcomes. $P(X)$ can be estimated in many different ways. The conceptually simplest estimator is the empirical estimator defined below, where the estimated probabilities are simply the observed frequencies of each outcome.

**Definition 5.12** Let $d = (d_1, \ldots, d_n)$ be a sequence of *observed outcomes* of a random variable $X$. Then the *empirical estimator* $\hat{f}_d^{\mathrm{emp}}$ corresponding to $d$ is the density defined by:

$$\hat{f}_d^{\mathrm{emp}}(x) = \frac{|d \cap \{x\}|}{|d|} = \frac{\text{number of observations of } x}{\text{total number of observations}}$$

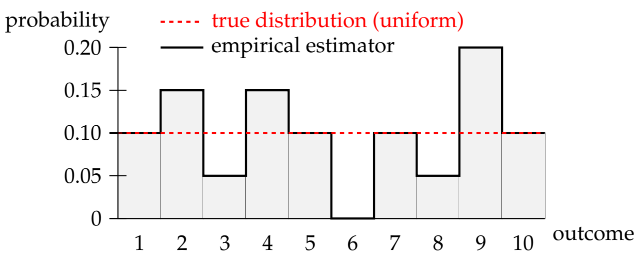


Figure 5.1: The empirical estimator is unreliable when there are few observations.

In theory, the empirical estimator corresponding to a sequence of observations of a random variable $X$ with a countable probability space converges towards $P(X)$ with probability 1 as the number of observations increases. However, the empirical estimator is an unreliable estimator of $P(X)$ when there are few observations, or if the space of outcomes is infinite. First of all, it tends to *overfit* the data, ie, it uses a complicated density function to capture statistically insignificant aspects of the observations, instead of using a simpler density function which only captures statistically significant aspects. Secondly, the empirical estimator assigns zero probability to outcomes that were not observed in the data sequence by pure chance, ie, it predicts that these outcomes are impossible. This failing, known as the *sparse data problem*, is especially serious if there are few observations, or if some outcomes have a very low probability of occurring.

**Example 5.13** The 20 observations in Fig. 5.1 are generated by a uniformly distributed random variable with 10 possible outcomes. The true distribution is shown as a dotted line, the empirical estimator is shown as a solid line. The empirical estimator is unnecessarily complicated (overfitting), and predicts that the outcome 6 is impossible (the sparse data problem).

To find an estimator $\hat{f}$ that solves the problems with overfitting and zero probabilities, we must make simplifying assumptions about the shape of the underlying density $f$. Classical statistical methods assume that $f$ belongs to a pre-defined family $\mathcal{F}$ of density functions (eg, the family of normal distributions or $\chi^2$-distributions), and selects $\hat{f}$ as the member of $\mathcal{F}$ that optimizes some quantity, usually likelihood, defined formally below.

**Definition 5.14** Let $d$ be a sequence of observations of a random variable

$X$, and let $f$ be any density on $\Omega(X)$. Then the *likelihood* $L_f(d)$ of the data $d$ given density $f$ is defined by:

$$L_f(d) = \prod_{d_i \in d} f(d_i)$$

If $\mathcal{F}$ is a family of densities, then a density $\hat{f} \in \mathcal{F}$ is called a *maximum likelihood estimator* for the data $d$ within the family $\mathcal{F}$ if $\hat{f}$ maximizes $L_f(d)$, ie, if:

$$\hat{f} = \arg\max_{f \in \mathcal{F}} L_f(d) \qquad (\text{ie, } L_{\hat{f}}(d) = \max_{f \in \mathcal{F}} L_f(d))$$

The following proposition shows that the maximum likelihood estimator within the family of all possible density functions coincides with the empirical estimator when there are only finitely many possible outcomes.

**Proposition 5.15** *Let $d$ be a set of observations of a random variable with possible outcomes $\Omega$ where $\Omega$ is finite, and let $\mathcal{F}$ be the family consisting of all density functions on $\Omega$. Then the maximum likelihood estimator of $d$ within $\mathcal{F}$ is unique and equals the empirical estimator $\hat{f}_d^{\mathrm{emp}}$.*

*Proof.* Write $\Omega = \{\omega_1, \ldots, \omega_m\}$, $d = \{d_1, \ldots, d_n\}$, $n_i = |\{d_j \in d \mid d_j = \omega_i\}|$, and let $f(\omega_i) = f_i / \sum_{j=1}^m f_j$ with $f_1, \ldots, f_m \in [0, \infty[$ denote the maximum likelihood estimator of the data $d$. Write:

$$L_f(d) = \prod_{i=1}^m \left( \frac{f_i}{\sum_{j=1}^m f_j} \right)^{n_i}.$$

Applying the logarithm on both sides gives:

$$l_f(d) = \log L_f(d) = \sum_{i=1}^m n_i \log f_i - n \log \left( \sum_{j=1}^m f_j \right).$$

In order to maximize $l_f(d)$, we must have $\frac{\partial l_f(d)}{\partial f_i} = 0$ for all $i = 1, \ldots, m$, ie, we must have:

$$\frac{\partial l_f(d)}{\partial f_i} = \frac{n_i}{f_i} - \frac{n}{\sum_{j=1}^m f_j} = 0 \quad \text{so that} \quad f(x_i) = \frac{f_i}{\sum_{j=1}^m f_j} = \frac{n_i}{n}.$$

This shows that $f$ equals the empirical distribution. □

An alternative to the classical statistical methods is to use Bayesian methods, where the density $f$ and the data $d$ are viewed as random variables $F$ and $D$. Assuming that the observations in $D$ are statistically independent given $F$ and that $P(D_i = d_i | F = f) = f(d_i)$, we get:

$$P(D = d|F = f). = \prod_{i=1}^{n} P(D_i = d_i|F = f) = \prod_{i=1}^{n} f(d_i) = L_f(d).$$

Thus, from a Bayesian point of view, maximum likelihood estimators maximize the probability $P(D = d|F = f)$ of the data $d$ given the model $f$. However, the data $d$ are given in advance whereas the model $f$ is unknown, so from a Bayesian point of view, it makes more sense to optimize the probability $P(F|D)$ of the model given the data, instead of optimizing the probability $P(D|F)$ of the data given the model. To calculate $P(F|D)$, we observe that $P(D = d)$ does not depend on $f$, so that Bayes' law[1] gives:

$$\begin{aligned}\arg\max_{f\in\mathcal{F}} P(F = f|D = d) &= \arg\max_{f\in\mathcal{F}} \frac{P(D = d|F = f)P(F = f)}{P(D = d)} \\ &= \arg\max_{f\in\mathcal{F}} P(D = d|F = f)P(F = f) \\ &= \arg\max_{f\in\mathcal{F}} L_f(d) \cdot P(F = f)\end{aligned}$$

Thus, a Bayesian estimator maximizes the likelihood $L_f(d)$ multiplied with the prior probability $P(F = f)$ of the density $f$. Bayesian estimators are defined formally below:

**Definition 5.16** Let $d$ be a sequence of observations of a random variable $X$, let $\mathcal{F}$ be a family of densities, and let $P(F)$ be a distribution on $\mathcal{F}$. Then a density $\hat{f} \in \mathcal{F}$ is called a *Bayesian estimator* within the family $\mathcal{F}$ given *prior distribution* $P(F)$ if it maximizes the quantity $L_f(d) \cdot P(F = f)$, ie, if:

$$\hat{f} = \arg\max_{f\in\mathcal{F}} L_f(d) \cdot P(F = f)$$

[1]The *conditional probability* $P(X|Y)$ of the outcome of the random variable $X$ given the outcome of the random variable $Y$ is defined as $P(X|Y) = P(X,Y)/P(Y)$. By applying this definition twice, we can deduce *Bayes' law*, which states that:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}.$$

**Remark 5.17** A maximum likelihood estimator is a special case of a Bayesian estimator, namely the case where $P(F)$ is uniform. For suppose $P(F)$ is uniform. Then $P(F = f) = P(F = f')$ for all $f, f' \in \mathcal{F}$ so that:

$$\arg\max_{f\in\mathcal{F}} L_f(d) \cdot P(F = f) = \arg\max_{f\in\mathcal{F}} L_f(d).$$

**Remark 5.18** In Bayesian estimation, we are often given a weight function $W\colon \mathcal{F} \to [0, \infty[$ rather than a prior distribution $P(F)\colon \mathcal{F} \to [0, 1]$. The weight can be interpreted as a *relative probability*, ie, if $f_1$ and $f_2$ are distributions in $\mathcal{F}$, we can interpret $f_1$ as being $W(f_1)/W(f_2)$ times more likely than $f_2$. If $\mathcal{F}$ is finite, we can turn $W$ into a distribution $P(F)$ by normalization:

$$P(F = f) = \frac{W(f)}{W_0} \qquad \text{where} \qquad W_0 = \sum_{f'\in\mathcal{F}} W(f')$$

A Bayesian estimator $\hat{f}$ given prior weight $W$ (and hence prior distribution $P(F)$) then satisfies:

$$\hat{f} = \arg\max_{f\in\mathcal{F}} L_f(d) \cdot \frac{W(f)}{W_0} = \arg\max_{f\in\mathcal{F}} L_f(d) \cdot W(f)$$

since $W_0$ is constant, ie, we can compute the Bayesian estimator corresponding to a weight function $W$ without normalizing $W$ first. If $\mathcal{F}$ is infinite, $W_0$ may not be defined because the sum does not converge to a finite value. In this setting, we will interpret $W$ as a function that expresses the relative probability of different outcomes, and say that $\hat{f}$ is a Bayesian estimator within the family $\mathcal{F}$ with *prior weight* (or *prior relative probability*) $W$ if $\hat{f}$ maximizes $L_f(d) \cdot W(f)$.

The biggest and most controversial problem in Bayesian analysis is how to define the prior distribution $P(F)$ or prior weight $W(F)$. Obviously, $W(F)$ cannot be estimated empirically, but must be defined theoretically. This means that Bayesian analysis is mostly used in cases where theoretical considerations lead to a natural estimate of $W(F)$. Maximum-likelihood estimation suffers from similar problems, since it depends on the choice of a predefined family $\mathcal{F}$ of distributions. It is therefore mostly used in cases where theoretical considerations lead to a natural choice of $\mathcal{F}$, and where the uniform distribution is the most natural prior distribution on $\mathcal{F}$.

In this section, we have introduced some of the most important ideas from probability theory and statistical estimation. In the following section, we define hierarchies before turning to the important question of how to estimate probability distributions for hierarchically organized data.

## 5.2 Hierarchies

***Summary**. We define hierarchies as a way of abstracting over observations in a treebank. We also show how hierarchies can be combined to form product hierarchies, restriction hierarchies, and intersection hierarchies. We also show how a probability distribution on the classes in a hierarchy can be extended to class unions, intersections, and differences.*

Hierarchies play an important role in linguistics because they are used for lumping together similar phenomena and expressing generalizations. They are also the basis for statistical language modelling. Intuitively, a hierarchy consists of a set of classes organized into a tree structure — or, more generally, a directed acyclic graph structure — by means of an immediate subclass relation. Hierarchies are defined formally below.

**Definition 5.19** Let $C$ be a set whose elements are called *classes*, and let $H\colon C \rightarrow \mathrm{Pow}(C)$ be a function that maps a class $c$ into the set $H(c)$ of *immediate subclasses* of $c$ in $H$. Given two classes $s, c \in C$, we say that $s$ is a *subclass* of $c$ in $H$ if there exists classes $c_1, \ldots, c_n$ in $C$ with $n > 1$ such that $c_1 = c$, $c_n = s$, and $c_{i+1}$ is an immediate subclass of $c_i$ for each $i = 1, \ldots, n-1$. The function $H$ is called a *hierarchy* with classes $C$ if $H$ satisfies the following three conditions:

- *Unique root condition*: There is a unique class $\top$ in $H$, called the *root class* of $H$, such that $\top$ is not a subclass of any other class in $H$.
- *Acyclicity condition*: No class in $H$ is a subclass of itself.
- *Connectedness condition*: Every class in $H$ is a subclass of $\top$, except for the root class $\top$ itself.

The set $C$ of classes in $H$ is sometimes denoted by $H^{\mathrm{class}}$. The set consisting of a class $c$ and all its subclasses in $H$ is denoted by $H^*(c)$. A class $c$ in $H$ is called a *terminal class* if $H(c)$ is empty and a *non-terminal class* if $H(c)$ is non-empty. We let $H^{\mathrm{term}}$ denote the set of terminal classes in $H$, we let $H^{\mathrm{nterm}}$ denote the set of non-terminal classes in $H$, and we let $H^{\mathrm{term}}(c) = H^{\mathrm{term}} \cap H^*(c)$ denote the set of terminal classes corresponding to a class $c$ in $H$. The hierarchy $H$ is sometimes said to be *defined on* $H^{\mathrm{term}}$.

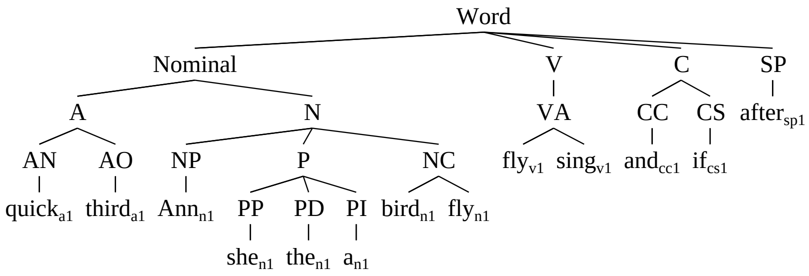


Figure 5.2: A word class hierarchy.

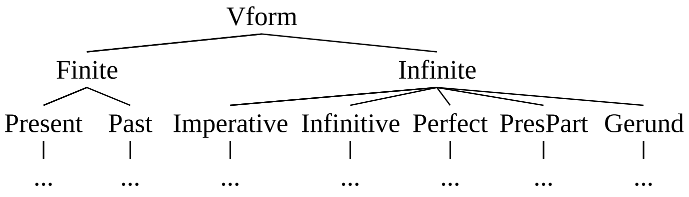


Figure 5.3: An inflectional hierarchy.

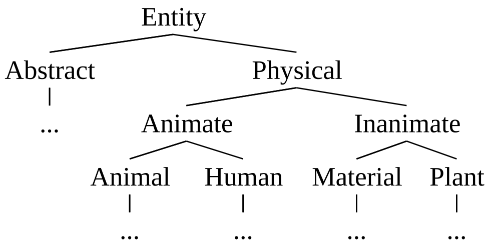


Figure 5.4: An ontological hierarchy.

**Example 5.20** Hierarchies are drawn in the usual way, ie, with superclasses at the top, terminal classes at the bottom, and with lines connecting classes with their immediate subclasses. For example, Figure 5.2 encodes a word-class hierarchy where $H(\mathrm{N}) = \{\mathrm{NP}, \mathrm{P}, \mathrm{NC}\}$, $H(\mathrm{VA}) = \{\mathrm{fly_{v1}}, \mathrm{sing_{v1}}\}$, $H^*(\mathrm{C}) = \{\mathrm{C}, \mathrm{CC}, \mathrm{CS}, \mathrm{and_{cc1}}, \mathrm{if_{cs1}}\}$, and $H^{\mathrm{term}}(\mathrm{C}) = \{\mathrm{and_{cc1}}, \mathrm{if_{cs1}}\}$. Hierarchies can also be used to encode inflectional hierarchies for number, gender, definiteness, diathesis, verb form, etc. (as in Figure 5.3), or to encode ontological hiearchies (as in Figure 5.4). Many ontologies are non-trees, including the large-scale ontology WordNet (Fellbaum 1998).

In many cases, we are particularly interested in hierarchies that are equipped with an intersection operation.

**Definition 5.21** A hierarchy $H$ with classes $C$ and an associated operation $\cap' \colon C_\emptyset \times C_\emptyset \to C_\emptyset$ on $C_\emptyset = C \cup \{\emptyset\}$ is called an *intersective hierarchy* with *class intersection operation* $\cap'$ if

$$H^*(a \cap' b) = H^*(a) \cap H^*(b)$$

for all $a, b \in C_\emptyset$, where $\cap$ denotes the normal set intersection operation and we define $H^*(\emptyset) = \emptyset$.

When estimating a probability distribution for a random vector with categorical data, we often have many natural hierarchies for each individual random variable in the random vector. For example, lexemes can be organized in word class hierarchies, inflectional hierarchies, and ontological hierarchies. We therefore need a way of combining several different hierarchies for a lexeme into a single hierarchy, and for combining the hierarchies associated with each individual random variable in a random vector into a single hierarchy for the entire random vector. The product hierarchy defined below is a natural way to combine many hierarchies into a single higher-dimensional hierarchy (we omit the proof that the product hierarchy is a hierarchy, and that the product operation is associative).

**Definition 5.22** Let $H_1, \ldots, H_n$ be hierarchies, and let $C_i$ denote the set of classes in $H_i$. The *product hierarchy* $H = H_1 \times \cdots \times H_n$ with classes $C = C_1 \times \cdots \times C_n$ is then defined as the function $H \colon C \to \mathrm{Pow}(C)$ that returns all classes $s \in C$ that are identical to $c$, except for one coordinate $c_i$ which has been replaced with an immediate subclass $s_i$, ie, $H$ is given by:

$$H(c) = \{s \in C \mid \exists i\colon\ s_i \in H_i(c_i) \text{ and } \forall j \neq i \colon s_j = c_j\}$$

where $c_i$ denotes the $i$th coordinate in the tuple $c$. If $H_1, \ldots, H_n$ are intersective, then $H$ is also intersective with intersection operation

$$(a_1, \ldots, a_n) \cap (b_1, \ldots, b_n) = (a_1 \cap b_1, \ldots, a_n \cap b_n).$$

**Example 5.23** Let $H_1$ and $H_2$ be the two simple word class and number hierarchies shown in Figure 5.5. Their product is shown in Figure 5.6. Note that both $H_1$, $H_2$, and $H_1 \times H_2$ are intersective.

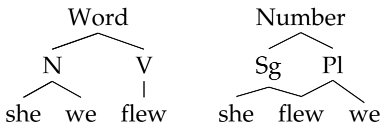


Figure 5.5: A simple word class hierarchy and number hierarchy.

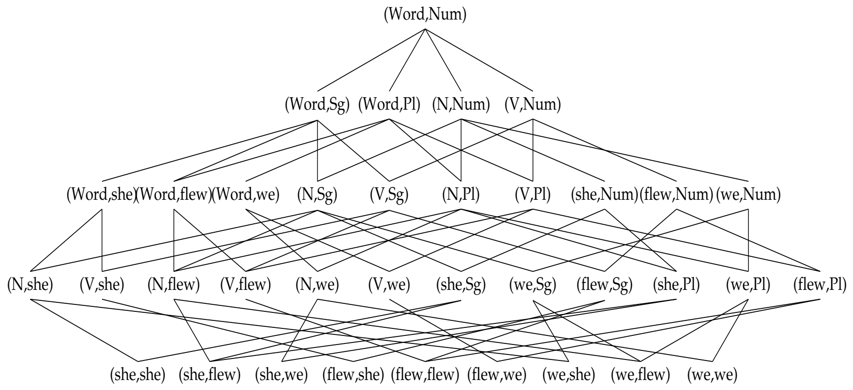


Figure 5.6: The product hierarchy of the two hierarchies in Figure 5.5.

The restriction hierarchy is another useful way of creating new hierarchies. Intuitively, the restriction hierarchy is the result of restricting a hierarchy to a given set of terminals (we omit the proof that the restriction hierarchy is a hierarchy).

**Definition 5.24** Let $H$ be a hierarchy, and let $T$ be a non-empty set of terminals contained in $H^{\text{term}}$. Then the *restriction hierarchy* $H/T$ is the hierarchy with classes $(H/T)^{\text{class}}$ given by:

$$(H/T)^{\text{class}} = \{c \in H^{\text{class}} \mid H^*(c) \text{ contains at least one member of } T\}$$

and function $H/T\colon (H/T)^{\text{class}} \to \text{Pow}((H/T)^{\text{class}})$ defined by:

$$H/T(c) = H(c) \cap (H/T)^{\text{class}}$$

**Remark 5.25** If $H$ is intersective, then $H/T$ is also intersective, using as its intersection operation the operation $\cap$ for $H$ restricted to $(H/T)^{\text{class}}$.

Finally, when we have more than one hierarchy for the same set of terminals $T$, we may want to combine the hierarchies into a single hierarchy for $T$. For example, we may have classified a set of lexemes by means of a word class hierarchy and an ontological hierarchy, and want to combine the two hierarchies into one. This can be accomplished with the intersection hierarchy defined below (being defined as the restriction of a product hierarchy, it is obviously a hierarchy).

**Definition 5.26** Let $H_1, \dots, H_n$ be hierarchies with the same set of terminals $T$, ie, $T = H_i^{\text{term}}$ for all $i$. The *intersection hierarchy* $H = H_1 \cap \ldots \cap H_n$ is defined as the hierarchy:

$$H = (H_1 \times \cdots \times H_n) / \iota(T)$$

where $\iota \colon T \to T^n$ is the function defined by $\iota(t) = (t, \dots, t)$. By identifying $T$ and $\iota(T)$ by means of $\iota$, we can say that $H$ defines a hierarchy on $T$. A class $(c_1, \dots, c_n)$ in the intersection hierarchy is often written as $c_1 \cap \ldots \cap c_n$.

Intersection hierarchies are particularly suited to combining multiple hierarchies for a random variable into a single hierarchy, whereas product hierarchies are more appropriate when creating a single hierarchy for a random vector on the basis of the hierarchies associated with the individual random variables in the random vector.

**Remark 5.27** If $H_1, \dots, H_n$ are intersective, then so is $H_1 \cap \ldots \cap H_n$, with the intersection operation defined by Definitions 5.22 and 5.24.

**Example 5.28** The intersection hierarchy for the two hierarchies in Figure 5.5 is shown in Figure 5.7. The intersection hierarchy is defined as the restriction of the product hierarchy in Figure 5.6 to the set of terminals consisting of (she,she), (flew,flew), and (we,we), where we identify (she,she) with "she," (flew,flew) with "flew," and (we,we) with "we." Because the two component hierarchies are intersective, so that both the product and restriction hierarchies are intersective, the intersection hierarchy is intersective as well.

When defining probability distributions over hierarchies, it is convenient to extend the set of classes to include finite unions, intersections, and differences of classes.

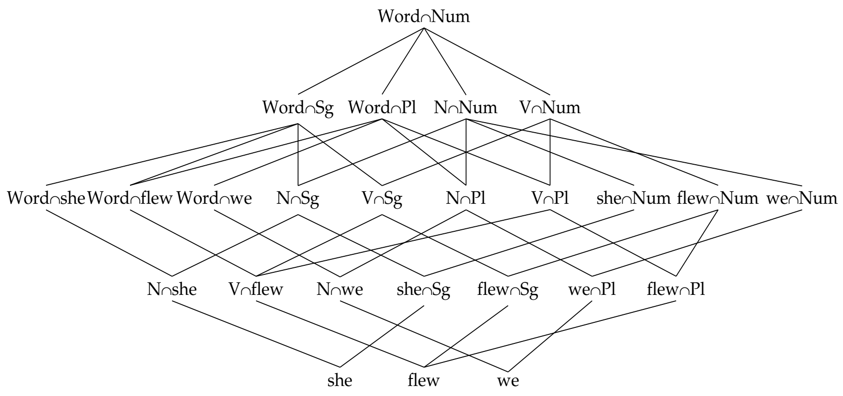


Figure 5.7: The intersection hierarchy of the two hierarchies in Figure 5.5.

**Definition 5.29** Let $H$ be a hierarchy with classes $C$. We define the *class algebra* $C^*$ as the smallest set of expressions satisfying:

(a) $\emptyset$ is an expression in $C^*$;
(b) $c$ is an expression in $C^*$ if $c \in C$;
(c) $c \cap c'$, $c \cup c'$, and $c - c'$ are expressions in $C^*$ if $c, c' \in C^*$.

We extend $H$ from $C$ to $C^*$ by recursively defining:

$$\begin{aligned} H(\emptyset) &= \emptyset \\ H(a \cap b) &= H(a) \cap H(b) \\ H(a \cup b) &= H(a) \cup H(b) \\ H(a - b) &= H(a) - H(b) \end{aligned}$$

Given $c, c' \in C^*$, we write $c = c'$ and say that $c$ and $c'$ are *equivalent* if $H(c) = H(c')$.

Given a probability distribution over the classes $C$ in a hierarchy, we can always compute a probability distribution over $C^*$ by the procedure described below.

**Remark 5.30** Let $H$ be an intersective hierarchy with classes $C$, and let $F$ be a distribution on $C$. Then $F$ induces a distribution $F^*$ on $C^*$ defined recursively by:

$$\begin{aligned}
F^*(\emptyset) &= 0 \\
F^*(c) &= F(c) && (\forall c \in C) \\
F^*(a-b) &= F^*(a) - F^*(a \cap b) && (\forall a,b,c \in C^*) \\
F^*(a \cup b) &= F^*(a) + F^*(b) - F^*(a \cap b) && (\forall a,b,c \in C^*).
\end{aligned}$$

Before we apply $F^*$ on an expression, we always ensure that the expression is intersection-free by applying the following reductions to it and by using the class intersection operation associated with $H$ to eliminate intersections between classes in $C$:

(a) map $a \cap (b-c)$ and $(b-c) \cap a$ to $(a \cap b) - (a \cap c)$ for all $a, b, c \in C^*$;
(b) map $a \cap (b \cup c)$ and $(b \cup c) \cap a$ to $(a \cap b) \cup (a \cap c)$ for all $a, b, c \in C^*$.

In general, this will result in a computation with exponential worst-case time complexity, a quite costly computation. However, the computation can be made reasonably efficient in most cases by applying further reductions, such as mapping $a \cap \emptyset$ to $\emptyset$, or $a \cup s$ to $a$ if $s$ is a subclass of $a$.

In this section, we have defined hierarchies and different ways of combining several different hierarchies into a single hierarchy, and we have described how to extend probability distributions over a set $C$ of classes to the set $C^*$ containing all finite unions, intersections, and set differences. In the following section, we will propose a family of statistical models that can be used to estimate probability distributions for hierarchically organized random variables given a data set.

## 5.3 Hierarchical Partition Models (HPMs and XHPMs)

***Summary**. We propose HPM and XHPM models as a general family of probability distributions that can be used to model random variables with hierarchically organized data, including random vectors where we want to calculate a conditional probability, and where some of the random variables follow arbitrary distributions. We show how any ordered cover in an HPM or XHPM model can be obtained by means of a sequence of partitionings and mergings of the entire space of outcomes. On the basis of the Minimum Description Length principle, we formulate a prior model probability that can be used to compare different HPM or XHPM models in Bayesian estimation, so that we can identify optimal estimates, ie, models that maximize the posterior probability given a data set, a set of hierarchies and prior distributions, and other estimation parameters.*

Most probabilistic language models use random vectors as their basic building blocks. For example, a random local tree $X \rightarrow Y_1 \ \dots \ Y_n$ in a PCFG can be viewed as a random vector $(X, Y_1, \dots, Y_n)$, a random complement frame with head $H$ and complements $C_1, \dots, C_n$ can be viewed as a random vector $(H, C_1, \dots, C_n)$, and a random adjunct frame with head $H$ and adjunct $A$ can be viewed as a random vector $(H, A)$. Given a treebank, we can obtain samples of a random vector $X = (X_1, \dots, X_n)$ from the treebank, and use the observations to estimate a probability distribution for $X$.

The empirical estimator is the simplest estimator of a random variable. It is a good estimator when there are few outcomes and all outcomes have been observed a reasonable number of times — the statistical literature often states 5 observations of each outcome as the minimum for reliable estimation. Unfortunately, the empirical estimator suffers from problems with overfitting and sparse data when there are many outcomes with few or no observations. Since treebanks usually contain only a tiny fraction of the possible word combinations in a language, treebank observations result in highly sparse data, ie, most word combinations are never observed. The empirical estimator is therefore unsuited to estimating treebank probabilities directly.

However, the problems with the empirical estimator can be overcome if we group similar outcomes into classes (or partitions), and observe classes of outcomes rather than individual outcomes. If the classes are made large enough to be observed at least 5 times, the empirical estimator can give a reliable estimate of the probability of the classes. In order to estimate the probability of individual outcomes, we must then distribute the estimated probability of a class among the individual outcomes in the class, using some prior probability distribution on the set of outcomes. In this way, we can estimate the probability of a class $c$ of outcomes by means of the estimator $\hat{F}$ defined by:

$$\hat{F}(c) = \sum_{s \in S} \overbrace{\frac{|d \cap s|}{|d|}}^{\substack{\text{empirical probability} \\ \text{of partition } s}} \cdot \overbrace{\frac{F^{\text{prior}}(c \cap s)}{F^{\text{prior}}(s)}}^{\substack{\text{prior probability of} \\ c \cap s \text{ within } s}}$$

where $d$ is the data set, $S$ is the set of partitions, and $F^{\text{prior}}$ is a prior distribution on the set of possible outcomes. This estimation method has two advantages: we can use the classification hierarchies to overcome the sparse data problem by making statistics on classes of outcomes rather than individual outcomes, and the use of a prior distribution allows us to create a complex statistical model as a chain of successive refinements to simpler statistical models. By using simple but robust priors, we get a model where the prior is used when there are few data, but where the model becomes increasingly more fine-grained when the amount of data increases.

When estimating a probability distribution on a hierarchically structured space of terminals, we have to partition the terminals into disjoint subsets so that we can compute the probability of each subset on the basis of observed frequencies. The disjoint subsets could be chosen to be disjoint classes in our hierarchy, as proposed by Li and Abe (1998, 1999). However, this strategy is somewhat impractical, for if we have a class $c$ and a subclass $s$, then the set difference $c - s$ does not normally correspond to a disjoint union of a few large classes in $H$, and we are not interested in disjoint unions of many small classes in $H$ because long unions are computationally expensive and small classes result in problems with sparse data. The obvious remedy is to pursue a strategy where the disjoint subsets in the partition may correspond to either classes or set differences between classes and finite unions of classes. The ordered covers defined below are a convenient way of representing partitions of this kind.

**Definition 5.31** Let $H$ be an intersective hierarchy with classes $C$. A sequence $o = (o_1, \ldots, o_n)$ of classes in $H$ is called an *ordered cover* of $H$ if every terminal in $H$ is contained in $H^{\text{term}}(o_i)$ for some $i$. The ordered cover $o$ partitions $H^{\text{term}}$ into $n$ disjoint *partitions* $\pi_{o,1}, \ldots, \pi_{o,n} \in C^*$ given by:

$$\pi_{o,k} = o_k - (o_1 \cup \ldots \cup o_{k-1}).$$

Partition $\pi_{o,i}$ is said to have $\pi_{o,j}$ as its *super partition* if $i < j$ and $o_i$ is a subclass of $o_j$; if $j$ is minimal in this respect, then $\pi_{o,j}$ is called an *immediate super partition* for $\pi_{o,i}$. A partition is said to be a *subpartition* of its super partitions, and a partition without super partitions is called a *root partition*. The class $o_i$ is sometimes denoted by $\bar{\pi}_{o,i}$.

Given a hierarchy, an ordered cover, a prior distribution, and a set of weights for each partition, we have an associated probability distribution.

**Definition 5.32** Let $H$ be an intersective hierarchy with classes $C$ and ordered cover $o = (o_1, \ldots, o_n)$, let $F$ be a prior distribution on $C^*$, and let $w = (w_1, \ldots, w_n)$ be a set of non-negative weights such that $\sum_{i=1}^{n} w_i F(\pi_{o,i}) = 1$. Then the tuple $M = (H, o, F, w)$ is called a *hierarchical partition model* (*HPM*) with hierarchy $H$, ordered cover $o$, prior distribution $F$, and weights $w$. The distribution $F_M$ on $C^*$ defined by:

$$F_M(c) = \sum_{i=1}^{n} w_i F(c \cap \pi_{o,i})$$

for all $c \in C^*$ is called the *HPM distribution* induced by the model $M$. The constant $w_i$ can be interpreted as the *average density* within partition $\pi_{o,i}$ within the measurable space induced by the prior distribution $F$.

The weights $w$ in a hierarchical partition model can be estimated from a data set by means of the empirical estimator.

**Remark 5.33** Let $H$ be a hierarchy with classes $C$, let $o$ be an ordered cover, and let $F$ be a prior density on $C^*$ such that $F(\pi_{o,i}) > 0$ unless $d \cap \pi_{o,i}$ is empty. A data set $d = (d_1, \ldots, d_n)$ then induces an HPM model $M_d = M_d(H, o, F) = (H, o, F, w)$ where $w$ is given by:

$$w_i = \frac{|d \cap \pi_{o,i}|}{|d|} \cdot \frac{1}{F(\pi_{o,i})}.$$

That is, we have:

$$F_{M_d}(c) = \sum_{i=1}^{n} \overbrace{\frac{|d \cap \pi_{o,i}|}{|d|}}^{\substack{\text{empirical probability} \\ \text{of partition } i}} \cdot \overbrace{\frac{F(c \cap \pi_{o,i})}{F(\pi_{o,i})}}^{\substack{\text{prior probability of} \\ c \cap \pi_{o,i} \text{ within } \pi_{o,i}}}.$$

The first quotient represents the proportion of the data set that falls within the given partition, and the second quotient represents the proportion of the partition that is contained in $c$. $M_d(H, o, F)$ is said to be the HPM model *induced* by $d$ given $H$, $o$, and $F$.

**Remark 5.34** The relation between a partition and its immediate super partitions defines a tree structure on the ordered cover, if the last class in the ordered cover is T. This tree structure can be used to compute $F_M(c)$ efficiently: instead of computing the sum over all partitions from left to right, we can compute it top-down, skipping all subpartitions of a partition $\pi_{o,i}$ where $c \cap o_i = \emptyset$. In particular, if $c$ is a terminal class, there is exactly one partition $\pi_{o,i}$ where $c \cap \pi_{o,i}$ is non-empty, so in this case, the tree functions as a classification tree (cf. Breiman et al. 1984).

In HPM estimation, we are given a data set $d$, a hierarchy $H$, and a prior distribution $F$, and must find a hierarchical partition model $M_d(H, o, F)$ that fits the data well, ie, we must determine an ordered cover $o$ whose resulting model $M_d(H, o, F)$ fits the data well. It is convenient to use the Bayesian estimation method described in section 5.1, ie, to assume that we are given a prior weight $W$ on the set of all HPM models, and must find an ordered cover $o$ that results in a model $M = M_d(H, o, F)$ that maximizes the *posterior weight* $W_{\text{post}}(M)$, defined as the prior weight $W(M)$ times the likelihood $L_{\hat{F}_M}(d)$, ie, a model satisfying:

$$\hat{M} = \arg\max_M W_{\text{post}}(M) = \arg\max_M W(M) \cdot L_{\hat{F}_M}(d).$$

The prior weight $W$ can be selected in many ways. Intuitively, the prior weight should be selected so that the estimated model $\hat{M}$ fits the data well while being as simple as possible, ie, we must strike a good balance between *underfitting* (failing to find patterns that are present in the data) and *overfitting* (finding spurious patterns that are not present in the true underlying distribution).

**Remark 5.35** One possible way of selecting the prior weight is to follow the *Minimum Description Length* (*MDL*) principle (cf. Rissanen 1978; Barron et al. 1998; Stolcke 1994; Li and Abe 1998). The philosophy behind MDL is that we should choose a model that is optimal for compressing the observed data, ie, a model in which the model, the model parameters, and the data can be encoded with as few bits as possible. Following Li and Abe (1998), we assume that the *parameter description length* $\ell(M)$ of a model $M = (H, o, F, w)$ estimated from the data set $d$ is defined by:

$$\ell(M) = \frac{|o| - 1}{2} \log_2(|d|).$$

The number $|o| - 1$ is the number of free parameters needed to encode the probabilities of the disjoint subsets in the partition corresponding to $o$, and

$\frac{1}{2}\log_2(|d|)$ is the number of bits that can reasonably be used to describe a parameter estimated from a data set with $|d|$ observations, from an information theoretic point of view. With these assumptions, the MDL principle gives prior weight:

$$W_{\text{MDL}}(M) = \left(\frac{1}{2}\right)^{\ell(M)} = \left(\frac{1}{2}\right)^{\frac{1}{2}(|o|-1)\log_2(|d|)} = \left(\sqrt{|d|}\right)^{1-|o|}$$

and hence posterior weight:

$$W_{\text{MDL-post}}(M) = \left(\sqrt{|d|}\right)^{1-|o|} \cdot L_{F_M}(d).$$

A model that maximizes $W_{\text{MDL-post}}$ is called an *MDL estimate* for the data set $d$. Intuitively, the parameter description length encodes how many bits are needed to encode the parameters, and the minus log likelihood value encodes how many bits are needed to encode the observed data within the given model. We will not go deeper into the MDL principle and its theoretical justifications here, but refer the interested reader to Li and Abe (1998), Quinlan and Rivest (1989), and Barron et al. (1998) for further details.

In statistical estimation, one often distinguishes between *bumps* (ie, areas where the density is high) and *dips* (ie, areas where the density is small) (cf. Basu et al. 2002). In sparse data, it is very difficult to identify dips reliably, whereas bumps are easier to identify because they contain many observations. The notions of bumps and dips within HPMs are defined formally below.

**Definition 5.36** Let $M = (H, o, F, w)$ be an HPM model. A partition $\pi_{o,i}$ in $M$ is called a *bump* if it has higher average density than its immediate super partition $\pi_{o,j}$ (ie, $w_i > w_j$), and a *dip* if it has lower average density than its immediate super partition (ie, $w_i < w_j$). $M$ is called a *bump model* if each of its partitions is either a root partition or a bump.

To ensure robustness in the statistical estimation, it is desirable to require HPM models to be bump models where each non-root partition contains at least 5 observations (or some other reasonable threshold). The prior MDL weight does not differentiate between bumps and dips, or place restrictions on the number of observations within non-root partitions, but it is straight-forward to modify the MDL weight so that it does.

**Definition 5.37** Let $M = (H, o, F, w)$ be a hierarchical partition model induced by the data set $d$. A partition $\pi_{o,i}$ in $o$ is called *degenerate* if $\pi_{o,i}$ fails to be a bump, or if $\pi_{o,i}$ is a non-root partition with fewer than *mincount* observations, where *mincount* is some globally defined threshold. $M$ is called *degenerate* if it contains a degenerate partition.

**Definition 5.38** The *bump-MDL prior weight* $W_{\mathrm{bMDL}}$ is a function which maps an arbitrary hierarchical partition model $M$ into the weight:

$$W_{\mathrm{bMDL}}(M) = \begin{cases} 0 & \text{if } M \text{ is degenerate} \\ W_{\mathrm{MDL}}(M) & \text{otherwise.} \end{cases}$$

The quantity $W_{\text{bMDL-post}}(M) = W_{\mathrm{bMDL}}(M) L_{\hat{F}_M}(d)$ is called the *bump-MDL posterior weight*. A hierarchical partition model $M = (H, o, F, w)$ that maximizes $W_{\text{bMDL-post}}(M)$ is called an *optimal bump-MDL estimate* for the data $d$ with hierarchy $H$ and prior distribution $F$.

Ordered covers can be constructed by means of the following two elementary cover operations.

**Definition 5.39** Let $H$ be a hierarchy, and let $O$ be the set of all possible ordered covers of $H$. If $o$ and $o'$ are ordered covers, we write $o.o'$ for the *concatenation* of $o$ and $o'$. A pair $o \mapsto o'$ with $o, o' \in O$ is called a *cover operation* mapping $o$ into $o'$. A cover operation $o.(c).o' \mapsto o.(s, c).o'$ where $s$ is an immediate subclass of $c$ is called a *partitioning* of $c$. A cover operation $o.(c).o' \mapsto o.o'$ is called a *merging* of $c$. Partitionings and mergings are collectively referred to as *elementary cover operations*.

**Example 5.40** An example of a sequence of partitionings and mergings within the unpartitioned square $\mathsf{T}$ is shown in Figure 5.8. The shade of gray indicates the density estimated for the class by the corresponding HPM estimator, where black corresponds to high densities, and white corresponds to low densities.

The proposition below shows that any ordered cover can be produced from the root class by means of a sequence of partitionings and mergings.

**Proposition 5.41** *Any ordered cover $c.(\mathsf{T})$ with $\mathsf{T} \notin c$ can be reached from $(\mathsf{T})$ by means of a series of partitionings and mergings, ie, there exists a sequence*

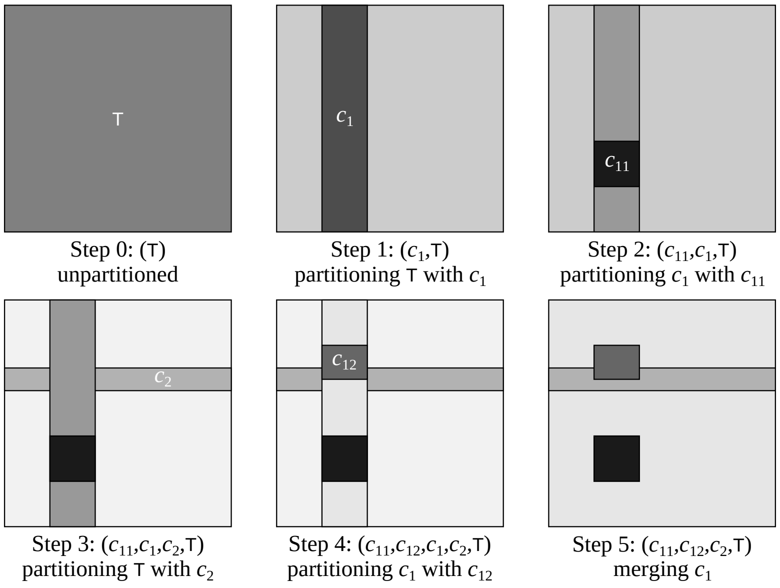

⊤
$c_1$
$c_{11}$
Step 0: (⊤) unpartitioned
Step 1: ($c_1$,⊤) partitioning ⊤ with $c_1$
Step 2: ($c_{11}$,$c_1$,⊤) partitioning $c_1$ with $c_{11}$
$c_2$
$c_{12}$
Step 3: ($c_{11}$,$c_1$,$c_2$,⊤) partitioning ⊤ with $c_2$
Step 4: ($c_{11}$,$c_{12}$,$c_1$,$c_2$,⊤) partitioning $c_1$ with $c_{12}$
Step 5: ($c_{11}$,$c_{12}$,$c_2$,⊤) merging $c_1$

Figure 5.8: A sequence of partitionings and mergings.

*$o_1, \ldots, o_n$ of ordered covers with $o_1 = (\top)$ and $o_n = c.(\top)$ such that $o_{i+1}.(\top)$ is a partitioning or merging of $o_i.(\top)$ for each $i = 1, \ldots, n-1$.*

*Proof.* Write $c = (c_1, \ldots, c_k)$. The proof proceeds by induction on $k$. If $k = 0$, then $c$ is empty, and the claim is trivial. So suppose the claim holds for all ordered covers whose length is less than $k$; we must prove the claim holds for an arbitrary ordered cover $c = (c_1, \ldots, c_k)$ with length $k$. By the induction, the ordered cover $c' = (c_1, \ldots, c_{k-1})$ can be produced by a sequence of local cover operations. Moreover, since $c_k$ is a subclass of $\top$, there obviously exists a sequence of partitionings that when applied to $c'.(\top)$ produces a cover $c'.(c''_1, \ldots, c''_m, \top)$ where $c''_1 = c_k$ and each $c''_i$ is an immediate subclass of $c''_{i+1}$. By merging the classes $c''_2, \ldots, c''_m$, we complete the construction of the ordered cover $c'.(c''_1, \top) = c$ by means of a sequence of local cover operations, thereby proving the induction. □

So far, we have described how HPM estimation can be used to model a single hierarchically structured random variable $X$. However, in practice, we often need to estimate probability distributions over conditional random variables of the form $X|Y$. We also need to deal with random vectors where only some of the random variables are hierarchically structured. For example, in a probabilistic language model, we might be interested in modelling dependencies as random vectors of the form $(D, L, N|H)$ where $H$ is the head word, $D$ is the dependent word, $L$ is the dependency label, and $N$ is the distance in time between head and dependent in the speech signal. To do this, we must be able to mix hierarchically organized categorial random variables with non-hierarchically organized random variables, such as continuous random variables.

**Definition 5.42** A conditional random vector of the form $(X, C|X', C', E')$ is said to follow a *conditional mixed HPM distribution* (or *XHPM distribution*) if $C, C'$ are hierarchically structured with hierarchies $H, H'$, respectively, and there exists an ordered cover $o = (o_1, \ldots, o_n)$ on $H \times H'$ and a hidden random variable $\Pi$ that maps an observation $(C, C')$ into the partition $\pi_{o,i}$ containing $(C, C')$, such that $E'$, $(C, C')$, $X$, and $X'$ are mutually independent given $\Pi$.

**Proposition 5.43** *Let $(X, C|X', C', E')$ be an XHPM distributed random variable. Then:*[2]

$$P(X, X', C, C', E') = \sum_{\Pi} P(E')P(\Pi|E')P(C, C'|\Pi)P(X|\Pi)P(X'|\Pi)$$

*and*

$$P(X, C|X', C', E') = \frac{\sum_{\Pi} P(\Pi|E')P(C, C'|\Pi)P(X|\Pi)P(X'|\Pi)}{\sum_{\Pi} P(\Pi|E')P(C'|\Pi)P(X'|\Pi)}.$$

*Proof.* From Bayes' law and our independence assumptions, we have:

$$\begin{aligned} &P(X, X', C, C', E', \Pi) \\ &\quad = P(E')P(\Pi|E')P(C, C'|E', \Pi)P(X, X'|E', \Pi, C, C') \\ &\quad = P(E')P(\Pi|E')P(C, C'|\Pi)P(X|\Pi)P(X'|\Pi). \end{aligned}$$

[2]To keep the formulas as readable as possible, we use the notation $\sum_{\Pi} P(\Pi|E') \cdots P(X'|\Pi)$ as a short-hand for the more accurate notation:

$$\sum_{\pi \in \Omega_{\Pi}} P(\Pi = \pi|E') \cdots P(X'|\Pi = \pi).$$

Since $P(X, X', C, C', E') = \sum_\Pi P(X, X', C, C', E', \Pi)$, we consequently have:

$$P(X, X', C, C', E') \\ = \sum_\Pi P(E')P(\Pi|E')P(C, C'|\Pi)P(X|\Pi)P(X'|\Pi).$$

By the definition of conditional probability, we have:

$$P(X, C|X', C', E') = \frac{P(X, X', C, C', E')}{P(X', C', E')} = \frac{P(X, X', C, C', E')}{\sum_{X,C} P(X, X', C, C', E')}.$$

We simplify the denominator by calculating:

$$\begin{aligned}&\sum_{X,C} P(X, X', C, C', E')\\ &= \sum_{X,C,\Pi} P(E')P(\Pi|E')P(C, C'|\Pi)P(X|\Pi)P(X'|\Pi)\\ &= P(E') \sum_\Pi P(\Pi|E') \sum_C P(C, C'|\Pi) \sum_X P(X|\Pi)P(X'|\Pi)\\ &= P(E') \sum_\Pi P(\Pi|E')P(C'|\Pi)P(X'|\Pi)\end{aligned}$$

since $\sum_C P(C, C'|\Pi) = P(C'|\Pi)$ and $\sum_X P(X|\Pi) = 1$. The second formula in the proposition arises by dividing $P(X, X', C, C', E')$ with $\sum_{X,C} P(X, X', C, C', E')$ and canceling out the common factor $P(E')$. □

**Remark 5.44** Suppose we are given a data set $d = \{(x_i, c_i, x'_i, c'_i, e'_i)\}_{i=1}^n$ for an XHPM distributed random variable $(X, C, X', C', E')$ with classification hierarchies $H, H'$ for $C, C'$, prior distributions $F_c, F_{c'}, F_x, F_{x'}$ for $C, C', X, X'$, and estimation algorithms $a, a'$ that given subsets $x_s, x'_s$ of $x = \{x_i\}_{i=1}^n$ and $x' = \{x'_i\}_{i=1}^n$ estimate distributions $a(x_s)$ and $a'(x'_s)$ for $X$ and $X'$. Given an ordered cover $o = (o_1, \dots, o_n)$ on $H \times H'$, we can use Proposition 5.43 to estimate distributions $\hat{F}_{m,o,d}(X, C, X', C', E')$ and $\hat{F}_{m,o,d}(X, C|X', C', E')$ for $(X, C, X', C', E')$ and $(X, C|X', C', E')$ by providing estimates for $P(\Pi|E')$, $P(C, C'|\Pi)$, $P(C'|\Pi)$, $P(X|\Pi)$, and $P(X'|\Pi)$; the predefined model parameters are encoded in $m = (H, H', F_c, F_{c'}, F_x, F_{x'}, a, a')$.

We will assume that the random variable $\Pi$ is defined in terms of the ordered cover $o$, ie, $\Pi$ maps an observation $(C, C')$ to the partition $\pi_{o,i}$ containing $(C, C')$. This allows us to estimate the probability $P(\Pi|E')$ empirically by means of:

$$P(\Pi|E') = \frac{\#(\Pi, E')}{\#E'}$$

where $\#X$ denotes the number of times the random variable $X$ has been observed with the given value.[3] We estimate $P(C, C'|\Pi)$ by means of:

$$P(C, C'|\Pi) = \frac{F^*(\Pi \cap (C, C'))}{F^*(\Pi)}$$

where $F^*$ is the distribution on the class algebra $(H \times H')^*$ induced by the distribution $F(c, c') = F_c(c)F_{c'}(c')$ on $H \times H'$ (cf. Remark 5.30). That is, the prior distributions $F_c$ and $F_{c'}$ on $H$ and $H'$ define a natural prior distribution $F$ on $H \times H'$, and $P(C, C'|\Pi)$ is the prior probability under $F$ of observing $(C, C')$ given that $(C, C') \in \Pi$. Using $F^*$, we can calculate:

$$P(C'|\Pi) = P(C \in \top, C'|\Pi) = \frac{F^*(\Pi \cap (\top, C'))}{F^*(\Pi)}.$$

The estimation algorithms $a$ and $a'$ can be used to construct estimates for $P(X|\Pi)$ and $P(X'|\Pi)$. The simplest method is to estimate $P(X|\Pi)$ and $P(X'|\Pi)$ by means of the estimators $a(x[\Pi])$ and $a'(x'[\Pi])$, where $x[\Pi]$ denotes all observations $x_i$ in $x$ where $(c_i, c_i') \in \Pi$, and similarly for $x'[\Pi]$. But as has been pointed out in credibility theory (cf. Norberg 2004), this method is unreliable if there are few observations in $\Pi$, and it is better to use a convex combination of the prior distribution and the estimates computed from the data in the partition $\Pi$ and the superclasses $\overline{\Pi}_1, \ldots, \overline{\Pi}_k$ of $\Pi$ (where $\overline{\Pi}_i$ denotes the union of partition $\Pi_i$ and all its transitive subpartitions, and we define $\Pi_{i+1}$ as the immediate super partition of $\Pi_i$, where $\Pi_1 = \Pi$ and $\Pi_k = \top$). This method results in estimators:

$$\begin{aligned} P(X|\Pi) &= \epsilon_0 a(x[\Pi]) + \epsilon_1 a(x[\overline{\Pi}_1]) + \cdots + \epsilon_k a(x[\overline{\Pi}_k]) + \epsilon_{k+1} f_x \\ P(X'|\Pi) &= \epsilon_0 a'(x'[\Pi]) + \epsilon_1 a'(x'[\overline{\Pi}_1]) + \cdots + \epsilon_k a'(x'[\overline{\Pi}_k]) + \epsilon_{k+1} f_{x'} \end{aligned}$$

where $\epsilon_i \in [0, 1]$ and $\epsilon_0 + \cdots + \epsilon_{k+1} = 1$. In Remark 5.45, we discuss how to select the weights $\epsilon_0, \ldots, \epsilon_{k+1}$. In this way, we have provided estimates for all the distributions that are needed to estimate $P(X, C, X', C', E')$ and $P(X, C|X', C', E')$.

[3]For notational convenience, the notation $P(\Pi|E') = \#(\Pi, E')/\#E'$ is used as a short-hand for the more accurate notation:

$$P(\Pi \in \pi | E' \in e') = \frac{\#(\Pi \in \pi, E' \in e')}{\#(E' \in e')}.$$

**Remark 5.45** Credibility theory (cf. Norberg 2004) states how to calculate the weights $\epsilon_0, \dots, \epsilon_{k+1}$ from Remark 5.44 (called *credibility factors* in credibility theory) so that we minimize the mean squared error (MSE) of the estimator. Unfortunately, this calculation requires that $X$ is a numerical random variable where the mean and variance of $X|\Pi$ are known to us in advance. In our general setting, these requirements are not necessarily fulfilled, because $X$ may be a non-numerical random variable where the mean and the variance are undefined. We will therefore choose the credibility factors on the basis of the formula below (where $n_0$ is the number of observations in $\Pi$, and $n_i$ is the number of observations in the superclass $\overline{\Pi}_i$), with the result that our estimator does not necessarily have minimal MSE when the MSE is well-defined:

$$\epsilon_i = \begin{cases} w(n_0) & \text{if } i = 0 \\ w(n_i) - w(n_{i-1}) & \text{if } i = 1, \dots, k \\ 1 - w(n_k) & \text{if } i = k+1 \end{cases}$$

The *credibility weight function* $w\colon [0,\infty] \to [0,1]$ must be a monotonically increasing function with $w(0) = 0$ and $w(\infty) = 1$. For example, $w$ could be chosen to be a Champernowne distribution $w(x) = x^\alpha/(x^\alpha + M^\alpha)$, whose parameters can be specified uniquely by means of the numbers $n_{50}$ and $n_{95}$ of observations that will result in the weights 0.5 and 0.95, respectively; we then have $M = n_{50}$ and $\alpha = \log 19/(\log n_{95} - \log n_{50})$. The numbers $n_{50}$ and $n_{95}$ must be chosen experimentally. As a reasonable starting point, we suggest $n_{50} = 10$ and $n_{95} = 100$.

**Definition 5.46** Let $d = \{(x_i, c_i, x'_i, c'_i, e'_i)\}_{i=1}^n$ be a dataset for an XHPM-distributed random variable $(X, C|X', C', E')$ with predefined model parameters $m = (H, H', F_c, F_{c'}, F_x, F_{x'}, a, a', w, w')$. Given an ordered cover $o$ on $H \times H'$, the estimator $\hat{F}_{m,o,d} = P(X, C|X', C', E')$ described in Remark 5.44 is called the *XHPM estimator* induced by the data $d$ with parameters $m$. The set $M$ of parameters needed to compute $\hat{F}_{m,o,d}$ is called the *XHPM model* associated with $\hat{F}_{m,o,d}$, and we often write $F_M$ instead of $\hat{F}_{m,o,d}$.

**Remark 5.47** In Remark 5.35, we described how to compute a parameter description length for an HPM model $M$, and how the parameter description length gave rise to a prior MDL weight $W_{\text{MDL}}$ on the set of HPM models. Similarly, we can define a prior MDL weight

$$W_{\text{MDL}}(M) = \left(\sqrt{|d|}\right)^{-n_M}$$

on the set of all XHPM models, where $n_M$ denotes the number of free parameters in the model $M$.[4] By excluding all degenerate models (like in Definition 5.38), we get a *bump-MDL prior weight* for XHPM models, which yields a *bump-MDL posterior weight* if we multiply the bump-MDL prior weight with the likelihood of the data for the conditional random variable $(X, C|X', C', E')$. An XHPM model with optimal bump-MDL posterior weight is called an *optimal bump-MDL estimate*. As with HPM models, Proposition 5.41 shows that any optimal bump-MDL estimate can be constructed by means of a sequence of partitionings and mergings.

**Remark 5.48** The HPM and XHPM models presented in this section can be seen as a generalization of Li and Abe's (1998; 1999) work on MDL-based case frame learning. HPM and XHPM models are a generalization in the following respects: (1) Li and Abe assume that all words within a disjoint subset in a partition (tree cut) are uniformly distributed, whereas we avoid this assumption by introducing a prior distribution which can be used to specify an arbitrary distribution of words within each disjoint subset. (2) Li and Abe assume that the disjoint subsets in a partition must correspond to word classes, whereas we have introduced ordered covers in order to allow the disjoint subsets to correspond to both word classes and set differences between word classes and finite unions of word classes. (3) Li and Abe assume that the hierarchy must be a tree, whereas we allow the hierarchy to be a general directed acyclic graph. (4) Li and Abe mostly deal with the 1-dimensional case, and their tree cut model does not scale well to higher-dimensional cases where hierarchies tend to be difficult to represent as trees, whereas our method works for arbitrary (finite) dimensions. (5) XHPM models are a further generalization that makes it possible

[4]There is a certain freedom with respect to how we define $n_M$. One extreme is to define $n_M = |o| - 1$, thereby ignoring anything but the parameters used to encode the number of data within each partition in $M$. Another extreme is to define $n_M$ as the total number of parameters stored in the model $M$, including the parameters used to store $\#(\Pi, E')$ and the parameters produced by the algorithms $a, a'$ for all partitions $\Pi, \bar{\Pi}$. It can be viewed as an experimental question to find the definition of $n_M$ that leads to the best experimental results.

to mix hierarchical and non-hierarchical data, and to compute conditional probability distributions.

In this section, we have proposed two families of distributions, HPM models which can be used to model random variables whose outcomes can be organized in a classification hierarchy, and XHPM models which can be used to model conditional random vectors with a mixture of hierarchically organized random variables and random variables with arbitrary distributions. We have shown the essential role played by ordered covers in both HPM and XHPM models, and how ordered covers can be constructed by means of partitioning and merging operations. Finally, on the basis of the Minimum Description Length principle, we have proposed a prior model probability on the set of all HPM and XHPM models that can be used to identify optimal HPM or XHPM models for a given data set, ie, HPM or XHPM models that optimize the posterior Bayesian probability of the model given the data set. In the following section, we will propose an algorithm that can be used to compute a near-optimal HPM or XHPM estimator for a given data set, using an algorithm based on local search.

## 5.4 Algorithm for HPM and XHPM estimation

***Summary**. We propose an algorithm based on first-improvement local search that often produces a near-optimal HPM or XHPM estimator for a given data set. The neighbourhoods in local search are defined by means of local merging operations and local partitioning operations with variable depth.*

In general, the problem of finding a globally optimal HPM or XHPM estimator for a given data set seems to be very difficult — we suspect that it is NP-hard in the general case. This means that the best algorithm we can hope for is a heuristic algorithm that works well in most cases, at the risk of returning a suboptimal solution in some cases. As our algorithm, we can in principle use any heuristic search algorithm, including beam search, $A^*$ search, or the many variants of local search. In this section, we will propose a heuristic algorithm based on first-improvement local search for finding a locally optimal HPM or XHPM model. In section 5.5, we will then present a simulation study that shows when our algorithm works well, and when it returns a suboptimal model. The algorithm is described below.

```
procedure find-bump-MDL-cover($m, d$)
begin
    $m$ := model parameters (including priors, etc.);
    $d$ := data;
    $newcover$ := $(\top)$;
    while $newcover \neq$ null do
        $cover$ := $newcover$;
        $newcover$ := partition($m$, $d$, $cover$, 1) || ... || partition($m$, $d$, $cover$, $|o|$)
            || merge($m$, $d$, $cover$, 1) || ... || merge($m$, $d$, $cover$, $|o| - 1$);
    return $cover$;
end
```

Figure 5.9: The find-bump-MDL-cover procedure for finding a locally optimal MDL estimate by means of first-improvement local search.

**Algorithm 5.49** The find-bump-MDL-cover algorithm takes as its input a data set $d$ and a collection $m$ of HPM or XHPM model parameters, and returns an ordered cover whose induced HPM or XHPM model is locally optimal with respect to its bump-MDL posterior weight. The model parameters in $m$ include hierarchies, prior densities, credibility weight functions and estimation algorithms for the different random variables in the random vector, as well as the parameter *mincount* which states the smallest number of observations that a partition is allowed to contain, and the parameter *maxdepth* which states the maximal search depth in the partition procedure.

The procedure find-bump-MDL-cover in Figure 5.9 uses first-improvement local search to find a locally optimal ordered cover. It starts with the ordered cover $(\top)$ — ie, its starting assumption is that the data follow the prior distribution — and then repeatedly improves its current cover by applying the first improvement it can find by either partitioning or merging a partition in the cover. When it fails to find a further improvement (ie, the current cover is locally optimal), it returns the current cover.

The procedure partition in Figure 5.10 searches for an improvement that can be obtained by partitioning a given partition in the ordered cover. The procedure divides the original cover into three parts: *pre* (which contains all partitions that precede the given partition), *mid* (which initially only consists of the given partition), and *post* (which contains all partitions that succeed the given partition). It then tries to find a subpartitioning of *mid* that improves the original cover. It does so by examining all immediate

```
procedure partition($m, d, cover, partition$)
begin
    $m$ := model parameters (including priors, etc.);
    $d$ := data;
    $cover$ := current cover;
    $partition$ := partition in $cover$ to be partitioned;
    $pre$ := all partitions in $cover$ before partition $partition$;
    $mid$ := $[cover[partition]]$;
    $post$ := all partitions in $cover$ after partition $partition$;
    $depth$ := $m$.maxdepth;
    do
        $nmax$ := 0;
        for $p$ := 1 to $|mid|$ do
            $s$ := subclass of $mid[p]$ with most observations;
            $n$ := number of observations in $s$;
            if $n > nmax$ and $n > m$.mincount then
                $nmax$ := $n$;
                $split$ := $p$;
                $subclass$ := $s$;
        if $nmax > 0$ then
            $mid$ := $mid$ with $subclass$ inserted before partition $split$;
            if sanitize($m, d, pre + mid + post$) improves $cover$ then
                $newcover$ := sanitize($m, d, pre + mid + post$);
    until $nmax = 0$ or $depth = 0$ or $newcover \neq$ null;
    return $newcover$;
end
```

Figure 5.10: The partition procedure for finding an improved partitioning of a given partition by means of first-improvement local search.

subdivisions of the partitions in *mid* produced by the given hierarchies, and selecting the subdivision with the largest possible data count. It then determines whether the sanitized (non-degenerate) ordered cover that results from using the new *mid* improves the original cover. The partitioning procedure continues until it is impossible to subdivide *mid* further, until the maximal search depth has been exceeded, or until the new cover given by *mid* is an improvement over the old cover. The procedure returns the new improved cover, if one is found, or null if no improved cover is found.

The procedure merge in Figure 5.11 searches for an improvement that

```
procedure merge($m, d, cover, partition$)
begin
    $m$ := model parameters (including priors, etc.);
    $d$ := data;
    $cover$ := current cover;
    $partition$ := partition in $cover$ to be merged;
    $newcover$ := $cover$ with partition $partition$ deleted;
    if sanitize($m, d, newcover$) improves $cover$ then
        return sanitize($m, d, newcover$);
    else
        return null;
end
```

Figure 5.11: The merge procedure for finding an improved cover by merging a given partition.

```
procedure sanitize($m, d, cover$)
begin
    $m$ := model parameters (including priors, etc.);
    $d$ := data;
    $cover$ := cover to be sanitized;
    while $cover$ is degenerate do
        remove the last degenerate partition in $cover$;
    return $cover$;
end
```

Figure 5.12: The sanitize procedure for modifying a cover to ensure that it is non-degenerate.

can be obtained by merging a given partition in the ordered cover. It returns the sanitized merged cover, if it improves the old cover, and null if it fails to be an improvement.

The procedure sanitize in Figure 5.12 modifies a given cover in order to ensure that it is non-degenerate. It does so by repeatedly merging the last degenerate partition in the cover, until the entire cover is non-degenerate.

**Remark 5.50** In the partition procedure, we select subdivisions on the basis of their data count — an approach which is partly inspired by the patient peeling strategy described by Friedman and Fisher (1999). This approach forces the search to focus on regions with many observations, which are more likely to contain bumps than regions with few observations. In this process, the partition procedure may create some partitions that actually decrease the bump-MDL posterior weight; however, these suboptimal subdivisions can often be repaired by the merge operation at a later stage.

An alternative criterion is to use variable-depth local search, ie, to select a subdivision that maximizes the bump-MDL posterior weight rather than the data count. We have tried this approach, but it works poorly. The reason is that in order to arrive at an improvement that requires more than one subdivision, the algorithm often has to apply subdivisions that temporarily result in a large degradation of the bump-MDL posterior weight. Unfortunately, instead of using the correct, but costly subdivision, the algorithm is usually able to find a spurious subdivision that involves very few observations and therefore only leads to a very small degradation, even though it is a wasted step in terms of getting closer to a real improvement.

A second alternative criterion is to select the subdivision that maximizes the *moved mass*, defined as the extra probability mass that a subdivision assigns to the new subpartition. We have tried this approach as well, and it suffers from the same problems as variable-depth local search.

**Remark 5.51** The find-bump-MDL-cover procedure assumes that the parameter $m$ includes all information needed to estimate an HPM or XHPM model from the data set and the ordered cover, and to compute the bump-MDL posterior weight for the data set in the resulting model. For HPM models, the procedure for computing an induced model from a data set, a cover, and the given HPM model parameters is stated in Remark 5.33. For XHPM models, the corresponding procedure is stated in Remarks 5.44 and 5.45.

**Remark 5.52** In the case where the hierarchy is a tree, Li and Abe (1998) have used dynamic programming to find a globally optimal MDL estimate. However, their solution does not address the general case where the hierarchy is a directed acyclic graph with a very large number of terminals, as in product and intersection hierarchies. As far as we can see, their dynamic programming algorithm does not work in this general setting.

**Remark 5.53** The idea of using merging and partitioning operations to find a near-optimal solution is not new. For example, Belz (2002) has proposed an algorithm for learning PCFGs that uses merging and partitioning operations in order to improve an existing PCFG. However, her approach is different from ours in most other respects. For example, she uses a variant of beam search to find a PCFG with an optimal F-score, and as partitions, she uses classes rather than set differences between classes and finite unions of classes.

In this section, we have described our heuristic algorithm for finding a near-optimal HPM or XHPM model. In the following section, we evaluate how well our algorithm works by means of a simulation study.

## 5.5 Empirical evaluation of HPM estimation

***Summary.*** *We evaluate how well our heuristic HPM estimation algorithm works by means of a simulation study. From a wide range of different two-dimensional distributions on the unit square* $[0,1] \times [0,1]$*, we generate data sets of different size, and measure the distance between the true model and the HPM estimate produced from the data. We also show how dependency relations can be interpreted as distributions on the unit square.*

In order to assess the quality of our statistical estimation method, we need to test it on data where we know the true underlying distribution so that we can compare the estimated distribution with the true distribution and get a precise measure of the error and the rate of convergence. Since we do not know the true distribution of treebank data, it is convenient to test the method on random data that have been generated from a known distribution, such as a distribution on the unit square $[0,1] \times [0,1]$.

**Remark 5.54** The different steps of the simulation are shown in Figure 5.13. From a predefined true distribution (shown left), we generate a random data set with $n$ observations (middle left). From this data set, the heuristic HPM estimation algorithm computes an HPM distribution (middle right) and an associated ordered cover (right) with near-optimal posterior bump-MDL weight. The random variation in the data sets leads to slight variations in the HPM estimates produced by the HPM estimation algorithm, so in order to measure the estimation error reliably, the simulation procedure is repeated 10 times. We can create a visualization of the "typical" HPM estimate by ranking the 10 estimates with respect to $L_2$ error (cf. Definition 5.56), and drawing the estimate with median $L_2$ error (ie, the 5th best estimate).

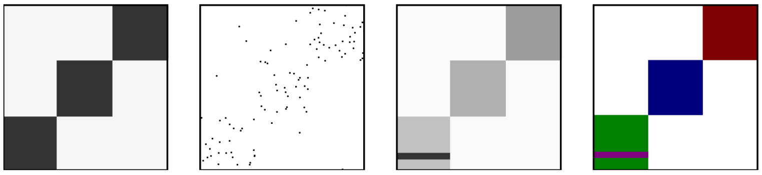


Figure 5.13: The simulation procedure. *Left*: The true distribution. *Middle left*: A data set with 100 observations generated randomly from the true distribution. *Middle right*: The HPM distribution induced from the data set. *Right*: The underlying ordered cover.

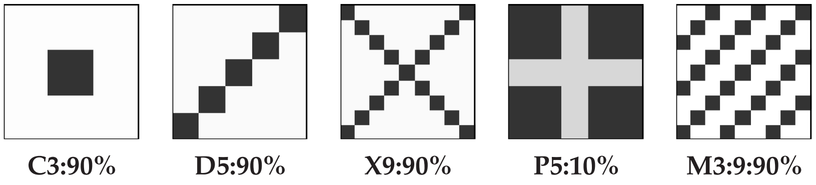


Figure 5.14: The different families of distributions used in our simulation.

To make it easier to visualize the distributions, we have tested the algorithm on a wide range of distributions on the unit square $[0,1] \times [0,1]$. The distributions are chosen to be easily recognizable geometric patterns, so that estimation errors are spotted easily. The distributions are defined formally below.

**Definition 5.55** We define the following families of distributions on the unit square $[0,1] \times [0,1]$. Examples of the five kinds of distributions are shown in Figure 5.14.

- **C*n*:*p***: the unit box is divided into two uniformly distributed areas: a centered square with side length $1/n$ containing probability mass $p$, and the remaining part of the unit box containing probability mass $1 - p$.
- **D*n*:*p***: the unit box is divided into two uniformly distributed areas: a single diagonal with $n$ squares with side length $1/n$ containing probability mass $p$, and the remaining part of the unit box containing probability mass $1 - p$.
- **X*n*:*p***: the unit box is divided into two uniformly distributed areas: two diagonals consisting of $2n - 1$ squares with side length $1/n$ con-

taining probability mass $p$, and the remaining part of the unit box containing probability mass $1 - p$.

- **P*n*:*p***: the unit box is divided into two uniformly distributed areas: two centered horizontal and vertical bars with widths $1/n$ containing probability mass $p$, and the remaining part of the unit box containing probability mass $1 - p$.
- **M*m*:*n*:*p***: the unit box is divided into two uniformly distributed areas: an area containing probability mass $p$ and consisting of all boxes $[i/n, (i+1)/n] \times [j/n, (j+1)/n]$ with $i, j = 0, \ldots, n-1$ where $i + j$ is a multiple of $m$, and the remaining part of the unit box containing probability mass $1 - p$.

When comparing the true distribution with the HPM estimate, it is convenient to have a measure of the estimation error. The $L_1$ and $L_2$ norms are often used as error measures for this purpose.

**Definition 5.56** The $L_n$ distance between two distributions $f$ and $f'$ on a space $S$ is defined as

$$L_n(f, f') = \sqrt[n]{\int_S |f(s) - f'(s)|^n \, ds}.$$

In the simulations, we must define some kind of hierarchy on the unit square. We have used an infinite 3-based hierarchy where every class can be subdivided into three equally large immediate subclasses either vertically or horizontally, ie, every class has six partly overlapping immediate subclasses. In this setting, it is interesting to test how well the HPM estimation algorithm performs when the hierarchy is compatible with the true distribution, and how well it performs when the hierarchy is incompatible — ie, to test how a bad choice of hierarchy affects the estimation. As our prior distribution, we have used the uniform distribution. We have tested the algorithm on a wide range of distributions, using random samples with $10, 30, 100, 300, 1{,}000, 3{,}000, 10{,}000$, and $30{,}000$ observations.

**Remark 5.57** Figure 5.15 is a visualization of the typical HPM estimates (ie, HPM estimates with median $L_2$ error) produced by the HPM estimation algorithm for data sets with different sample sizes, generated from distributions that are compatible with the 3-based hierarchy used in the HPM estimation algorithm (with maximal search depth 20, and minimal

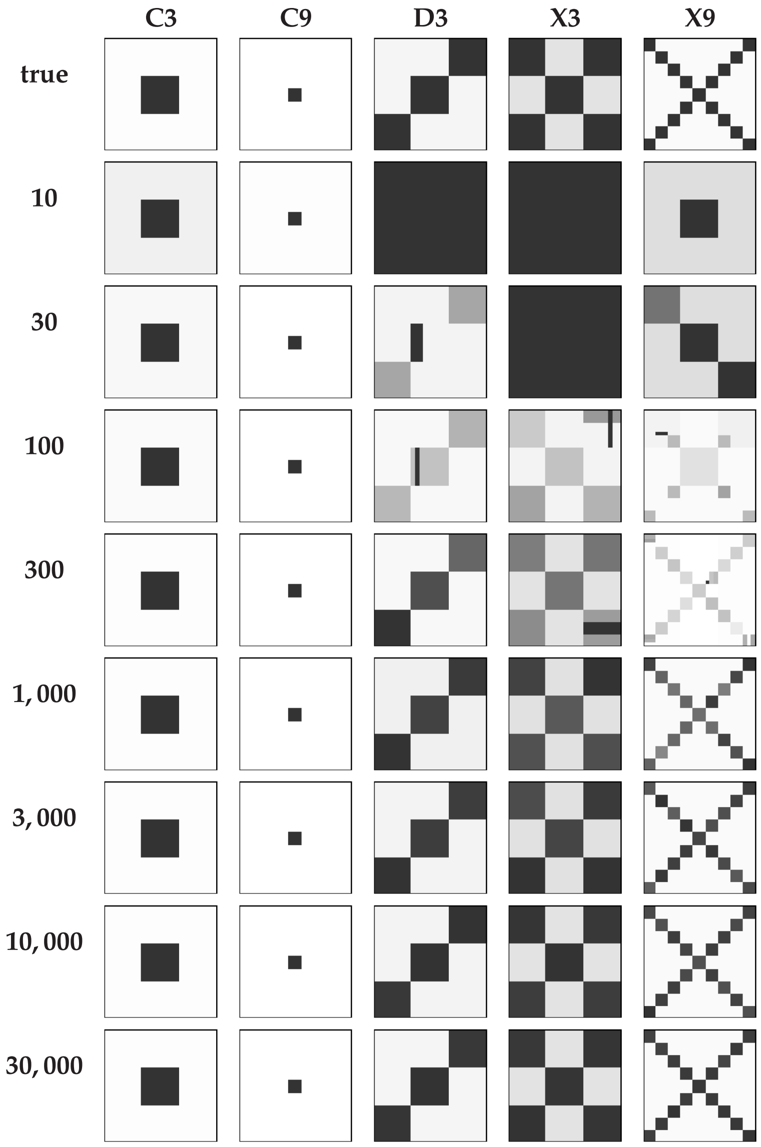


Figure 5.15: HPM estimates with median $L_2$ error of distributions that are compatible with the hierarchy, with different sample sizes.

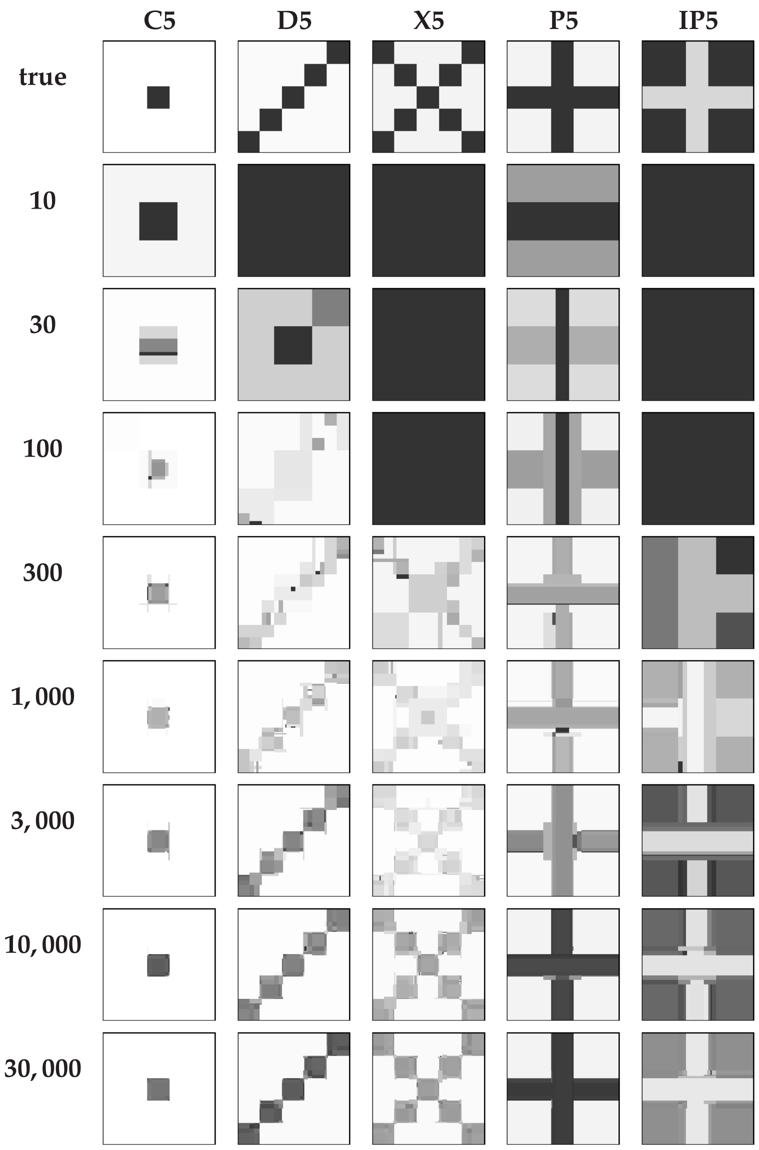


Figure 5.16: HPM estimates with median $L_2$ error of distributions that are incompatible with the hierarchy, with different sample sizes.

| $n$ | C3:90% | C9:90% | D3:90% | X3:90% | X9:90% |
|---|---|---|---|---|---|
| **10** | 0.522 (*0.128*) | 0.698 (*0.132*) | 1.115 (*0.054*) | 0.675 (*0.019*) | 1.348 (*0.073*) |
| **30** | 0.310 (*0.188*) | 0.347 (*0.109*) | 0.426 (*0.214*) | 0.605 (*0.164*) | 1.180 (*0.070*) |
| **100** | 0.109 (*0.072*) | 0.115 (*0.098*) | 0.206 (*0.096*) | 0.318 (*0.083*) | 0.759 (*0.072*) |
| **300** | 0.038 (*0.021*) | 0.059 (*0.029*) | 0.090 (*0.041*) | 0.121 (*0.042*) | 0.272 (*0.055*) |
| **1,000** | 0.018 (*0.024*) | 0.020 (*0.013*) | 0.054 (*0.029*) | 0.066 (*0.021*) | 0.109 (*0.020*) |
| **3,000** | 0.009 (*0.007*) | 0.005 (*0.005*) | 0.028 (*0.015*) | 0.045 (*0.011*) | 0.058 (*0.011*) |
| **10,000** | 0.004 (*0.004*) | 0.005 (*0.003*) | 0.011 (*0.007*) | 0.018 (*0.006*) | 0.038 (*0.004*) |
| **30,000** | 0.003 (*0.002*) | 0.004 (*0.002*) | 0.007 (*0.002*) | 0.010 (*0.003*) | 0.018 (*0.002*) |

| $n$ | C5:90% | D5:90% | X5:90% | P5:90% | P5:10% |
|---|---|---|---|---|---|
| **10** | 1.253 (*0.112*) | 1.400 (*0.002*) | 1.076 (*0.009*) | 0.878 (*0.097*) | 0.544 (*0.090*) |
| **30** | 1.008 (*0.115*) | 1.155 (*0.093*) | 1.060 (*0.037*) | 0.704 (*0.065*) | 0.554 (*0.086*) |
| **100** | 0.606 (*0.096*) | 1.008 (*0.065*) | 1.039 (*0.053*) | 0.527 (*0.060*) | 0.479 (*0.062*) |
| **300** | 0.327 (*0.060*) | 0.707 (*0.070*) | 0.731 (*0.049*) | 0.281 (*0.084*) | 0.362 (*0.048*) |
| **1000** | 0.184 (*0.027*) | 0.376 (*0.039*) | 0.527 (*0.034*) | 0.138 (*0.034*) | 0.184 (*0.024*) |
| **3000** | 0.099 (*0.020*) | 0.236 (*0.013*) | 0.315 (*0.023*) | 0.100 (*0.027*) | 0.123 (*0.012*) |
| **10000** | 0.049 (*0.015*) | 0.109 (*0.008*) | 0.171 (*0.005*) | 0.040 (*0.009*) | 0.074 (*0.007*) |
| **30000** | 0.022 (*0.003*) | 0.061 (*0.003*) | 0.084 (*0.003*) | 0.017 (*0.004*) | 0.029 (*0.004*) |

Figure 5.17: $L_1$ errors: mean error and standard deviation (in parentheses)

partition count 5). Likewise, Figure 5.16 is a visualization of the typical HPM estimates for the distributions that are incompatible with the 3-based hierarchy. Figure 5.17 and 5.18 show the mean $L_1$ and $L_2$ errors, and the associated standard deviations.

From the $L_1$ and $L_2$ errors, we see that the estimates seem to eventually converge towards the true distribution, both for distributions that are compatible with the hierarchy and for distributions with an incompatible hierarchy — but the convergence is significantly slower for distributions with an incompatible hierarchy (as with the 5-based distributions).

The distributions C3:90% and C9:90% show that the algorithm can produce reliable HPM estimates with as little as 10 observations, even in the presence of 10% background noise, if the shape of the distribution is sufficiently simple — ie, if the distribution contains a single high-density class, regardless of the size of the class (this result also holds for C27:90%). If most of the observations are concentrated in a single class (as in C5:90%), 10 observations are enough for the HPM estimation algorithm to zoom in on that class.

More generally, if the hierarchy is compatible with the true ordered cover, the estimation algorithm seems to require approximately 10–20 ob-

| *n* | **C3:90%** | **C9:90%** | **D3:90%** | **X3:90%** | **X9:90%** |
|---|---|---|---|---|---|
| **10** | 0.830 (*0.204*) | 3.152 (*0.597*) | 1.220 (*0.033*) | 0.730 (*0.047*) | 1.704 (*0.016*) |
| **30** | 0.622 (*0.512*) | 1.757 (*1.025*) | 0.646 (*0.350*) | 0.787 (*0.308*) | 1.611 (*0.043*) |
| **100** | 0.174 (*0.113*) | 0.582 (*0.628*) | 0.457 (*0.234*) | 0.559 (*0.113*) | 1.374 (*0.165*) |
| **300** | 0.065 (*0.034*) | 0.251 (*0.147*) | 0.173 (*0.085*) | 0.241 (*0.201*) | 0.816 (*0.133*) |
| **1,000** | 0.042 (*0.075*) | 0.084 (*0.047*) | 0.083 (*0.053*) | 0.090 (*0.022*) | 0.288 (*0.053*) |
| **3,000** | 0.014 (*0.011*) | 0.022 (*0.020*) | 0.051 (*0.031*) | 0.065 (*0.017*) | 0.147 (*0.033*) |
| **10,000** | 0.007 (*0.007*) | 0.022 (*0.015*) | 0.019 (*0.014*) | 0.029 (*0.013*) | 0.093 (*0.013*) |
| **30,000** | 0.005 (*0.003*) | 0.016 (*0.011*) | 0.012 (*0.004*) | 0.015 (*0.006*) | 0.048 (*0.006*) |

| *n* | **C5:90%** | **D5:90%** | **X5:90%** | **P5:90%** | **P5:10%** |
|---|---|---|---|---|---|
| **10** | 3.665 (*0.053*) | 1.766 (*0.020*) | 1.133 (*0.016*) | 1.041 (*0.072*) | 0.603 (*0.143*) |
| **30** | 3.348 (*0.265*) | 1.673 (*0.123*) | 1.157 (*0.052*) | 0.961 (*0.193*) | 0.616 (*0.149*) |
| **100** | 2.674 (*0.306*) | 1.669 (*0.175*) | 1.160 (*0.052*) | 0.817 (*0.118*) | 0.565 (*0.099*) |
| **300** | 1.911 (*0.278*) | 1.374 (*0.124*) | 1.059 (*0.057*) | 0.594 (*0.099*) | 0.501 (*0.079*) |
| **1000** | 1.422 (*0.192*) | 1.005 (*0.083*) | 0.844 (*0.051*) | 0.364 (*0.080*) | 0.353 (*0.036*) |
| **3000** | 0.880 (*0.135*) | 0.698 (*0.028*) | 0.624 (*0.025*) | 0.283 (*0.080*) | 0.256 (*0.008*) |
| **10000** | 0.486 (*0.145*) | 0.417 (*0.016*) | 0.418 (*0.011*) | 0.158 (*0.045*) | 0.187 (*0.011*) |
| **30000** | 0.246 (*0.046*) | 0.259 (*0.010*) | 0.250 (*0.024*) | 0.053 (*0.025*) | 0.093 (*0.009*) |

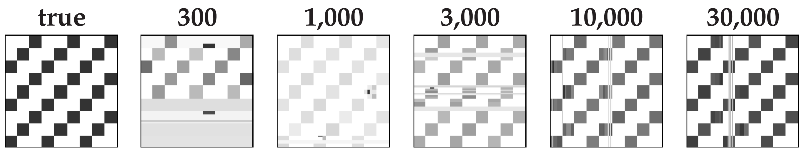
Figure 5.18: $L_2$ errors: mean error and standard deviation (in parentheses)

servations within each true partition in order to guess most of the partitions in the true ordered cover. Thus, the estimation becomes reasonably reliable at approximately 10 observations for C3:90% and C9:90%, at 30 observations for D3:90%, at 100 observations for X3:90%, and at 300 observations for X9:90%. If the hierarchy is incompatible with the true ordered cover, the estimation algorithm will produce increasingly more accurate approximations to the true ordered cover.

While our algorithm seems to converge towards the true distribution for all distributions we have tested, it is possible to find distributions where the algorithm is being tricked into returning a suboptimal solution.

**Remark 5.58** The property that seems to cause most problems in our estimation algorithm, apart from incompatible hierarchies, is *apparent uniformity* — ie, distributions that appear to be uniform down to a certain depth. For example, Figure 5.19 shows the estimates for a true distribution that is uniform down to depth 4, ie, where all classes in the hierarchy that can be obtained from the unit square by at most three subdivisions appear to have the same density.

The estimates seem to converge slowly towards the true distribution,



| | 300 | 1,000 | 3,000 | 10,000 | 30,000 |
|---|---|---|---|---|---|
| **$L_1$ error:** | 0.702 (*0.267*) | 0.318 (*0.164*) | 0.272 (*0.124*) | 0.110 (*0.058*) | 0.087 (*0.039*) |
| **$L_2$ error:** | 1.065 (*0.223*) | 0.625 (*0.225*) | 0.568 (*0.203*) | 0.296 (*0.137*) | 0.280 (*0.104*) |

Figure 5.19: The HPM estimates for M3:9:90%, a pathological distribution that is uniform down to depth 4, and the associated $L_1$ and $L_2$ mean error and standard deviation.

but the apparent uniformity seems to make the convergence much slower than usually seen because of two kinds of inadequate partitioning. First of all, the uniformity means that the partition procedure tries to partition the space in a breadth-first manner, ie, it starts by partitioning the unit square into 3 depth 1 subspaces, then 9 depth 2 subspaces, then 27 depth 3 subspaces etc. As a consequence, it may reach the maximal search depth before it reaches a depth 4 subspace where it can detect the non-uniformity. Secondly, because of the uniformity, the partition procedure may inadvertently choose the wrong subpartitioning — eg, instead of creating a small square, it creates a beam with the same area that is too long vertically, but too short horizontally. These beams are clearly visible in all of the HPM estimates produced by the algorithm for M3:9:90%.

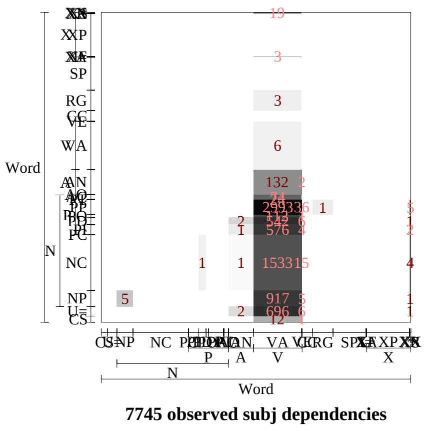
The second problem can probably be fixed by introducing an abstraction operation that tries to successively replace a class with each of its immediate superclasses, replacing the class with the best superclass if an improvement is possible. This will probably not lead to significantly improved estimates for the other, non-pathological families of distributions that we have tested, but it is conceivable that it may lead to improved estimates for complex treebank distributions, with a large enough improvement to justify the increase in computing time.

On the distributions where we have tested our algorithm, it seems to perform well. However, it is difficult to determine how well it performs for distributions with very different distributions, priors, hierarchies, etc.

**Remark 5.59** So far, we have only tested the estimation algorithm on distributions on the two-dimensional unit square, using a simple hierarchy.

These tests can be extended in many directions. First of all, instead of using the same probabilities in all the boxes in the C, D, X, P, and M families of distributions, we could use different probabilities in each box — we have tried this, and it only seems to make the estimation converge faster, which is not so surprising, given that apparent uniformity is the most difficult case to handle for the algorithm. We could also try to increase or decrease the amount of background noise in the distributions — we have tried this, and unsurprisingly, the amount of data needed to keep the error below a certain level increases with the signal-noise ratio. We could also try to increase the dimensionality — we have not tried this, but we suspect that dimensionality is not an overly important factor with this algorithm: what matters is the search depth, and the number of partitions needed to encode the true distribution. Finally, we could try using different hierarchies and prior distributions (eg, hierarchies where the immediate subclasses of a class have very different prior mass) — we have not tried this, but these choices are likely to influence the quality of the estimation, although it is difficult to predict how. We have not had the time to look into these questions, but we plan to address them when we have reimplemented the learning algorithm.

**Remark 5.60** Two-dimensional distributions on the unit square may seem far removed from treebank distributions. However, the difference is not as big as it may seem at first sight. Figure 5.20 shows the observed distribution of head-dependent pairs in subj and vobj dependencies in the Danish Dependency Treebank. The horizontal axis represents the word class of the head in the dependency relation, and the vertical axis represents the word class of the dependent in the dependency relation. The word classes are grouped together by means of the word class hierarchy in Figure 5.21, which is based on the Danish PAROLE word classes (cf. Keson and Norling-Christensen 1998). This word class hierarchy is shown in compact form along the axes in Figure 5.20, and the width of the interval representing each word class is chosen so that it is proportional to the frequency of the word class in the treebank (eg, the length of the line representing the class "N" is approximately 35% of the length of the line representing the class "Word" because 35% of all words in the treebank are nouns). The frequency of each word-class pair is indicated with a coloured box and a frequency count, where dark colours represent high frequencies relative to the area of the box, and light colours represent low frequencies relative to



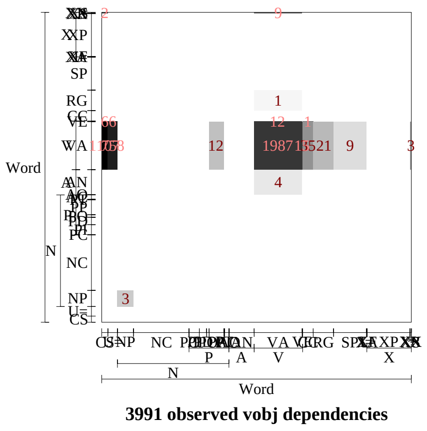


Figure 5.20: The observed distribution of subj and vobj dependencies in the Danish Dependency Treebank.

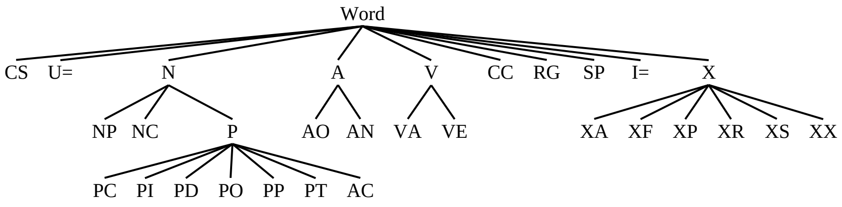


Figure 5.21: The hierarchy of PAROLE word classes used in Figure 5.20.

the area of the box.[5] For example, the darkest box in the “vobj” graph is (VA,VA) (a normal verb taking a normal verb as its verbal object), a combination that has been observed 1,987 times in the treebank. In this way, the distribution for a dependency relation in the treebank can be viewed as a distribution on the unit square. In general, if we have a mapping of the random variables in a random vector to the interval $[0, 1]$, a random vector with $n$ random variables can be viewed as a distribution on the $n$-dimensional unit box $[0, 1]^n$.

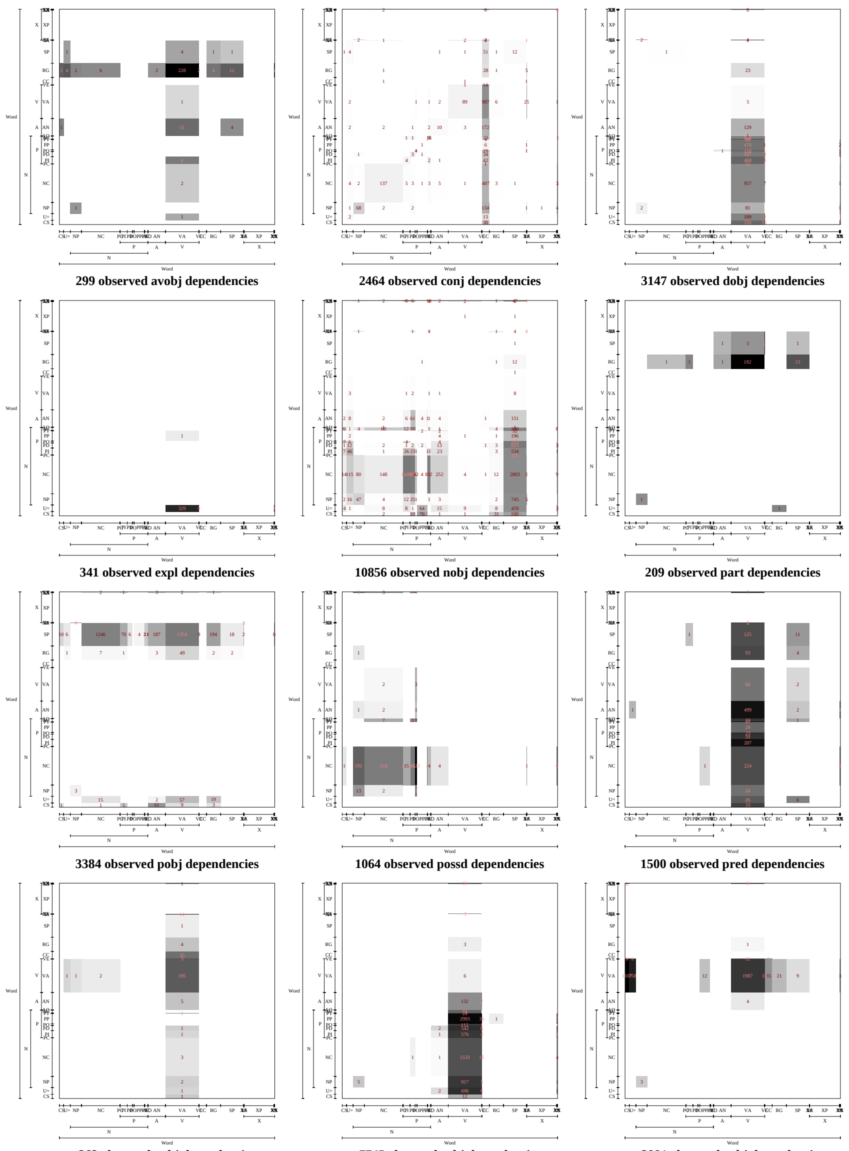
Figure 5.22 shows the observed distribution of the 12 most frequent

[5]The number of observations divided by the area of the box is a measure of how many times more probable the word-class pair is, compared to the probability one gets by assuming that the head and dependent are chosen independently of each other, using the observed probability distribution of single word classes.

dependencies in the Danish Dependency Treebank. When we look at the treebank data, we see that most observations are clustered within a relatively small number of word-class combinations, which provide a highly characteristic "finger print" for each dependency relation. For example, in the verbal object relation, there are 1,987 observations in (VA,VA) (a verb subcategorizing for a verb), 1,107 observations in (CS,VA) (a subordinating conjunction subcategorizing for a verb), 758 observations in (U=,VA) (infinitival marker "at" ("to") subcategorizing for a verb), with the remaining 139 observations distributed among a small number of word-class combinations.[6] Because of the small frequency counts, the learning algorithm will ignore the least frequent class combinations and analyze them as uniformly distributed noise. The nominal object and conjunct relations are more complicated than the verbal object relation, but again, the observations are clustered within relatively few word-class combinations, in a highly non-symmetric and non-uniform way that should be easy for our HPM estimation algorithm to learn.

In this section, we have tested our algorithm on a wide range of distributions on the unit square. The simulations have indicated that the algorithm usually converges towards the true distribution, but that the rate of convergence depends crucially on the quality of the hierarchy provided to the learning algorithm: a good hierarchy allows the algorithm to zoom in on the true distribution quickly, whereas a bad hierarchy significantly increases the number of observations required for reliable estimation. We have seen that, as a rule of thumb, the algorithm requires approximately 10 observations within a partition before the partition appears in the ordered cover produced by our estimation algorithm. We have also tested the algorithm on distributions that seem to be uniform down to partition depth 4, and seen that the algorithm can cope with them, but that they do present a challenge to the algorithm. Finally, we have seen that we can often define a mapping from the space of outcomes of a random variable to the interval $[0,1]$, so that most $n$-dimensional random vectors (including random vectors for treebank data) can be viewed as distributions on the $n$-dimensional

[6]Some of the less frequent word-class combinations may represent "design errors" in the treebank. For example, the three occurrences of (NP,NP) (a proper noun subcategorizing for a proper noun) stem from English song titles like "You've Got to Hide Your Love Away," which have received a normal dependency analysis in the treebank, although all the words have been tagged as "NP" in the original Danish PAROLE corpus.



299 observed avobj dependencies
2464 observed conj dependencies
3147 observed dobj dependencies
341 observed expl dependencies
10856 observed nobj dependencies
209 observed part dependencies
3384 observed pobj dependencies
1064 observed possd dependencies
1500 observed pred dependencies
262 observed qobj dependencies
7745 observed subj dependencies
3991 observed vobj dependencies

Figure 5.22: Observed distribution of the 12 most frequent dependencies in the Danish Dependency Treebank.

unit box $[0,1]^n$.

## 5.6 Summary

In this chapter, we have presented an algorithm for estimating probability distributions from both categorical and continuous data. In section 5.1, we outlined the main concepts in statistical estimation. In section 5.2, we defined our notion of hierarchies and how several hierarchies for the same variable can be combined into a single intersection hierarchy, and how hierarchies for the variables in a random vector can be combined into a single product hierarchy for the entire random vector. In section 5.3, we proposed HPM and XHPM models with ordered covers as models for hierarchically organized random variables with known hierarchies and prior distributions, including conditional random vectors containing mixtures of hierarchically organized and continuous random variables, and we introduced the Minimal Description Length principle for selecting an optimal model. In section 5.4, we proposed a heuristic algorithm based on local search that can compute an HPM or XHPM estimate for a given data set, given a set of model parameters containing hierarchies, prior distributions, etc. Finally, in section 5.5, we demonstrated that our algorithm performs well on a wide range of distributions on the unit square, and that distributions for dependency relations in treebanks can be transformed into distributions on the unit square.

In the following chapter, we will introduce a formal language that can be used to define probabilistic language models by specifying how data are extracted from treebanks, and how they should be modelled statistically by means of HPM and XHPM distributions, and other statistical distributions.

# Chapter 6

# Probabilistic language models

In this chapter, we describe how DG can be combined with probabilistic language models. In section 6.1, we define a formal language that can be used to specify probabilistic language models in DG. In section 6.2, we propose a probabilistic language model for DG that models most of the linguistic aspects described in chapters 2–4.

## 6.1 Specifying probabilistic language models

***Summary.*** *We define a modelling language that can be used to specify probabilistic language models in DG. We describe the primitives in the modelling language for specifying submodels, extracting submodel data from the DG graph, computing probabilities, and specifying the underlying model parameters such as the choice of probability distribution, hierarchies, prior distribution, credibility weights, and smoothing.*

Instead of hard-coding a particular probabilistic language model into the DG system, we will provide a more general solution by defining a formal modelling language that can be used to specify arbitrary probabilistic language models within DG. This approach makes it easier to specify complex language models, experiment with different language models, and extend language models to new domains. A language model must perform three tasks: it must decompose a given graph into a set of observations of different conditional random vectors, it must estimate parameters for the conditional random vectors on the basis of the observations extracted from the graphs in a treebank, and it must compute the probability of a given graph on the basis of the parameters estimated from the treebank.

We start by modelling the extracted data by means of *augmentations*, defined formally below, which consist of an ordinary graph where some of the nodes have been augmented with a unique identifier and a set of attribute-value pairs. These are used to translate the graph into an observation of a conditional random vector whose random variables correspond to the node attributes in the augmentation — ie, the random variables are identified uniquely by a node identifier and an attribute name. The augmentations also specify an optional set of attribute types for each node attribute which allows model parameters to be specified for classes of node attributes rather than individually for each node attribute.

**Definition 6.1** An *augmentation* in a graph $G$ is a tuple $(G, N, I, A)$ such that $N$ is a subset of the nodes in $G$, $I$ is a function that maps a node $n \in N$ into a typed value of the form $I(n) = (i, t)$ where $i$ is a unique identifier and $t$ is a list of types associated with $n$, and $A$ is a partial function that maps a node $n \in N$ and an attribute $a$ into a typed value of the form $A(n, a) = (v, t)$ where $v$ is a value and $t$ is a list of attribute types associated with the attribute $a$ in $n$.

Two augmentations of a graph $G$ can be concatenated, provided they do not contain incompatible node identifiers or node attributes.

**Definition 6.2** Let $\alpha_1 = (G, N_1, I_1, A_1)$ and $\alpha_2 = (G, N_2, I_2, A_2)$ be two augmentations of a graph $G$. We will say that $\alpha_1$ and $\alpha_2$ are *incompatible* if there exists a node $n \in N_1 \cap N_2$ such that $I_1(n) \neq I_2(n)$, or if there exists a node $n \in N_1 \cap N_2$ and an attribute $a$ such that $A_1(n, a)$ and $A_2(n, a)$ are both defined, but unequal; $\alpha_1$ and $\alpha_2$ are called *compatible* otherwise. If $\alpha_1$ and $\alpha_2$ are compatible, we define their *concatenation* $\alpha = (G, N, I, A)$ by $N = N_1 \cup N_2$, by letting $I(n)$ equal $I_1(n)$ or $I_2(n)$ (whichever is defined), and by letting $A(n, a)$ equal $A_1(n, a)$ or $A_2(n, a)$ (whichever is defined).

We can now define a submodel of a probabilistic language model as a tuple consisting of an extraction procedure, an estimation procedure, and an evaluation procedure.

**Definition 6.3** A *submodel* is a tuple $s = (\mathrm{Ext}_s, \mathrm{Est}_s, \mathrm{Prob}_s)$ where (a) $\mathrm{Ext}_s$ is an *extraction procedure* that given a graph $G$ and a node $n$ computes a set $\mathrm{Ext}_s(G, n)$ of augmentations extracted from $G$ at $n$; (b) $\mathrm{Est}_s$ is an *estimation* *procedure* that given a set $\alpha$ of augmentations computes a set of *model parameters* $\pi_s(\alpha) = \mathrm{Est}_s(\alpha)$; and (c) $\mathrm{Prob}_s$ is an *evaluation procedure* that given a set of parameters $\pi$ and an augmentation $a$ computes a probability $\mathrm{Prob}_{s,\pi}(a)$. Given a treebank $T$, we define $\mathrm{Prob}_{s,T} = \mathrm{Prob}_{s,\pi(T)}$ with model parameters $\pi(T)$ given by

$$\pi(T) = \mathrm{Est}_s\left(\bigcup_{G\in T}\bigcup_{n\in G}\mathrm{Ext}_s(G,n)\right),$$

and we define the probability associated with a node $n$ in $G$ by:

$$\mathrm{Prob}_{s,T}(G,n) = \prod_{a\in\mathrm{Ext}_s(G,n)}\mathrm{Prob}_{s,T}(a).$$

In a probabilistic language model, a graph is viewed as the result of a series of decisions taken during the generation of the graph, eg, the choice of root words, complement frames, complements, adjuncts, landing sites, word order, fillers, etc. This means that a language model can be viewed as a set of submodels corresponding to the different kinds of decisions made during the generation of the graph.

**Definition 6.4** A *language model* is a set $M = s_1, \ldots, s_k$ of submodels. Given a treebank $T$, the number $\mathrm{Prob}_{M,T}(G,n)$ defined by

$$\mathrm{Prob}_{M,T}(G,n) = \prod_{i=1}^{k}\mathrm{Prob}_{s_k,T}(G,n)$$

is called the *probability* of $n$ in $G$ under the model $M$, and the number $\mathrm{Prob}_{M,T}(G)$ defined by

$$\mathrm{Prob}_{M,T}(G) = \prod_{n\in G}\mathrm{Prob}_{M,T}(G,n)$$

is called the *probability* of $G$ under model $M$. The distribution $\mathrm{Prob}_{M,T}(G)$ is called *defective* if it does not define a probability distribution on the set of all finite DG graphs, ie, if it fails to integrate to 1 and assign a non-negative probability to all graphs.

In actual computations, it is convenient to make calculations with probabilities under a minus log transform (with base 10) in order to avoid numerical underflow. That is, instead of using multiplicative probabilities, we will use the corresponding additive costs.

| **Operator** | **Produces** |
|---|---|
| $\mathsf{id}(i, n, t)$ | augmentation assigning identifier $i$ and types $t$ to node $n$ |
| $\mathsf{attr}(n, a, v, t, f)$ | augmentation that assigns attribute value $v$ with attribute types $t$ to the attribute $a$ in node $n$ with filter $f$ |
| $\mathsf{setof}(x, s, \phi)$ | the union of all augmentations produced by the augmentation operator $\phi$ when setting the variable $x$ to some object matching the type specification $s$ |
| $\mathsf{concat}(\phi)$ | augmentation consisting of the concatenation of all augmentations produced by $\phi$ |
| $\phi \mid \psi$ | union of augmentations in $\phi$ and $\psi$ |
| $\phi + \psi$ | set of all augmentations $\mathsf{concat}(a \mid b)$ with $a \in \phi$ and $b \in \psi$ |

Figure 6.1: Augmentation operators used in extraction procedures.

**Definition 6.5** The number

$$\text{Cost}_{s,T}(G, n) = -\log(\text{Prob}_{s,T}(G, n))$$

is called the *cost* at $n$ in $G$ under submodel $s$. Similarly,

$$\text{Cost}_{s,T}(G) = -\log(\text{Prob}_{s,T}(G))$$

is called the *cost* of $G$ under submodel $s$, and

$$\text{Cost}_{M,T}(G) = -\log(\text{Prob}_{M,T}(G))$$

is called the *cost* of $G$ under the model $M$.

We will now introduce the DG modelling language, which is used to specify language models, submodels, and their associated extraction procedures, estimation procedures, and evaluation procedures — including the choice of distribution, estimation method, prior distributions, hierarchies, credibility weights, smoothing, etc.

## Specifying augmentations and augmentation filters

We start by defining the augmentation operators that are used to specify extraction procedures. The operators are summarized in Figure 6.1.

**Definition 6.6** The operator $\mathsf{id}(i, n, t)$ is defined as follows: (a) if the type specifications $n$ and $i$ have a unique match, the operator returns the set $\{\alpha\}$

consisting of the augmentation $\alpha$ that assigns the identifier given by $i$ and the types given by $t$ to the node given by $n$; (b) otherwise, the operator returns the empty set $\emptyset$. The $t$ argument is optional.

**Definition 6.7** The operator $\mathsf{attr}(n, a, v, t, f)$ is defined as follows: (a) if the type specifications $n$ and $a$ have a unique match, the operator returns the set $\{\alpha\}$ consisting of the augmentation $\alpha$ that specifies that the attribute $a$ for the node given by $n$ has attribute types $t$ and equals the value $v$ after it has been filtered through the optional filter $f$ (cf. Pseudocode 6.19); (b) otherwise, the operator returns the empty set $\emptyset$. The $t$ and $f$ arguments are optional.

**Definition 6.8** The operator $\phi \mid \psi$ returns the union of the augmentation sets produced by $\phi$ and $\psi$.

**Definition 6.9** The operator $\mathsf{setof}(x, s, \phi)$ returns the union $A_1 | \ldots | A_k$ where $A_i$ is the augmentation set produced by the expression $\phi$ after the variable $x$ has been set to the object $s_i$, where $s_1, \ldots, s_k$ are the objects matched by the type specification $s$.

**Definition 6.10** The operator $\phi + \psi$ returns the augmentation set $A$ consisting of the concatenation of all pairs $a, b$, where $a$ is some augmentation in the augmentation set produced by $\phi$, and $b$ is some augmentation in the augmentation set produced by $\psi$.

**Definition 6.11** The operator $\mathsf{concat}(\phi)$ returns the concatenation of all the augmentations in the augmentation set produced by $\phi$.

The following examples show how augmentation operators can be used to extract different sets of augmentations from a graph.

**Example 6.12** The expression below can be used to extract all triples consisting of a governor category, a complement role, and a complement category for a given node $node.

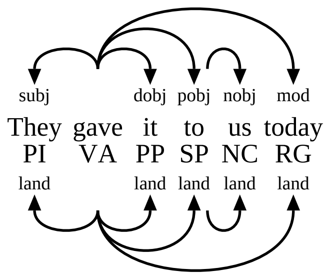


Figure 6.2: Example dependency graph.

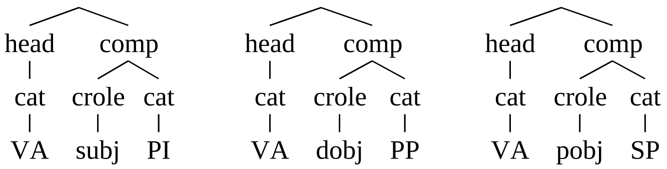


Figure 6.3: Augmentations produced from Figure 6.2 by the extraction procedure in Example 6.12.

id(head, \$node)
+ attr(head, cat, val(cat, \$node))
+ setof(\$comp, $[\xleftarrow{\text{comp}}$ \$node$]$,
    id(comp, \$comp)
    + attr(comp, crole, etype($[\xrightarrow{\text{comp}}$ \$comp$]$))
    + attr(comp, cat, val(cat, \$comp)))

When applied to the dependency graph in Figure 6.2 at the node "gave", this expression produces the augmentations shown in Figure 6.3. The top layer in the graph indicates node identifiers, the second layer indicates attribute names, and the third layer indicates attribute values.

**Example 6.13** The expression below can be used to extract the category and entire complement frame for a node $node in a single augmentation, rather than extracting each individual complement dependency in a separate augmentation as in Example 6.12. The expression specifies that each complement node is labeled by its complement edge type, so there is no need to encode the complement edge type as an independent attribute.

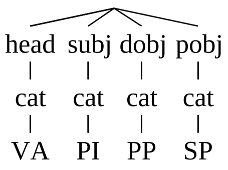


Figure 6.4: Augmentation produced from Figure 6.2 by the extraction procedure in Example 6.13.

| Operator | Produces |
|---|---|
| aug() | matches the input augmentation in an augmentation filter |
| subaug$(n, a, \phi)$ | subaugmentation of $\phi$ containing all nodes that match $n$ and all attributes that match $a$ |
| rename$(n, n', \phi)$ | augmentation $\phi$ where unique node matching $n$ is renamed as $n'$ |
| rename$(n, a, a', \phi)$ | augmentation $\phi$ where the attribute $a$ is renamed as $a'$ in all nodes matching $n$ |

Figure 6.5: Augmentation operators used in augmentation filters.

$$\begin{aligned}
&\mathsf{id}(\mathsf{head}, \mathsf{\$node})\\
&+ \mathsf{attr}(\mathsf{head}, \mathsf{cat}, \mathsf{val}(\mathsf{cat}, \mathsf{\$node}))\\
&+ \mathsf{concat(setof(\$comp,}\ [\xleftarrow{\mathsf{comp}} \mathsf{\$node}],\\
&\qquad \mathsf{id}(\mathsf{etype(\$comp)}, \mathsf{\$comp})\\
&\qquad + \mathsf{attr}(\mathsf{etype(\$comp)}, \mathsf{cat}, \mathsf{val}(\mathsf{cat}, \mathsf{\$comp}))))
\end{aligned}$$

When applied to the dependency graph in Figure 6.2 at the node "gave", it produces the single augmentation shown in Figure 6.4.

When computing the prior probability of the extracted augmentation, we often need to map the extracted augmentation into a subaugmentation in which the nodes and attributes are named according to the conventions of the prior submodel. For this reason, we need a set of augmentation operators (shown in Figure 6.5) for extracting a subaugmentation from an augmentation and for renaming nodes and attributes in the augmentation.

**Definition 6.14** The operator subaug$(n, a, \phi)$ returns the subaugmentation of the augmentation $\phi$ containing all attributes matching $a$ at all nodes matching $n$.

**Definition 6.15** The operator rename$(n, n', \phi)$ produces an augmentation

$\phi'$ from the augmentation $\phi$ by renaming the unique node matching $n$ as $n'$. The operator $\mathsf{rename}(n, a, a', \phi)$ produces an augmentation $\phi'$ from the augmentation $\phi$ by renaming the unique attribute matching $a$ as $a'$ in all nodes matching $n$.

**Example 6.16** Let $A$ = [head.cat=VA, comp.crole=subj, comp.cat=PI] be the augmentation shown in Figure 6.3 (left). Then $\mathsf{subaug}(\mathsf{any}, \mathsf{any}, A)$ produces the subaugmentation $A$, $\mathsf{subaug}(\mathsf{any}, \mathsf{cat}, A)$ produces the subaugmentation [head.cat=VA, comp.cat=PI], and $\mathsf{subaug}(\mathsf{comp}, \mathsf{any}, A)$ produces the subaugmentation [comp.crole=subj, comp.cat=PI]. Similarly, the operator $\mathsf{rename}(\mathsf{comp}, \mathsf{dep}, A)$ produces the augmentation [head.cat=VA, dep. crole=subj, dep.cat=PI], and $\mathsf{rename}(\mathsf{any}, \mathsf{cat}, \mathsf{class}, A)$ produces augmentation [head.class=VA, comp.crole=subj, comp.class=PI].

## Specifying language models and submodels

We will now describe how language models and submodels are specified in DG. The extraction procedure is specified by means of augmentation operators, and the estimation and evaluation procedures are specified by stating the underlying distributions, augmentation filters, prior distributions, etc. Submodels that are computed from other submodels without independent estimation of parameters are called *complex submodels*, whereas submodels that perform their own parameter estimation from the training data are called *simple submodels* or simply *distributions*.

**Pseudocode 6.17** The command distribution(*name*, *vars*, *submodel*) specifies that the submodel specification *submodel* defines a simple submodel named *name* with input variables *vars*. The command complex(*name*, *vars*, *vardecls*, *submodel*) specifies that the submodel specification *submodel* defines a complex submodel named *name* with input variables *vars* and an optional list *vardecls* of variable declarations. The command model(*name*, $sm_1$, $sm_2$, . . .) specifies that the submodels named $sm_1$, $sm_2$, etc. constitute a language model named *name*. The command usemodel(*name*) specifies that the language model named *name* is the current language model.

Language models and submodels are specified by stating their underlying extraction procedures, distributions, and other parameters. Some parameters are shared by all submodels, while other parameters only apply to specific distributions. We will now describe the parameters in detail.

**Pseudocode 6.18** A *submodel specification* has the form *sm*->*par*$_1$...->*par*$_n$, where *sm* is a *submodel type* and *par*$_1$, ..., *par*$_n$ are parameter specifications. The parameter specifications (a)–(b) below apply to all submodel types.

(a) where(*cond*): the submodel only applies to nodes where the type specification *cond* is satisfied.

(b) error_threshold_total(*cost*): indicate an error at the current node if the total cost produced by the submodel exceeds *cost*.

The parameter specifications (c)–(l) below apply to all distributions.

(c) augmentation(*vardecls*, *aug*): use augmentation *aug* as extraction procedure, with variable declarations *vardecls*.

(d) new(*ntype*, *atype*): attributes with type *atype* at nodes with type *ntype* are interpreted as new variables (ie, $X$ variables) in the conditional random vector $X|Y$.

(e) given(*ntype*, *atype*): attributes with type *atype* at nodes with type *ntype* are interpreted as given variables (ie, $Y$ variables) in the conditional random vector $X|Y$.

(f) hierarchy(*ntype*, *atype*, *hierarchy*): attributes with type *atype* at nodes with type *ntype* are assigned hierarchy *hierarchy*.

(g) train(*vardecls*): if the first input variable is set to the current node, the remaining input variables in the training data set can be computed from the current node by means of *vardecls*.

(h) trainwhere(*cond*): train the distribution on nodes where the type specification *cond* is satisfied; defaults to the value of the where parameter.

(i) feed(*dist*, *filter*): add the augmentations produced by the augmentation filter *filter* to the training data for the distribution *dist*; the filter input consists of the training variables and augmentations used for this distribution.

(j) prior(*submodel*): use the complex submodel *submodel* as prior distribution for this distribution.

(k) credibility_function(*credfunc*): use the function *credfunc* as a credibility function for this distribution; *credfunc* must map a data count in $[0, \infty[$ into a credibility weight in $[0, 1]$.

(l) default(*ntype*, *atype*, *default*): undefined attributes with type *atype* at nodes with type *ntype* are assigned default value *default*.

**Pseudocode 6.19** The submodel type empirical() models an empirical distribution. In addition to the general parameters for distributions, it also supports the parameters (a)–(c) below, which allow the empirical distribution to function as a filter for unknown words (cf. Definition 6.7):

(a) known_mincount(*mincount*): an outcome is marked as unknown if there are fewer than *mincount* observations of it.
(b) known_minprob(*minprob*): an outcome is marked as unknown if it occurs with a probability smaller than *minprob*.
(c) unknown_marker(*marker*): unknown outcomes are replaced by *marker*.

**Pseudocode 6.20** The submodel type hpm() models a hierarchical partition model. In addition to the general parameters for distributions, it also supports the parameters (a)–(c) below:

(a) partition_mincount(*mincount*): do not create partitions with fewer than *mincount* observations.
(b) partition_minprob(*minprob*): do not create partitions with a probability smaller than *minprob*.
(c) error_threshold_partition(*cost*): indicate an error at the current node if the cost corresponding to the choice of partition exceeds *cost*.

**Pseudocode 6.21** The submodel type xhpm() models an XHPM distribution. In addition to the parameters for hpm distributions, it also supports the parameters (a)–(k) below:

(a) c(*ntype*, *atype*): attributes with type *atype* at nodes with type *ntype* belong to the variables $C$ or $C'$ in the XHPM model.
(b) x(*ntype*, *atype*): attributes with type *atype* at nodes with type *ntype* belong to the variables $X$ or $X'$ in the XHPM model.
(c) e(*ntype*, *atype*): attributes with type *atype* at nodes with type *ntype* belong to the variable $E'$ in the XHPM model.
(d) cnew_prior(*ntype*, *atype*, *submodel*): the complex submodel given by the submodel specification *submodel* is used as a prior distribution for each individual $C$ variable with node type *ntype* and attribute type *atype*; if *ntype* and *atype* are omitted, then *submodel* is interpreted as a joint prior distribution for all $C$ variables.
(e) cgiven_prior(*ntype*, *atype*, *submodel*): the complex submodel given by the submodel specification *submodel* is used as a prior distribution for

each individual $C'$ variable with node type *ntype* and attribute type *atype*; if *ntype* and *atype* are omitted, then *submodel* is interpreted as a joint prior distribution for all $C'$ variables.

(f) xnew_prior(*ntype*, *atype*, *submodel*): the complex submodel given by the submodel specification *submodel* is used as a prior distribution for each individual $X$ variable with node type *ntype* and attribute type *atype*; if *ntype* and *atype* are omitted, then *submodel* is interpreted as a joint prior distribution for all $X$ variables.

(g) xgiven_prior(*ntype*, *atype*, *submodel*): the complex submodel given by the submodel specification *submodel* is used as a prior distribution for each individual $X'$ variable with node type *ntype* and attribute type *atype*; if *ntype* and *atype* are omitted, then *submodel* is interpreted as a joint prior distribution for all $X'$ variables.

(h) xnew_est(*dist*, *filter*): the distribution *dist* is used to estimate parameters for $X$, using augmentation filter *filter*.

(i) xgiven_est(*dist*, *filter*): the distribution *dist* is used to estimate parameters for $X'$, using augmentation filter *filter*.

(j) xnew_cred(*ntype*, *atype*, *credfunc*): use the function *credfunc* as a credibility function for each individual $X$ variable with node type *ntype* and attribute type *atype*; if *ntype* and *atype* are omitted, then *credfunc* is interpreted as a credibility function for all $X$ variables.

(k) xgiven_cred(*credfunc*): use the function *credfunc* as a credibility function for each individual $X'$ variable with node type *ntype* and attribute type *atype*; if *ntype* and *atype* are omitted, then *credfunc* is interpreted as a credibility function for all $X'$ variables.

**Pseudocode 6.22** The submodel type champ() models a projective Champernowne distribution. Its augmentations must consist of a single attribute with a non-negative number as its value.

**Pseudocode 6.23** The submodel type egain() models the expected gain in a Champernowne distribution. The probability produced by egain() depends on the time where the current node was added to the graph, the time where the last node was added to the graph, and the time of the last attempt to reanalyze the current node with respect to the cost function containing the call to egain().

**Pseudocode 6.24** A complex submodel is any expression of the form (a)–

(f) described below:

(a) *number*: any number.
(b) prob(*smname*, *vars*): the probability produced by submodel named *smname* when applied to the input variables *vars*.
(c) prob(*smname*, *filter*): the probability produced by submodel named *smname* when applied to the augmentation produced by filter *filter*.
(d) *expr1* + *expr2*, *expr1* - *expr2*, *expr1* * *expr2*, *expr1* / *expr2*: the sum, difference, product, or quotient of any two complex submodels *expr1* and *expr2*.
(e) sum(*var*, *range*, *vardecls*, *expr*): the sum of the probabilities produced by *expr* when setting the variable *var* to each of the objects matched by the type specification *range*; *vardecls* is an optional list of variable declarations computed from *var*.
(f) product(*var*, *range*, *vardecls*, *expr*): the product of the probabilities produced by *expr* when setting the variable *var* to each of the objects matched by the type specification *range*; *vardecls* is an optional list of variable declarations computed from *var*.

In this section, we have defined a language for specifying probabilistic language models in DG with empirical distributions, HPM and XHPM distributions, projective Champernowne distributions, and arithmetic combinations of such distributions. We have also seen how extraction procedures can be specified by means of augmentation operations. In the following section, we will demonstrate how this model language can be used to specify a probabilistic language model for DG.

## 6.2 A probabilistic language model for DG

***Summary***. *We propose a probabilistic language model for DG. Following Eisner (1996), Collins (1997), and Charniak (2000), we assume that the graph is created by a stochastic generative process that elicits probabilities at each choice point. We describe this process in detail and how it can be modelled by a probabilistic language model for DG.*

In this section, we will propose a language model for DG inspired by the methodology proposed by Eisner (1996), Collins (1997), and Charniak (2000). We will follow these authors by formulating the language model generatively, ie, by describing the probability of the entire graph as a sequence of probabilities generated by different choices during the generation of the graph.[1] Since DG is not context-free and must deal with both discontinuous word orders, fillers, gaps, and anaphora, our generative stochastic procedure differs from the context-free generative stochastic procedure proposed by these authors in most of the details, although the general philosophy is the same.

### The generative stochastic procedure

Eisner (1996), Collins (1997), and Charniak (2000) assume that a context-free graph is generated by a top-down procedure which first creates the root node in the graph, and then recursively expands each newly created node by generating its context-free complements and adjuncts in inside-out order. We will follow this general methodology by assuming that DG graphs are generated by a top-down procedure which expands each newly created node by identifying its landing site, its complements and adjuncts, its associated simple fillers and gapping fillers, its antecedents, etc. In order to account for partial analyses, we will also introduce a mechanism for generating deep roots.

The DG graph generation procedure is shown in Figure 6.6. We assume that the graph contains a formal top node TOP (cf. Remark 7.67), and that the graph is generated in two top-down passes starting at TOP: a primary top-down pass (shown in Figure 6.7) that generates landing sites, fillers, complements, non-punctuation adjuncts, and gapping conjuncts, and a secondary top-down pass (shown in Figure 6.8) that generates deep roots and punctuation marks, and identifies antecedents for anaphora. The order in which these steps occur is important because the choice made at a given step can be conditioned on all prior choices in the generation. In the following remarks, we will describe our reasons for some of these design decisions. We will then give a detailed description of the individual steps in the generative procedure and the associated probabilities.

**Remark 6.25 (landing sites and word order)** Since a node's transitive governors constitute its potential landing sites, we cannot select a landing site

[1]The generative model used for calculating the probability of a graph is a convenient way of specifying the statistical dependencies between different aspects of the graph. It does not say anything about how analyses are generated incrementally from a semantic representation when a speaker produces an utterance.

```
procedure stochastic graph generation
begin
    generate primary top-down expansion of TOP (cf. Figure 6.7);
    generate secondary top-down expansion of TOP (cf. Figure 6.8);
    return generated graph and probability;
end
```

Figure 6.6: Our stochastic graph generation procedure.

```
PRIMARY TOP-DOWN EXPANSION OF NODE
    1. Identify landing site
        1a. Identify landing site and relative word order
    2. Generate and expand complements and simple fillers
        2a. Select complement frame
        2b. Generate fillers
        2c. Consume or pass on fillers
        2d. Generate and expand complements
    3. Generate and expand non-punctuation adjuncts
        3a. Generate and expand non-punctuation adjuncts
    4. Generate and expand gaps
        4a. Generate and expand gapping conjuncts
```

Figure 6.7: The steps in the primary top-down expansion of a node in our stochastic graph generation procedure.

for a node before all of its transitive governors have been generated. Moreover, when choosing a landing site for a node, the choice of landing site does not only depend on the landed node and the landing site, but also on the presence of other landed nodes at the landing site. In the generative process, it is therefore natural to assume that nodes are given landing sites in a top-down manner, that landed nodes are added to a landing site one by one, and that the probability of adding a new landed node depends on the previously added landed nodes at the landing site. We must therefore specify the order in which the nodes in the tree are created and given a landing site. We will return to this question in Remark 6.32.

**Remark 6.26 (fillers)** In our top-down expansion of nodes, fillers are pass-

```
SECONDARY TOP-DOWN EXPANSION OF NODE
    1. Generate and expand deep roots
        1a. Generate deep roots if node is at the right periphery
        1b. Generate primary expansion of deep roots
    2. Identify antecedents
        2a. Identify antecedents for anaphoric frames
        2b. Identify antecedents for missing complements
    3. Generate punctuation marks
        3a. Generate punctuation marks at node
        3b. Identify landing sites for punctuation marks
    4. Secondary top-down expansion of subnodes
        4a. Secondary top-down expansion of dependents left-right
        4b. Secondary top-down expansion of deep roots left-right
```

Figure 6.8: The steps in the secondary top-down expansion of a node in our stochastic graph generation procedure.

ed down from a node to its dependents, resembling the way Collins deals with long-distance extraction in his Model 3. This means that after a node has chosen its complement frame, it decides (a) whether it should create any new fillers, (b) whether any of its inherited or newly created fillers should be used to satisfy a local complement or adjunct role, and (c) how it should pass on the remaining fillers to its complements and adjuncts. First when these decisions have been made, it starts generating its complements and adjuncts. In this way, the choice of complement and adjunct lexemes can depend on the set of fillers received from the governor.

**Remark 6.27 (adjuncts)** The choice of an adjunct is often syntactically or semantically constrained by the choice of complements. For example, the choice of a relative clause depends on the choice of relativized NP, the choice of a subject-oriented modifier depends on the choice of subject, etc. For this reason, it is reasonable to assume that the adjuncts at a node are created after all the complements have been created.

**Remark 6.28 (gaps)** Since a gap is essentially a copy of an entire subtree where at least two of the branches have been replaced with gapping dependents, the gap licensor should create all its other dependents before it creates any gapping conjuncts.

**Remark 6.29 (deep roots)** In order to account for analyses of partial input (cf. section 7.4), we will assume that a landing site at the right frontier of the graph can generate a set of temporary landed nodes that lack a governor. Being deep roots that are not yet incorporated into the dependency tree, it is natural to assume that these nodes are generated after all fillers, complements, adjuncts, and gaps have been generated.

**Remark 6.30 (antecedents)** When choosing one or more antecedents for an anaphor, we must determine the set of potential antecedents in the graph, compute a weight that quantifies their relative probability, and select the most probable antecedent. The relative probability of each antecedent depends on almost all other aspects of the graph: the deep path from anaphor to antecedent, the word order of the nodes on the path, the antecedents assigned to preceding anaphora, etc. (cf. Jurafsky and Martin 2000, pp. 671ff). For this reason, it is most natural to model anaphor resolution as a process that is carried out after the generation of the entire deep tree and surface tree in the graph.

**Remark 6.31 (punctuation)** In section 4.6, we saw that a punctuation mark at a higher node can satisfy the punctuation requirements of one or more lower nodes, and that a punctuation mark at a lower node only rarely satisfies a punctuation mark at a higher node (cf. Remark 4.70). For this reason, in our generative stochastic model, punctuation marks at higher nodes should be created before punctuation marks at lower nodes. Moreover, since punctuation marks occur exclusively in written language, it is reasonable to assume that all other aspects of the graph are completely independent of the punctuation. For these reasons, we will assume that punctuation marking is performed as the last step in the secondary top-down expansion of a node.

**Remark 6.32 (derivation ordering of subnodes)** The probability of a graph is the sum of the probabilities of all its derivations. We can therefore simplify the computation of graph probabilities by ensuring that each graph has a unique derivation. In the DG language model, this can be achieved by defining a derivation ordering on the set of all nodes in the graph. This derivation ordering (which in general does not coincide with the linear ordering) can be constructed by assuming that each node imposes an ordering on its local complements, adjuncts, fillers, gapping fillers, gapping

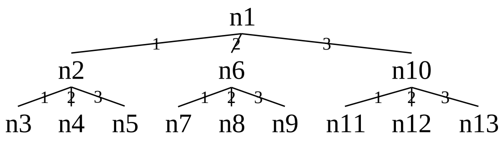


Figure 6.9: Top-down depth-first ordering of nodes in the graph induced from a local ordering of the subnodes.

dependents, and deep roots. The local ordering can be used to define a global ordering on the entire graph if we assume that the nodes are visited in a top-down depth-first search with local node ordering, as illustrated in Figure 6.9. It is hard-coded in the DG language model that complements are always visited first, then adjuncts, gaps, and finally deep roots.

Since we can only condition a decision on the previous decisions in the derivation, the local derivation ordering at a node is important, at least to the extent that the language model is sophisticated enough to take advantage of knowledge about earlier decisions. However, within these bounds, there are many sensible orderings, and it will take some experimentation to identify the best ordering — an issue that we leave for further research.

One idea is to use a *scope ordering* based on the functor-argument application ordering in a semantically annotated graph. Another idea, which we prefer because the Danish Dependency Treebank does not encode semantics and scope, is motivated by the desire to add landed nodes to their landing site in inside-out order, as in Collins (1997). Because DG graphs can be discontinuous, it is not possible to define a computationally feasible language model that ensures that landed nodes are always added in inside-out order, but we can ensure that landed nodes tend to be added in inside-out order by ordering dependent roles according to the average number of landed nodes at the landing site between the node and the landing site; we will refer to this ordering as *average proximity ordering*.

In the following, we will give a detailed description of the individual steps in the primary and secondary top-down expansion and the associated probabilities. The detailed DG encoding of the individual steps in the language model is listed in Appendix A.

**Primary expansion 1a. Identify landing site and relative word order**

The first step in the primary expansion of a node $N$ (cf. submodel `pri1a` on p. 361) is to choose a landing site $L$ for $N$ among the transitive governors of $N$, and a linear ordering $\omega$ that indicates where $N$ is placed in the landing field of $L$ relative to the previously landed nodes at $L$. By assigning a probability weight $W_{\text{pri1a}}(L,\omega)$ to each choice of $(L,\omega)$, we can define a probability distribution $P_{\text{pri1a}}$ on $(L,\omega)$ by:

$$P_{\text{pri1a}}(L,\omega) = \frac{W_{\text{pri1a}}(L,\omega)}{\sum_{L,\omega} W_{\text{pri1a}}(L,\omega)}.$$

The probability weight $W_{\text{pri1a}}(L,\omega)$ must quantify how well-formed the choice is with respect to word order and island constraints in the extraction path. In order to capture these intuitions, we define $W_{\text{pri1a}}$ by:

$$\begin{aligned} W_{\text{pri1a}}(L,\omega) = {} & P_{\text{worder}}(N|L, F^{\omega}, C_{-}^{\omega}, A_{-}^{\omega}, A_{+}^{\omega}, C_{+}^{\omega}) \\ & \cdot \prod_{\omega' \neq \omega} P_{\text{worder}}(\text{STOP}|L, F^{\omega'}, C_{-}^{\omega'}, A_{-}^{\omega'}, A_{+}^{\omega'}, C_{+}^{\omega'}) \\ & \cdot \prod_{(I \to D) \in \text{ExtPath}(N)} \min\left(1, \frac{P_{\text{ldepext}}(D,I|N)}{P_{\text{ldep}}(D,I)}\right) \end{aligned}$$

The first two factors quantify the word order probability. At position $\omega$, we insert $N$ with probability $P_{\text{worder}}(N|\dots)$; at any other position $\omega'$ we insert the dummy node STOP with probability $P_{\text{worder}}(\text{STOP}|\dots)$, in order to indicate that in this stage of the derivation, no nodes are inserted at these positions. The insertion of STOP at the alternative positions makes it possible to detect the absence of an obligatory landed node. The variables $(F^{\omega}, C_{-}^{\omega}, A_{-}^{\omega}, A_{+}^{\omega}, C_{+}^{\omega})$ indicate the word order context at $\omega$: $F^{\omega}$ indicates whether $\omega$ is to the left or right of $L$, $C_{-}^{\omega}$ and $C_{+}^{\omega}$ are the immediately preceding and immediately following locally landed complements at $\omega$, and $A_{-}^{\omega}$ and $A_{+}^{\omega}$ are the immediately preceding and immediately following landed nodes at position $\omega$ between $C_{-}^{\omega}$ and $C_{+}^{\omega}$. That is, the local word order context at position $\omega$ is:

$$C_{-}^{\omega} \quad \dots \quad A_{-}^{\omega} \quad \omega \quad A_{+}^{\omega} \quad \dots \quad C_{+}^{\omega}$$

where $L$ is either to the left or the right of all the nodes $C_{-}^{\omega}, A_{-}^{\omega}, A_{+}^{\omega}, C_{+}^{\omega}$.

When creating the training data for $P_{\text{worder}}$, we generate an instance of $N$ for every landed node at $L$, and for any empty position between two

| $L$ | $F$ | $C_-$ | $A_-$ | $N$ | $A_+$ | $C_+$ |
|---|---|---|---|---|---|---|
| $\text{met}_0^{\text{root}}$ | left | – | – | STOP | – | $\text{we}_1^{\text{subj}}$ |
| $\text{met}_0^{\text{root}}$ | left | – | – | $\text{we}_1^{\text{subj}}$ | – | – |
| $\text{met}_0^{\text{root}}$ | left | $\text{we}_1^{\text{subj}}$ | – | STOP | $\text{just}_3^{\text{temp}}$ | – |
| $\text{met}_0^{\text{root}}$ | left | $\text{we}_1^{\text{subj}}$ | – | $\text{just}_3^{\text{temp}}$ | – | – |
| $\text{met}_0^{\text{root}}$ | left | $\text{we}_1^{\text{subj}}$ | $\text{just}_3^{\text{temp}}$ | STOP | – | – |
| $\text{met}_0^{\text{root}}$ | right | – | – | STOP | $\text{him}_2^{\text{dobj}}$ | – |
| $\text{met}_0^{\text{root}}$ | right | – | – | $\text{him}_2^{\text{dobj}}$ | – | – |
| $\text{met}_0^{\text{root}}$ | right | $\text{him}_2^{\text{dobj}}$ | – | STOP | – | – |

Figure 6.10: Training instances for $P_{\text{worder}}$ generated from the landing frame "$\text{we}_1^{\text{subj}}$ $\text{just}_3^{\text{temp}}$ $\text{met}_0^{\text{root}}$ $\text{him}_2^{\text{dobj}}$".

landed nodes at $L$, ie, the phrase "we just met him" will generate the training data shown in Figure 6.10. The local word order context shown above is sufficient to rule out many incorrect word orders. For example, the context can be used to deduce that $N$ is unlikely to be an adjunct or a direct object if it occurs before a subject $C_+$, or that $N$ is unlikely to be a relative clause unless $A_+$ and $C_+$ are empty.

The third factor quantifies the probability of the extraction path by assigning a weight in $[0,1]$ to each edge in the extraction path from $N$ to $L$. The intuition is to measure how often the deep edge $I \rightarrow D$ occurs in an extraction path for a node like $N$ relative to how often it occurs in the graph in general. If this ratio is smaller than 1, then $I \rightarrow D$ occurs in extraction paths for $N$ below chance level, ie, it acts as an island whose strength is inversely proportional with the ratio. If the probability ratio is larger than 1, then $I \rightarrow D$ occurs above chance level, ie, it is not an island edge, and we assign it the ratio 1 in order to ensure that the weight assigned to an extraction path decreases monotonically with the path length.

The distributions $P_{\text{worder}}$, $P_{\text{ldepext}}$, and $P_{\text{ldep}}$ are estimated with XHPM distributions, using as attributes the lexeme, deep edge type, surface edge type, and locality properties (ie, whether the node has been extracted or not) for each node, using the hierarchies for lexemes and edge types defined in the lexicon. The DG encoding of these distributions is shown in Appendix A.

### Primary expansion 2a. Select complement frame

In step 2a of the primary expansion, the node $N$ must choose a complement frame. This choice is modelled by means of the XHPM distribution $P_{\text{cframe}}$ which encodes the node's complement frame as an empirically distributed variable that is conditioned on the node's lexeme, deep role, and surface role. The DG encodings for the submodels $P_{\text{pri2a}}$ and $P_{\text{cframe}}$ are shown in Appendix A.

### Primary expansion 2b. Generate fillers

In step 2b of the primary expansion, the node $N$ must choose which fillers it is going to generate. This is modelled as a process where $N$ is presented with a sequence $\mathcal{S}$ of potential filler sources in the graph, and a set $\mathcal{T}$ of possible filler types. For each filler source $S \in \mathcal{S}$ and filler type $T \in \mathcal{T}$, $N$ must then determine whether it generates a filler for $S$ with type $T$. The choice is conditioned on the choices made at the previous filler source $S_{\text{prev}}$ in $\mathcal{S}$. The filler sources are presented in outside-in order, starting with the filler sources to the left of $N$ in left-right order, then the filler sources to the right of $N$ in right-left order, and finally the complements of $N$ (which have not received a landing site yet) in generation order. That is, if $C_{S,T}$ encodes whether the filler was created or not, the probability $P_{\text{pri2b}}(N)$ of the choices made at $N$ with respect to filler generation can be written as:

$$P_{\text{pri2b}}(N) = \prod_{S \in \mathcal{S}} \prod_{T \in \mathcal{T}} P_{\text{fgen}}(C_{S,T} | N, S, T, S_{\text{prev}})$$

where $P_{\text{fgen}}(C_{S,T} | N, S, T, S_{\text{prev}})$ denotes the probability of the generation choice $C_{S,T}$ given $N, S, T, S_{\text{prev}}$. The set $\mathcal{S}$ of potential filler sources is computed by means of a set of intermediate nodes $\mathcal{I}$, as follows: (a) $\mathcal{I}$ consists of any node $I$ that equals $N$, the governor of $N$, or the grand-governor of $N$ if $N$ is the head of a relative clause that modifies a subject; (b) $\mathcal{S}$ consists of any node $S$ which has some $I \in \mathcal{I}$ in its extraction path.

The distribution $P_{\text{fgen}}$ is modelled as an XHPM distribution where the choice $C_{S,T}$ is conditioned on a number of features that capture the most relevant contextual information from our linguistic analyses of control, parasitic gaps, elliptic coordinations, and other filler constructions in sections 2.5, 4.3, and 4.4: (a) the lexeme, deep role, surface role, and complement frame of $N$; (b) whether $S$ is local or extracted, and whether $S$

precedes its landing site or not; (c) the upwards and downwards part of the path from $N$ to $S$, and whether $N$ precedes $S$ or not; (d) whether $N$ created a filler with type $T$ for $S_{\text{prev}}$, whether some other node created a filler with type $T$ for $S_{\text{prev}}$, and whether the upwards part of the path from $N$ to $S$ equals the upwards path from $N$ to $S_{\text{prev}}$. At the dependents of $N$, some of these variables are undefined.

**Primary expansion 2c. Consume or pass on fillers**

In step 2c of the primary expansion, $N$ must determine how it will dispose of the fillers received from its governor or generated by itself, ie, whether it will analyze a given filler as its own complement or adjunct, or pass on the filler to one of its dependents. This is modelled as a process where each filler $F$ in the list $\mathcal{F}$ of fillers is presented to $N$ in generation order so that $N$ can decide whether $F$ should be analyzed as a local complement or adjunct of $N$, or be passed on to one of the complements or adjuncts of $N$. That is, the probability $P_{\text{pri2c}}(N)$ of the choices made at $N$ with respect to filler consumption can be written as:

$$P_{\text{pri2c}}(N) = \prod_{F \in \mathcal{F}} \frac{P_{\text{fdispose}}(C, D | N, F, \chi_F)}{\sum_{C,D} P_{\text{fdispose}}(C, D | N, F, \chi_F)}$$

where $\chi_F$ is the set of empty complement roles at $N$ when $F$ is presented to $N$, the variable $C$ encodes whether the filler is consumed by $N$ or passed on to a dependent, and the variable $D$ encodes the dependent role which $F$ is used to fill, or is passed on to.

The distribution $P_{\text{fdispose}}$ is modelled as an XHPM distribution where the choice $(C, D)$ is conditioned on the filler type, the deep path from the filler licensor to the filler, and on the lexemes associated with $N$, the filler $F$, and its filler licensor.

**Primary expansion 2d. Generate and expand complements**

In step 2d of the primary expansion, $N$ must generate complements in all the complement slots that are not occupied by a filler. That is, the probability $P_{\text{pri2d}}(N)$ of the choices made at $N$ with respect to complement generation can be written as:

$$P_{\text{pri2d}}(N) = \prod_{C \in \mathcal{C}} P_{\text{comp}}(C | N, \rho_C)$$

where $\mathcal{C}$ is the set of non-filler complements at $N$, and $\rho_C$ is the complement role associated with complement $C$.

The distribution $P_{\text{comp}}$ is modelled as an XHPM distribution where the choice of complement lexeme is conditioned on the governor lexeme and the complement role. Although an XHPM model can be used to learn statistical dependencies between different complement slots, we use a simpler model where the complements are generated independently of each other. The simple model is justified by Li and Abe (1999), who report that the statistical dependencies between complement roles are rather weak, and therefore difficult to detect.

### Primary expansion 3a. Generate and expand non-punctuation adjuncts

In step 3a of the primary expansion, $N$ must generate all non-punctuation adjuncts, including any adjuncts that received a filler in step 2c. We model this as a process where the governor creates adjuncts one by one: after generating an adjunct role (with probability 1 if the adjunct role was generated by the filler-passing mechanism in step 2c in the primary expansion), the governor chooses an adjunct lexeme, and this process is repeated until the governor generates the special adjunct role STOP. That is, the probability $P_{\text{pri3a}}(N)$ of the choices made at $N$ with respect to the generation of non-puncutation adjuncts can be written as:

$$P_{\text{pri3a}}(N) = P_{\text{arole}}(\text{STOP}|N) \cdot \prod_{A \in \mathcal{A}} \max(\phi_A, P_{\text{arole}}(\rho_A|N)) P_{\text{adj}}(A|N, \rho_A)$$

where $\mathcal{A}$ is the set of adjuncts at $N$, $\rho_A$ is the adjunct role associated with adjunct $A$, and $\phi_A$ is 1 if $A$ was generated by the filler-passing mechanism in step 2c in the primary expansion, and 0 otherwise.

The distribution $P_{\text{arole}}$ is modelled as an XHPM distribution where the choice of adjunct role is conditioned on the governor lexeme. The distribution $P_{\text{adj}}$ is modelled as an XHPM distribution where the choice of adjunct lexeme is conditioned on the governor lexeme and the adjunct role.

### Primary expansion 4a. Generate and expand gaps

In step 4a of the primary expansion, $N$ must generate all its gapping fillers and their gapping dependents. This is modelled as a process where $N$ chooses the number of gapping fillers, the number of gapping dependents for each gapping filler, and the replacement path and lexeme for each gapping dependent. That is, the probability $P_{\text{pri4a}}$ of the choices made at $N$ with respect to gaps can be written as:

$$\begin{aligned}P_{\text{pri4a}}(N) = P_{\text{gapnum}}(\#\mathcal{G}|N) \\ \cdot \prod_{G\in\mathcal{G}} P_{\text{gapdepnum}}(\#\mathcal{D}_G|N,G) \\ \cdot \prod_{D\in\mathcal{D}_G} P_{\text{gappath}}(\pi_D|N,D)P_{\text{gapdep}}(D|\Gamma_D,\rho_D)\end{aligned}$$

where $\mathcal{G}$ is the set of gapping fillers generated by $N$, $\mathcal{D}_G$ is the set of gapping dependents at gapping filler $G$, $\pi_D$ is the path from $N$ to the node $R$ that $D$ replaces, $\Gamma_D$ is the governor of the replaced node $R$, and $\rho_D$ is the dependent role of $R$. The probability $P_{\text{gapdep}}(D|\Gamma,\rho)$ is defined as $P_{\text{comp}}(D|\Gamma,\rho)$ if $\rho$ is a complement role, and as $P_{\text{adj}}(D|\Gamma,\rho)$ if $\rho$ is an adjunct role. This model is a simplification in that we assume that the gapping dependent satisfies the same dependency role as the dependent it replaces.

The distribution $P_{\text{gapnum}}$ is modelled as an XHPM distribution where the number of gaps at $N$ is modelled by an empirical distribution, conditioned on the lexeme associated with $N$. The distribution $P_{\text{gapdepnum}}$ is modelled as an XHPM distribution where the number of gapping dependents at a gap $G$ is modelled as an empirical distribution conditioned on the lexeme associated with $N$. Finally, the distribution $P_{\text{gappath}}$ is modelled as an XHPM distribution where the path from gap licensor to the replaced node is modelled as an empirical distribution conditioned on the lexeme associated with $N$.

### Secondary expansion 1a+b. Generate deep roots if node is at the right periphery, and expand deep roots

In step 1a of the secondary expansion, a node $N$ at the right periphery of the graph can generate zero or more deep roots. This is modelled as a process where $N$ first generates the number of deep roots, and then generates each deep root. That is, the probability $P_{\text{sec1a}}$ of the choices made at $N$ with respect to deep roots can be written as:

$$P_{\text{sec1a}}(N) = P_{\text{drootnum}}(\#\mathcal{R}|N) \cdot \prod_{R\in\mathcal{R}} P_{\text{droot}}(R|N)$$

where $\mathcal{R}$ is the set of surface roots at $N$. The distribution $P_{\text{drootnum}}$ is modelled as an XHPM distribution where the number of surface roots is modelled as an empirical variable that is conditioned on the lexeme associated with $N$. The distribution $P_{\text{droot}}$ is modelled as an XHPM distribution where the lexeme of the deep root $R$ is conditioned on the lexeme for $N$ and the surface role of $R$ at $N$. Step 1b in the secondary expansion performs a primary expansion of each generated deep root.

### Secondary expansion 2a+b. Identify antecedents for anaphoric frames and missing complements

Steps 2a and 2b of the secondary expansion of $N$ identify the antecedents for the anaphoric frames and the missing complements of $N$. Although these steps have not been implemented in the DG language model in Appendix A, step 2a can be modelled as a process where $N$ identifies an antecedent $A_F$ for each anaphor frame $F$ at $N$ from the set $\mathcal{A}_F$ of all possible antecedents for anaphor frame $F$ (including the dummy null antecedent NULL). That is, the probability $P_{\text{sec2a}}$ of the choices made at $N$ with respect to antecedents for anaphora can be written as:

$$P_{\text{sec2a}}(N) = \prod_{F \in \mathcal{F}} \frac{W_{\text{anaphor}}(A_F, \pi_{A_F} | N, F)}{\sum_{A' \in \mathcal{A}_F} W_{\text{anaphor}}(A', \pi_{A'} | N, F)}$$

where $\mathcal{F}$ denotes the set of anaphor frames at $N$, and $\pi_A$ denotes the deep path from $N$ to the antecedent $A$. The weight function $W_{\text{anaphor}}(A, \pi_A | N, F)$ must assign non-negative weights to the possible antecedents that quantify the relative well-formedness of the path from the anaphor to its antecedent, and of the lexemes associated with the anaphor and its antecedent. We imagine that $W_{\text{anaphor}}$ could be modelled as the product of two factors, a factor $W_{\text{antecedent}}$ that measures the well-formedness of the antecedent ignoring the path, and a factor $W_{\text{antepath}}$ that measures the well-formedness of the path from anaphor to antecedent ($W_{\text{antepath}}$ could be constructed in a way that resembles the weight assigned to extraction paths in $W_{\text{pri1a}}$ on p. 278). We leave it as a topic of further research to determine appropriate definitions of $W_{\text{antecedent}}$ and $W_{\text{antepath}}$.

Similarly, step 2b can be modelled as a process where an antecedent is chosen for each empty complement slot $C$ at $N$. That is, the probability $P_{\text{sec2b}}$ of the choices made at $N$ with respect to antecedents for empty complement slots can be written as:

$$P_{\mathrm{sec2b}}(N) = \prod_{C \in \mathcal{C}} \frac{W_{\mathrm{anaphor}}(A_C, \pi_{A_C} | N, C)}{\sum_{A' \in \mathcal{A}_C} W_{\mathrm{anaphor}}(A', \pi_{A'} | N, C)}$$

where $\mathcal{C}$ denotes the set of empty complement slots, $A_C$ denotes the antecedent for complement slot $C$, $\pi_A$ denotes the path from $N$ to antecedent $A$, and $\mathcal{A}_C$ denotes the possible antecedents for complement slot $C$.

**Secondary expansion 3a+b. Generate punctuation marks at node and identify their landing site**

In step 3a of the secondary expansion, the node $N$ must generate its punctuation marks. This is modelled as a process where each position $\omega$ in the current landing field (ie, each position before or after a non-punctuation node) is examined, and zero or more punctuation marks are inserted at $\omega$ by a process that generates punctuation marks until the special punctuation mark STOP is generated. That is, the probabilities of the choices made at $N$ with respect to punctuation can be written as:

$$P_{\mathrm{sec3a}}(N) = \prod_{F \in \mathcal{F}} P_{\mathrm{pnct}}(\mathrm{STOP} | N, \chi_{F,\mathrm{STOP}}) \prod_{P \in \mathcal{P}_F} P_{\mathrm{pnct}}(P | N, \chi_{F,P})$$

where $\mathcal{F}$ denotes the set of punctuation fields at $N$, $\mathcal{P}_F$ denotes the sequence of punctuation marks in the field $F$, and $\chi_{F,P}$ denotes the punctuation context for punctuation mark $P$ in field $F$. The context consists of the non-punctuation nodes to the immediate left and right of the punctuation field $F$, the oldest non-punctuation nodes in the landing field of $N$ to the left and right of $F$, the last open begin mark at $N$ before $P$, the directly preceding punctuation mark before $P$ in $F$, and string representations of the left and right outside punctuation fields of $F$, ie, all punctuation marks that do not land at $F$ but are placed in a punctuation field that is adjacent to $F$, without any intervening non-punctuation nodes.

The distribution $P_{\mathrm{pnct}}(P | N, \chi_{F,P})$ is modelled as an XHPM distribution where the lexeme, deep role, and surface role of $P$ is conditioned on the lexeme, deep role, and surface role of $N$ and the nodes in $\chi_{F,P}$, and on the string representation of the outer peripheries.

In step 3b, the DG language model simplifies the problem by assuming that punctuation marks always land locally, ie, that their landing sites always equal their governors. The DG model is therefore unable to deal with quote transpositions in its present form.

**Secondary expansion 4a+b. Secondary top-down expansion of dependents and deep roots left-right**

Step 4a in the secondary expansion of $N$ performs a secondary expansion of the dependents of $N$ in left-right order; step 4b performs a secondary expansion of the deep roots of $N$ in left-right order.

**Remark 6.33 (robustness and zero probabilities)** In terms of robustness in parsing and learning, zero probabilities are problematic and should be avoided as much as possible in probabilistic language models. First of all, speakers often produce speech errors that violate the grammar, rendering a grammatical analysis of the speech input impossible — yet hearers often understand the intended meaning anyway, ie, they manage to analyze ungrammatical input. Since analyses with zero probabilities are excluded from consideration in parsing and generation, it is evident that if we want our language models to be robust against speech errors, they should not assign zero probability to ungrammatical analyses that could plausibly result from a slip of tongue.

Moreover, computational language learning is often a two-step process where we create a new grammar from an old one by first parsing a corpus with the old grammar, and then inducing a new grammar from the parsed input. In this process, we cannot expect to learn analyses that we have excluded as ill-formed in the old grammar — whereas if an analysis is given a small non-zero probability, there is a chance that it will be selected if no better alternative is around, so that the probabilities can be re-estimated correctly in the subsequent learning phase. From an even more principled point of view, we cannot exclude certain constructions simply because they have not been observed: absence of evidence is not the same thing as evidence of absence. This problem is exacerbated by Zipf's law (Samuelsson 1996; Powers 1998), which tells us that no matter how big a corpus is, a language will contain a significant number of low-frequent constructions that are not present in the corpus.

Because XHPM distributions work by moving probability mass to high-probability areas, XHPM distributions can only assign zero probability to outcomes if either (a) the prior distribution assigns zero probability to the outcome, or (b) the outcome is located in a top rest class with zero outcomes. In order to ensure robustness, DG language models should therefore be designed so that prior distributions rarely give rise to zero probabilities, and so that top rest classes are unlikely end up with zero observations. In order to avoid zero probabilities entirely, one can smooth the XHPM distribution with a non-zero prior, using a weight that is inversely proportional with the data count in the XHPM distribution.

## 6.3 Summary

In this chapter, we have described the formal machinery needed to encode probabilistic language models in DG. We have also presented a DG language model that is designed to deal with a wide range of linguistic phenomena: complement and adjunct dependencies; fillers used in control constructions, relative clauses, elliptic coordinations, and parasitic gaps; word order phenomena including long-distance movement, island constraints, and missing landed nodes; models for gapping coordinations, deep roots, and punctuation marks; and the outline of a model for identifying antecedents for anaphora.

To our knowledge, the DG language model is the first probabilistic language model for a dependency grammar with this level of linguistic sophistication. However, it is far from perfect: it does not take word distances into account when choosing a landing site for a word (a dependent should be close to both its governor and landing site); it does not consider the time-dependent probabilities needed to calculate the probabilities of partial analyses (missing complements and adjuncts are more acceptable right after the creation of the head than after a long period of time); and it implicitly assumes that each lexeme has a unique phonetic realization (in spoken language, a lexeme is never pronounced in exactly the same way twice). Moreover, the model should ideally be extended to deal with translations, multi-speaker dialogue, and even semantics in order to make it useful in natural language applications such as word alignment, machine translation, speech processing, etc.

Despite the model's shortcomings, the wide range of phenomena covered by the model makes it reasonable to hope that the DG modelling language is expressive enough to encode most of the refinements and extensions sketched above. We therefore view the modelling language and our proposed DG model as important steps towards a sophisticated probabilistic language model for a dependency formalism such as DG.

# Part IV

# Language processing

# Chapter 7

# Local optimality parsing

In this chapter, we will describe local optimality parsing in detail. In section 7.1, we generalize the algorithm for local optimality parsing so that it can take care of unsegmented input and speech signals with multiple speakers. In section 7.2, we describe how incrementality can be used to identify parsing operations that provide a good model of human parsing, and tentatively propose $k$-error parsing operations as a plausible candidate for modelling human parsing. In section 7.3, we define a set of elementary parsing operations for manipulating DG graphs, and formalize the $k$-error family of parsing operations in terms of these operations; we also describe how the parser can be implemented efficiently, provide estimates for the time complexity of the different parsing operations and parsers, and show that island constraints and other graph constraints are essential for keeping down the time complexity of local optimality parsing. In section 7.4, we propose a set of refinements to the parsing model that makes it compatible with the psycholinguistic evidence on head-driven parsing. Finally, in section 7.5, we discuss how the DG parsers can be evaluated and describe the steps we have taken in this direction.

## 7.1 Local optimality parsing with segmentation

***Summary***. *We describe how the local optimality parsing algorithm introduced in section 1.3 can be extended so that it can take care of segmentation, noise, and speech signals with multiple speakers and modalities.*

In Definition 1.23, we introduced the LOP algorithm for local optimality parsing. The algorithm requires a segmented speech signal as its input, ie, it does not model the speaker's segmentation of the speech signal. In this respect, it is not a realistic model of human parsing, and we will therefore propose an extended algorithm that includes segmentation, and which can be used for both written and spoken language.

**Definition 7.1** The algorithm for *local optimality parsing with segmentation* (*LOPS*) shown in Figure 7.1 can be used to compute a locally optimal analysis for the unsegmented speech signal $s$, given a lexicon $L$ that defines a space $\mathcal{A}$ of analyses with cost measure $c\colon \mathcal{A} \to \mathbb{R}$, and a set $\pi_1, \ldots, \pi_k\colon \mathcal{A} \to \mathrm{Pow}(\mathcal{A})$ of parsing operations. We assume $L$ specifies a possibly empty set $L_{\text{concat}}$ of dummy lexemes that are used for concatenating a node with a preceding node, and a possibly empty set $L_{\text{noise}}$ of dummy lexemes that are used for nodes representing noise (ie, segments of the speech signal that do not contain any starting point at which some lexeme in $L$ matches the speech signal). The algorithm maintains a set $B$ of segment boundaries in the speech signal, which indicate the possible starting points for new lexemes. The algorithm starts with the empty analysis and an initial $B$ containing the starting position $s^{\text{begin}}$ of the speech signal. The algorithm then parses the input incrementally by applying a parsing operation, or, if that fails, by adding a new node to the graph. To add a new node to the graph, the algorithm removes the first boundary $t_0$ from $B$, and finds the set $M$ of all lexemes in $L$ that match the speech signal from $t_0$; the ending points of the lexemes in $M$ are then added to $B$; finally, if $B$ is non-empty, the algorithm finds the first boundary $t_1$ in $B$ and adds a node for the input span $[t_0, t_1]$ with lexemes $M \cup L_{\text{concat}}$; if $B$ is empty (ie, a segment of the speech signal starting at $t_0$ cannot be analyzed as part of any lexeme), the algorithm instead finds the first position $t_1$ in the speech signal at which the speech signal either matches a lexeme in $L$ or terminates, and adds a node for the input span $[t_0, t_1]$ with noise lexemes $L_{\text{noise}}$.

The subroutine improve($a$) searches for an improvement $a'$ that can be obtained by applying a parsing operation $\pi_i$ to $a$, and is given by:

$$\text{improve}(a) = \begin{cases} \text{any locally optimal } a' \in \bigcup_{i=1}^{k} \pi_i(a) \text{ with } c(a') < c(a) \text{ if such an } a' \text{ exists;} \\ \text{`no' otherwise.} \end{cases}$$

```
procedure local optimality parsing with segmentation (LOPS)
begin
    $s$ := speech signal;
    $L$ := lexicon;
    $a$ := current analysis (initially the empty analysis);
    $t_0$ := starting time (initially $s^{\text{begin}}$);
    $B$ := set of segment boundaries (initially $\{s^{\text{begin}}\}$);
    while $t_0 < s^{\text{end}}$ or improve($a$) $\neq$ 'no' do
        if improve($a$) $\neq$ 'no' then
            $a$ := improve($a$);
        else
            $M$ := set of all lexemes in $L$ that match $s$ from $t_0$;
            $B := B \cup \{t_0 + \ell_{\text{duration}} \mid \ell \in M\} - \{t_0\}$;
            if $B$ is non-empty then
                $t_1$ := first boundary in $B$;
                $M'$ := concatenation lexemes in $L_{\text{concat}}$;
            else
                $t_1$ := first point in $s$ after $t_0$ where a lexeme starts or $s$ ends;
                $B := \{t_1\}$;
                $M'$ := noise lexemes in $L_{\text{noise}}$;
            $n$ := node spanning $[t_0, t_1]$ with lexemes $M \cup M'$;
            $a$ := partial analysis obtained by appending $n$ to $a$;
            $t_0$ := first boundary in $B$;
    return $a$;
end
```

Figure 7.1: The LOPS algorithm for local optimality parsing with segmentation and noise recovery.

**Remark 7.2** In the LOPS algorithm, the lexical matching can be based on a matching algorithm that is approximate rather than exact, such as an *n*-best algorithm that returns the closest-matching lexemes and assigns a cost to each lexeme that expresses its relative probability. In spoken language, approximate matching is a necessity because of the background noise, the natural variation between speakers, and the small random variation in pronunciation that means that a speaker never pronounces a sentence in exactly the same way twice. Even in written language, which follows standard orthography most of the time, there are occasional spelling errors that can only be recognized by means of an approximate matching algorithm.

**Remark 7.3** The LOPS algorithm allows us to add a set $L_{\text{concat}}$ of dummy concatenation lexemes to nodes that can be analyzed as corresponding to a part of a lexeme. To deal with concatenation, the lexicon must include a concatenation lexeme, like the one shown below. The lexical entry says that a concatenation lexeme should attach itself as a concatenation adjunct with edge label "+" to its immediately preceding node, adding a cost of 1000 if it fails to do so.

```
type(concat)
    -> super(lexeme)
    -> aframe(mod_concat, sub {return $_->{'+'}},
            '+' => prev(any, this))
    -> cost(missing_concat, 1000 * (prev(any, this) − [$\xrightarrow{'+'}$ this]));
```

We will assume that the parser encodes the starting time and ending time of each node in the features beginn and endn, the starting time and ending time of the current lexeme in the features beginl and endl, and the ending time of the node's concatenation in the dynamic feature endc. The lexical entry below specifies that the endc feature of a node is computed by returning the endc feature value of the "+" adjunct, if it exists, and the node's own endn feature otherwise.

```
type(node)
    -> setd(end_of_concat, endc, sub {
            my ($G, $n) = @_;
            my $concat = $G->get_node($n, [$\xleftarrow{+}$ this]);
            return defined($concat)
                ? $G->getd($concat, '/^endc$/)
                : {endc => $n->get(endn)};
        });
```

The following cost function can be used to punish any mismatch between the ending times for the node's current lexeme and the node's current concatenation (provided the node is not itself a concatenation). The cost is the same, regardless of whether the mismatch is small or large.

```
type(node)
    -> cost(bad_concatenation,
            1000 * (val(endl, this) $\neq$ val(endc, this) if $\neg$[$\xrightarrow{+}$ this]);
```

Alternatively, we can use a more complicated cost function to ensure that the cost depends on how much the ending time of the concatenation differs from the ending time of the lexeme, relative to the length of the lexeme. This will ensure that if the parser cannot create the correct concatenation in one step, it will be guided towards the correct concatenation.

```
type(node)
    -> cost(bad_concatenation, 1000 * code(sub {
        my ($G, $n) = @_;
        return 0 if ($G->get_node($n, [$\xrightarrow{+}$ this]);
        my $beginl = $G->get($n, beginl);
        my $endl = $G->get($n, endl);
        my $endc = $G->get($n, endc);
        return abs(($endl - $endc) / ($endl - $beginl));
    }));
```

**Example 7.4** As an example of how the LOPS algorithm (with the concatenation handling outlined in Remark 7.3) segments a speech signal, consider the Danish sentence "Hårfarve er arvelig." ("Hair colour is hereditary."). The table below lists all legal words contained as substrings in the sentence, excluding names for simplicity; we see that the phrase is locally ambiguous with respect to morphology.

| **H** | **å** | **r** | **f** | **ar** | **v** | **e** | **␣** | **er** | **␣** | **ar** | **v** | **e** | **l** | **i** | **g** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| hår | å | | far | ar | ve | | ␣ | er | ␣ | ar | ve | el | li | | |
| | år | | farv | arv | | | | | | arv | vel | | lig | | |
| | | | farve | arve | | | | | | arve | | | | | |
| | | | | | | | | | | arvelig | | | | | |

Since the LOPS parser only inserts boundaries after lexemes, a lexicon like the one sketched above will result in the following segmentation:

| **Hår** | **far** | **v** | **e** | **␣** | **er** | **␣** | **ar** | **v** | **e** | **l** | **i** | **g** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| hår | far | ve | | ␣ | er | ␣ | ar | ve | el | li | | |
| | farv | | | | | | arv | vel | | lig | | |
| | farve | | | | | | arve | | | | | |
| | | | | | | | arvelig | | | | | |

This segmentation of the speech signal into nodes is compatible with the following six segmentations of the speech signal into lexemes: "Hårfarve" can be segmented into either "Hår|farve" (hair-colour) or "Hår|far|ve" (hair-father-woe), and "arvelig" can be segmented into either "arvelig" (hereditary), "arve|lig" (inherit-corpse), or "ar|ve|lig" (scar-woe-corpse). Assuming that two adjacent nouns can form noun compounds in two different

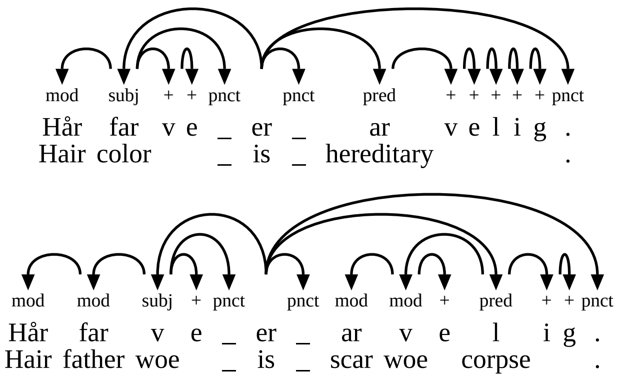

mod subj + + pnct pnct pred + + + + + pnct
Hår far v e _ er _ ar v e l i g .
Hair color _ is _ hereditary .
mod mod subj + pnct pnct mod mod + pred + + pnct
Hår far v e _ er _ ar v e l i g .
Hair father woe _ is _ scar woe corpse .

Figure 7.2: Two morphological analyses with LOPS segmentation, corresponding to "Hair-color is hereditary" (correct) and "((Hair)father)woe is ((scar)woe)corpse" (semantically ill-formed).

ways (headed either by the left noun, or the right noun), we get $2+3\cdot2=8$ different morphological analyses of "Hårfarve", and $1+2+3\cdot2=9$ different analyses of "arvelig", resulting in 72 different analyses of the sentence. Most of these analyses are syntactically well-formed but semantically ill-formed. However, since semantic ill-formedness is difficult for a machine to detect, the machine will think that the ambiguity is real.

Two of the resulting DG analyses are shown in Figure 7.2. The edge label "+" is used to denote a concatenation edge, ie, a dependency between a node and an immediately following node that is viewed as a continuation of the speech signal of the previous node, without any independent lexical interpretation. For example, in the first analysis, the three nodes "far", "v", and "e" are concatenated so that they represent the lexeme corresponding to the word "farve".

**Remark 7.5** In written language, we consider spaces to be punctuation signs which have their own associated lexemes, and which attach themselves as pnct adjuncts to the immediately preceding word. The secondary top-down step 3a in our probabilistic language model will punish words that are not followed by spaces by giving them a somewhat lower probability, which is non-zero if the treebank contains a few errors where a space is missing. This means that the LOPS parser is in principle capable of parsing a text where all spaces have been removed. The lack of spaces means that the resulting analysis has an increased cost (every missing space will be marked as ungrammatical), and that it may be more difficult for the parser to find the correct analysis because the spaces are used for disambiguation. In spoken language, prosody and pronunciation play the same role with respect to disambiguation as punctuation in written language.

**Example 7.6** The LOPS algorithm allows us to specify a set $L_{\text{noise}}$ of lexemes that are assigned to nodes whose ending point does not coincide with the end of a lexeme — ie, to nodes whose speech segment is not matched by any lexeme, and which must therefore be analyzed as noisy, inaudible, or unknown input. This error recovery mechanism allows the parser to continue its processing from the next point in the speech signal where it can recognize a lexeme, and to deal intelligently with the unanalyzed part of the speech signal. The lexeme below deals with noisy input: it simply allows the noise node to attach itself as an adjunct to the immediately preceding node, without affecting the interpretation if mod_noise is defined as a modifier that simply passes on its argument unchanged.

```
type(noise)
    -> super(lexeme)
    -> aframe(mod_noise, sub {return $_->{noise}},
            noise => prev(any, any))
    -> cost(missing_parent, 100 * (-[$\xrightarrow{\text{noise}}$ this]));
```

Dealing with unknown input is far more difficult. A rather ambitious idea is to have dummy lexemes for different combinations of word class and semantic type (event, state, object, attribute, etc). After the dummy lexemes have been assigned to the node, the parser will disambiguate between the alternatives on the basis of the syntactic, semantic, and pragmatic context. If the speaker model includes high-level cognitive processing, the speaker may be able to form a number of hypotheses about the meaning of the word. When the parser's disambiguation process has stabilized, the speaker can insert the lexemes corresponding to the most plausible hypotheses into the speaker's lexicon. The next time the unknown word is used, these lexemes must then compete with each other. Lexemes that often win in this competetion will receive a high probability, whereas lexemes that never win will receive a low probability and eventually be weeded out by the speaker's learning algorithm. Inaudible input can be handled in the same way, except that the speaker uses an existing lexeme rather than hypothesizing a set of new lexemes. Obviously, this proposal requires a model of the speaker's cognitive abilities that is far beyond the current state-of-art within cognitive science. It should therefore be viewed as a rather incomplete and tentative account of how humans deal with unknown words.

In the LOPS algorithm, the input consists of a single speech signal. Thus, there is no way to distinguish between different speakers or different modalities (ie, speech, vision, etc.), and the LOPS algorithm is therefore not well suited to parsing dialogue with several speakers and overlapping speech. For this reason, we will propose the following extension of LOPS where the speech signal consists of $k$ different channels that are segmented independently of each other. The channels can be used to represent different speakers and different modalities.

**Definition 7.7** Let $\mathcal{A}$ be a set of analyses, let $c\colon \mathcal{A} \to \mathbb{R}$ be a cost measure on $\mathcal{A}$, let $s$ be a speech signal with $k$ channels $s_1, \ldots, s_k$, let $L$ be a lexicon with sublexicon $L_i$ for speech channel $i$, and let $\pi_1, \ldots, \pi_k\colon \mathcal{A} \to \mathrm{Pow}(\mathcal{A})$ be a set of parsing operations. The algorithm for *multi-channel local optimality parsing with segmentation* (*MLOPS*) shown in Figure 7.3 can be used to compute a locally optimal analysis for the unsegmented multi-channel speech signal $s$. The set $L_i^{\text{concat}}$ denotes a possibly empty set of dummy lexemes for channel $i$ that are used for concatenating a node with a preceding node, and the set $L_i^{\text{noise}}$ denotes a possibly empty set of dummy lexemes for channel $i$ that are used for nodes representing noise (ie, a part of the speech signal that does not match any lexeme in $L_i$). The procedure improve($a$) is defined as in Definition 7.1.

**Remark 7.8** The LOPS and MLOPS parsers are *serial parsers with repair*, because they only maintain one analysis at a time and allow destructive changes to the current analysis; they are also *incremental* because they always produce locally optimal analyses before adding the next segment to the graph (cf. Definition 1.24).

**Remark 7.9** The LOP parser resembles the parser proposed by Daum and Menzel (2002) in that it is based on local search with a cost-based grammar. The most important difference between the two parsers is that the parser proposed by Daum and Menzel is based on guided local search with 1-change operations rather than simple local search with $k$-error operations,

```
procedure multi-channel local optimality parsing (MLOPS)
begin
    $s$ := speech signal with channels $s_1, \ldots, s_k$;
    $L_i$ := lexicon for channel $i$;
    $a$ := current analysis (initially the empty analysis);
    $t_0$ := starting time (initially $s^{\text{begin}}$);
    $B_i$ := set of segment boundaries for channel $i$ (initially $\{s^{\text{begin}}\}$);
    while $t_0 < s^{\text{end}}$ or improve($a$) $\neq$ 'no' do
        if improve($a$) $\neq$ 'no' then
            $a$ := improve($a$);
        else
            $i$ := any channel where $B_i$ contains first boundary $t_0$;
            $M_i$ := set of all lexemes in $L_i$ that match $s_i$ from $t_0$;
            $B_i := B_i \cup \{t_0 + \ell_{\text{duration}} \mid \ell \in M_i\} - \{t_0\}$;
            if $B_i$ is non-empty then
                $t_1$ := first boundary in $B_i$;
                $M_i'$ := concatenation lexemes in $L_i^{\text{concat}}$;
            else
                $t_1$ := first point in $s_i$ after $t_0$ where a lexeme starts or $s_i$ ends;
                $B_i := \{t_1\}$;
                $M_i'$ := noise lexemes in $L_i^{\text{noise}}$;
            $n$ := node spanning $[t_0, t_1]$ in $s_i$ with lexemes $M_i \cup M_i'$;
            $a$ := partial analysis obtained by appending $n$ to $a$;
            $t_0$ := first boundary in $B_1 \cup \ldots \cup B_k$;
    return $a$;
end
```

Figure 7.3: The MLOPS algorithm for multi-channel local optimality parsing with segmentation and noise recovery.

that their parsing algorithm is parallel and non-incremental and does not start with the empty analysis, and that their dependency analyses are not allowed to be discontinuous.

In this section, we have proposed two extensions to our algorithm for local optimality parsing (LOP): local optimality parsing with segmentation (LOPS), and multi-channel local optimality parsing with segmentation (MLOPS). These extensions enable our parsing algorithm to deal with segmentation of the speech input, so that it can handle unsegmented written and spoken language, as well as dialogues with multiple speakers and modalities. However, we have not yet specified the set of parsing operations used by these algorithms — an issue we will address in the following two sections.

## 7.2 Identifying the parsing operations used by humans

*Summary. We argue that the choice of parsing operations in human parsing is determined by the need to limit the time complexity as much as possible without significantly reducing the parser's ability to find the global optimum. We show that local ambiguities provide important information about the parsing operations used in human parsing, and propose different families of parsing operations (k-connect, k-displace, reattach, and k-error) that are almost as powerful as k-change, but require far less computation.*

In sections 1.3 and 1.4, we introduced the $k$-change family of parsing operations. Because the edit distance between alternative analyses in a local ambiguity rarely exceeds 3 in natural language, $k$-change with $k \leq 3$ is powerful enough to resolve most ambiguities during local optimality parsing. However, while $k$-change is a very powerful family of parsing operations, it is not particularly efficient, as described below.

**Remark 7.10** Cost functions in DG are local, ie, the computation of the cost at a particular node in the graph only involves a small set of nodes that are not too far apart. Consequently, when trying to eliminate a high cost at a particular node by performing a $k$-change operation, it is wasteful to try $k$-change operations where some of the involved nodes are located far from the node we want to improve. From a computational point of view, we therefore expect humans to use local $k$-change operations and avoid the non-local ones. This is indeed what the psycholinguistic literature seems to suggest: in a famous eyetracking experiment with garden path sentences, Frazier and Rayner (1982) demonstrated that humans pursue a preferred analysis, and when the initial analysis proves wrong, they quickly focus on the problematic region in the sentence, allowing them to revise just that part of the initial analysis which led to the conflict. This observation is consistent with local $k$-change operations, whereas we would expect focus on other regions too if non-local $k$-change operations were involved.

If we accept the idea that human parsing is based on local $k$-change operations, we must ask ourselves whether these parsing operations are innate or learned, and how we can identify the *k*-change operations that are used by a particular speaker.

**Remark 7.11** Just as different speakers of the same language have slightly different grammars, it may well be the case that parsing operations are learned, and that different speakers acquire slightly different parsing operations (obviously, the parsing operations they acquire can be expected to be near-identical because they are trying to learn the same language). The hypothesis can be tested by looking at strong garden-path effects: if speakers can learn to resolve strong garden-paths without conscious effort, or if speakers of one language can resolve garden-paths that are hard to resolve for speakers of another language, then this can be interpreted as evidence that parsing operations are learned rather than hard-wired.

If parsing operations are learned, how does this learning take place? There are many possible explanations. One possibility is that the parsing algorithm and the *k*-change operations are innate, but that children gradually learn to refine the *k*-change operations by observing the patterns in the *k*-change operations that lead to successful reanalyses. In this way, they can formulate the specializations of *k*-change that they need to perform efficient parsing, and use the original *k*-change operation as a fallback operation when everything else fails. This strategy would explain why humans can analyze even strong garden-paths, but with a large delay.

Although there is no direct way of determining which *k*-change operations are available to a given speaker, we can determine the operations indirectly by looking at how the speaker parses different utterances. We will assume that parsing is *incremental*, ie, that the speaker produces an optimal analysis before adding the next speech segment to the graph. If the speaker can parse an utterance without problems, then the speaker's parsing operations are powerful enough to produce the correct analysis via a sequence of locally optimal analyses; but if there is a strong garden-path effect, then the speaker's parsing operations are too weak to get from the analysis that was locally optimal before the disambiguating region appeared, to the analysis that was globally optimal after the disambiguating region appeared. Since we can ask speakers about their favoured (=optimal) analysis of the first *k* words of the input before we reveal the remaining words to them, we can learn a lot about human parsing operations by looking at how speakers analyze partial examples.

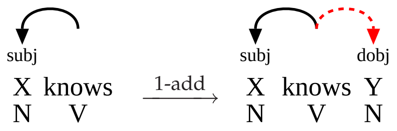


Figure 7.4: Example of 1-add. The optimal analysis of "X knows Y" before and after "Y" is added to the analysis.

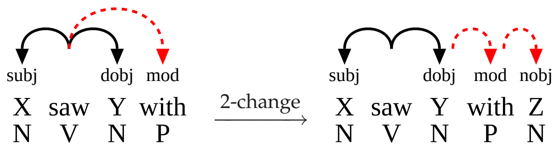


Figure 7.5: Example of 2-change repair. The optimal analysis of "X saw Y with Z" immediately before and after "Z" is added to the graph.

**Example 7.12** Figure 7.4 shows the optimal analysis of "X knows Y" before and after "Y" is added to the analysis. Since the sentence is not a garden-path, we conclude that English speakers must be capable of non-destructively adding dependency edges between nodes, ie, they must possess a non-destructive 1-change operation, which we will call 1-*add*.

**Example 7.13** While 1-add is often enough to incorporate a new word correctly into the analysis, destructive repairs are sometimes needed. Figure 7.5 shows the optimal analysis of "X saw Y with Z" immediately before and after "Z" is added to the analysis. We assume that the speaker favours the VP-attachment by default, but prefers the NP-attachment after actually seeing "Z". The two analyses are related by a destructive 2-change operation. Since the sentence is not a strong garden-path, we conclude that English speakers must be capable of descructively changing a single dependency edge after adding another dependency edge by means of some subfamily of 2-change repairs.

**Example 7.14** Figure 7.6 shows the optimal analysis of "X knows Y is Z" immediately before and after "is" is added to the graph. Since the sentence is not a strong garden-path, we again conclude that English speakers must be capable of destructively changing a single dependency edge after adding another by means of some subfamily of 2-change repairs.

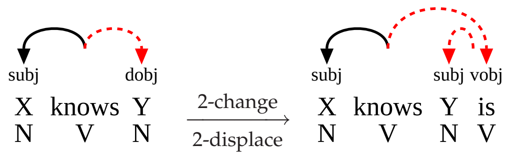


Figure 7.6: Example of 2-change repair. The optimal analysis of "X knows Y is Z" immediately before and after "is" is added to the analysis.

**Remark 7.15** Sentences like "X knows Y is Z" that do not exhibit strong garden-path effects, but do involve some kind of reanalysis, often result in a slight delay (called a *weak garden-path effect*) that can be measured in eye-tracking experiments (cf. Ferreira and Henderson 1990). Weak garden-path effects are not surprising if we assume that the human parser tries fast parsing operations (eg, 1-add) before it tries slower ones (eg, 2-change). In our example, this means that the parser must perform a 1-add operation and a slower 2-change operation, which takes more time than performing a 1-add operation alone, thereby explaining the slight delay.

**Example 7.16** Examples of 3-change repairs in naturally occurring language are harder to find, but they do exist. Figure 7.7 shows the German example "Die Walzer haben die Musiker erfreut" ("The waltzes have pleased the musicians"), where the initial analysis of "Die Walzer haben die Musiker" ("The musicians possess the waltzes") is optimal before the lookup, and the revised analysis is optimal after the lookup. The repair reverses the subject and direct object roles, and reanalyzes "haben" as having a verbal object rather than a direct object.

Examples exhibiting strong garden-path effects are as interesting as normal examples because they provide negative information about the parsing operations employed in human parsing.

**Example 7.17** From the observation that the strong garden-path in Figure 7.8 can be repaired with a 2-change operation, we can conclude that the full power of 2-change is not readily available in human parsing.

**Example 7.18** The headline "Dismay With Saudi Arabia Fuels Pullout Talks" (from New York Times, January 16, 2002) is an example of a strong garden-path sentence, which is highly ambiguous because many of the words both

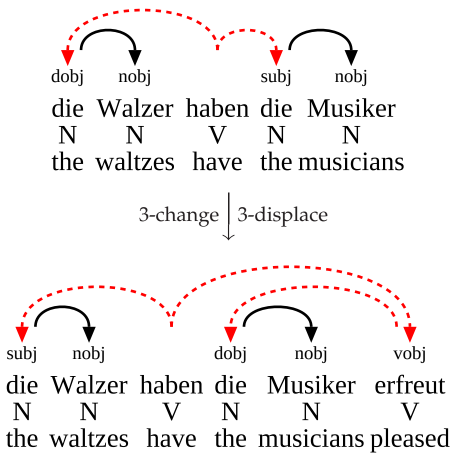

dobj
nobj
subj
nobj
die Walzer haben die Musiker
N N V N N
the waltzes have the musicians
3-change
3-displace
subj
nobj
dobj
nobj
vobj
die Walzer haben die Musiker erfreut
N N V N N V
the waltzes have the musicians pleased

Figure 7.7: Example of 3-change repair. The optimal analysis of "Die Walzer haben die Musiker erfreut" immediately before and after "erfreut" is added to the analysis.

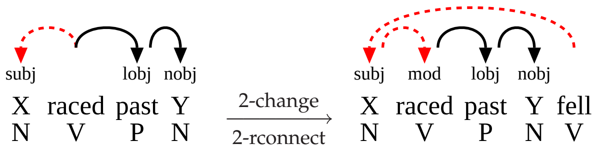
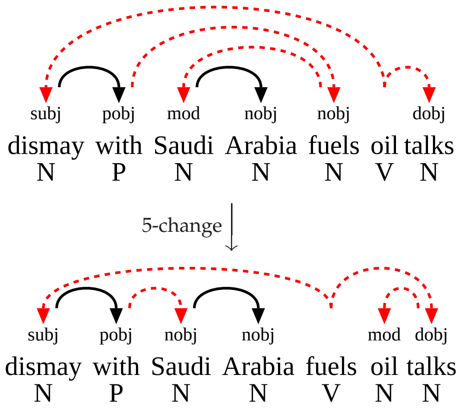


Figure 7.8: Example of garden-path with edit distance 2. The optimal analysis of "The horse raced past the barn fell" immediately before and after "fell" is added to the analysis.

have a noun reading and a verb reading. As shown in Figure 7.9, the edit distance between two ambiguous readings can be as high as 5, ie, we need a 5-change operation to change the first analysis into the other. Because of the high edit distance, readers who misinterpret one of the words — eg, by analyzing "Fuels" as a noun that is part of the noun compound "Saudia Arabia Fuels" — have difficulties in finding a parsing operation that changes their initial incorrect analysis to the intended analysis, and therefore experience a strong garden-path effect.

The positive and negative examples presented above can be used to identify families of parsing operations that are powerful enough to handle



subj
pobj
mod
nobj
nobj
dobj
dismay with Saudi Arabia fuels oil talks
N P N N N V N
5-change
subj
pobj
nobj
nobj
mod
dobj
dismay with Saudi Arabia fuels oil talks
N P N N V N N

Figure 7.9: Two alternative analyses of an ambiguous sentence, related by a 5-change operation.

the positive examples, but weak enough to fail to handle the negative examples. First of all, we observe that in all of the positive examples we have presented so far, the changed edges have formed a connected subgraph of the original dependency graph. Even in the strong garden-path examples, the involved edges are not arbitrarily far apart. Since $k$-change does not prevent the involved edges from being arbitrarily far apart, this suggests that we should look at the following family of parsing operations.

**Definition 7.19** A *k-connect operation* is a $k$-change operation which reanalyses $k$ nodes $n_1, \ldots, n_k$ where both the changed edges and the new edges at the nodes $n_1, \ldots, n_i$ form a connected subgraph for each $i = 1, \ldots, k$. A *rooted k-connect operation* (or *k-rconnect operation*) is a $k$-connect operation where $n_1$ is a root. An *error-initiated k-connect operation* (or *k-econnect operation*) is a $k$-connect operation where $n_1$ is a dynamic error node.

We can formulate an even tighter subfamily of $k$-connect by observing that the changed edges in the positive examples of repairs presented here do not only form a connected graph: the repair often seems to be triggered by an unanalyzed word $w_1$ whose new dependency role is blocked by the dependency role of a second word $w_2$, whose new dependency role may in turn be blocked by a third word $w_3$, and so on (eg, the direct object "Y" blocks the analysis of "is" as a verbal object in Figure 7.6). This suggests we should look at the following family of parsing operations.

**Definition 7.20** A *k-displace operation* (*k-edisplace operation*) is a $k$-rconnect ($k$-econnect) operation which reanalyzes $k$ nodes $n_1, \ldots, n_k$ where the new dependency role of $n_i$ is blocked by the old dependency role of $n_{i+1}$ for each $i = 1, \ldots, k-1$.

**Remark 7.21** Note that 1-connect coincides with 1-change, and that 1-rconnect and 1-displace coincide with 1-add.

**Example 7.22** The repair in Figure 7.5 is both a 2-connect and a 2-rconnect operation since $Z$ is initially a root, but not a 2-displace operation because the VP attachment of "with" does not block the attachment of $Z$ as a nominal object. Alternatively, we can analyze Figure 7.5 as the result of first applying a cost-improving 1-add operation followed by a cost-improving 1-change operation.

**Example 7.23** Figure 7.6 is a 2-displace repair where the root "is" is reanalyzed as a verbal object of "knows" that displaces the old direct object "Y", and where "Y" is subsequently reanalyzed as the subject of "is".

**Example 7.24** Figure 7.7 is a 3-displace repair where the root "erfreut" ("pleased") is reanalyzed as a verbal object of "haben" ("have") that displaces the old direct object "die Walzer" ("the waltzes"); "die Walzer" is then reanalyzed as a subject of "haben" that displaces the old subject "die Musiker" ("the musicians"); and finally, "die Musiker" is reanalyzed as the direct object of "erfreut".

**Example 7.25** The operation needed to repair the strong garden-path in Figure 7.8 is a 2-rconnect operation where the root "raced" is reanalyzed as a modifier of "X", and where "X" is reanalyzed as a subject of "fell". However, it is not a $k$-displace operation because the reanalysis of "raced" does not displace "fell". Since the sentence is a strong garden-path, it shows that the full power of 2-rconnect is not available in human parsing, but the garden-path does not disprove 2-displace.

From the preceding examples, we see that $k$-connect is in general more powerful than the repairs actually observed in human parsing, and that $k$-displace with $k \leq 3$ can account for all of our positive examples of repairs, except for the PP-reattachment example in Figure 7.5 which can be handled by a 2-rconnect operation, or by a 1-add operation followed by a 1-change. Thus, we might hypothesize that human parsing can be modelled by 1-change and $k$-displace with $k \leq 3$, as proposed by Kromann (2001). However, while 3-displace captures many of the structural regularities in parsing operations, there are alternative explanations. In our implementation of costs, we have used localized cost functions that assign a cost to each node in the dependency graph. Within such a framework, the error-driven $k$-error parsing operation defined informally below is a sensible alternative to $k$-displace. The underlying intuition is that in each parsing step, we should focus on one error and try to repair only the chosen error and any new errors created during the repair.

**Definition 7.26** If a cost function $c$ at a node $n$ generates a static cost that exceeds the cost function's predefined threshold, then $n$ is called a *static error node signalled by* $c$, and the cost produced by $c$ is called an *error cost*. The node $n$ is called a *dynamic error node* if the dynamic cost of $c$ exceeds the threshold as well (cf. p. 102).

**Definition 7.27** We will assume that cost functions that signal an error always specify a *blame set* of nodes and edges that are responsible for the error. In general, we will assume that the operator $\neg a$ at a node $n$ returns the blame set consisting of $n$ alone, that any other numerical cost operator $c$ returns the union of $n$ and all the blame sets associated with the argument cost functions of $c$, and that any non-numerical cost operator $c$ returns the union of $n$ and all nodes and edges matched by $c$.

**Example 7.28** An agreement error between a subject and a verb should be blamed on the subject, the verb, and the subject edge. Similarly, an island violation should be blamed on the island and on the extracted node and its surface edge.

**Definition 7.29** Let $n$ be a node in a graph $G$. A parsing operation $\pi$ is said to *affect* $n$ if $\pi$ changes the chosen lexeme at $n$, adds or deletes some edge at $n$, or causes some cost function to blame $n$ for an error $e$ introduced by $\pi$ (ie, the error must be present in $\pi(G)$, but not in $G$).

**Definition 7.30** A *$k$-error operation* is a $k$-change operation which reanalyses $k$ nodes $n_1, \ldots, n_k$ in a graph $G$ such that the following three conditions are satisfied, where $G_0 = G$ and $G_i$ denotes the graph $G$ after changing $n_1, \dots, n_i$: (a) $n_1$ is blamed by a cost function $c$ that signals a dynamic error in $G$, but no error in $G_1$; (b) each $n_i$ with $i > 1$ is affected by the change from $G_{i-1}$ to $G_i$; and (c) each $n_i$ is blamed by a cost function $c$ that signals a static error in $G_{i-1}$, but no error in $G_i$.

**Remark 7.31** $k$-error coincides with 1-change on blame nodes when $k = 1$. Moreover, if all cost functions are local — ie, a cost function only depends on its associated node and its local neighbourhood in the graph (a property that does not hold for the extraction operator) — then a $k$-error operation is also a $k$-connect operation since a local cost function cannot change without some change in the local neighbourhood.

**Example 7.32** Assuming that roots are always error nodes, Figure 7.4 is an example of 1-error, and Figure 7.5 is an example of 2-error if we assume that the VP-attachment analysis turns "with" into an error node, ie, that "with" detects that it cannot sensibly modify the verb because of semantic mismatch when it has "Z" as its nominal object. Figure 7.6 is an example of 2-error, and Figure 7.7 is an example of 3-error. The strong garden-path in Figure 7.8 is not an example of $k$-error because the repair can only start by reanalyzing "X", and "X" is not an error node (the parsing operations do not allow the root "raced" to be reanalyzed as a dependent of "X" before "X" has been deleted as a dependent of "raced" — ie, the parsing operations can never result in cyclic dependencies).

From the preceding examples, we see that $k$-error with $k \leq 3$ explains all our positive and negative examples of repairs. Moreover, 1-error is far more restrictive than 1-change because 1-error only applies to error nodes whereas 1-change applies to all nodes. This means that $k$-error is a more efficient family than 1-change and $k$-displace. For that reason, we believe that $k$-error is an even better model of human parsing than 1-change and $k$-displace, and we will therefore use $k$-error with $k \leq 3$ as the basis of our DTAG implementation of local optimality parsing.

In order to ensure that the parser arrives at a locally optimal analysis within $O(n)$ parsing operations, where $n$ is the number of segments in the input, we must place special restrictions on the parsing operations.

**Definition 7.33** A parsing operation is called *edge-monotonic* if it never decreases the number of edges in the graph, and *edge-increasing* if it always increases the number of edges in the graph.

**Remark 7.34** Since a dependency graph always has at most $O(n)$ edges, the parser is guaranteed to arrive at a locally optimal analysis within $O(n)$ parsing operations when all parsing operations are edge-increasing. If the parsing operations are only edge-monotonic, the parser is not guaranteed to terminate within $O(n)$ operations, unless we assume that there exists a constant $k$ such that we cannot apply more than $k$ locally optimal cost-increasing and edge-monotonic parsing operations after each other without increasing the number of edges. In practice, this assumption is not unrealistic, and we will therefore assume that if the parsing operations are edge-monotonic, then the parser will terminate within $O(n)$ operations.

**Remark 7.35** Other authors have proposed parsing operations that resemble the parsing operations proposed here. For example, 1-change is equivalent to the "snip" operation proposed by Lewis (1999), and 1-change operations are also used in the parsing algorithm by Daum and Menzel (2002). Similarly, the Grammatical Dependency Principle proposed by Fodor and Inoue in (Fodor and Ferreira 1998, ch. 4) can be viewed as a special instance of $k$-error operations.

In this section, we have presented different families of parsing operations ($k$-connect, $k$-displace, $k$-error) that seem powerful enough to deal with most of the cases that can be handled by humans, while being more computationally efficient than $k$-change. We have made only modest use of the psycholinguistic evidence, and much more research is clearly needed to determine which parsing operations are actually used in human parsing. However, we have provided a methodology that can be used to determine this set of parsing operations by psycholinguistic observations of how humans analyze partial input.

## 7.3 An efficient set of parsing operations

***Summary.*** *We describe our elementary parsing operations for changing a node's lexeme, landing site, and governor, and how these operations are chained together in our implementation of k-error operations. We describe the time complexity of the operations in four scenarios with different assumptions about graph structure, island constraints, and discontinuity. Finally, we discuss how k-error operations can be implemented efficiently by means of first-improvement search algorithms with anytime properties.*

So far, we have only focused on the changes that parsing operations make to the dependency structure, ie, we have ignored any changes to a node's lexeme and landing site. We will now adress this problem. We start by specifying the elementary operations for modifying a node $n$ in a dependency graph $G$ with respect to its active lexeme, landing site, governor, fillers, and gap dependents.

**Definition 7.36** A *lex-chn* operation at $n$ is a 1-change operation that replaces the active lexeme at $n$ with one of $n$'s non-active lexemes.

**Definition 7.37** An *lsite-del* operation at an ungoverned non-filler node $n$ with landing edge $l \xrightarrow{e} n$ is a 1-change operation that deletes $l \xrightarrow{e} n$.

**Definition 7.38** A *gov-del* operation at a node $n$ with governor edge $g \xrightarrow{e} n$ is a 1-change operation that deletes $g \xrightarrow{e} n$ and any incoming replacement edge at $n$.

**Definition 7.39** A *fill-del* operation at a filler node $f$ is an operation that deletes the node $f$ and all fillers licensed by it from the graph along with all edges connected to the deleted nodes.

**Definition 7.40** A *src-del* operation at a filler node $f$ with filler source edge $s \xrightarrow{e} f$ is a 1-change operation that deletes $s \xrightarrow{e} f$.

When adding edges to the graph, we distinguish between edges that are licensed by an active lexeme ("add" operations), and edges that are only licensed by a non-active lexeme ("addx" operations). Since the edge produced by an addx-operation is not licensed by the active lexeme, addx-operations must usually be followed by a lex-chn operation that changes the active lexeme to the lexeme that licensed the edge.

**Definition 7.41** An *lsite-add* (*lsite-addx*) operation at a node $n$ without a landing site is a 1-change operation that adds a new surface edge of the form $l \xrightarrow{e} n$, where the node $l$ is yield-adjacent to $n$ and has an active (non-active) lexeme that licenses $n$ as a landed node with edge label $e$.

**Definition 7.42** A *comp-add* (*comp-addx*) operation at a node $n$ without a governor edge, but with landing site $l$ where $l$ is a non-filler, is a 1-change operation that adds a new governor edge of the form $g \xrightarrow{e_g} n$ where the node $g$ equals $l$ or is dominated by $l$ in the deep tree or the surface tree, and has an active (non-active) lexeme whose complement frame licenses $n$ as a complement with complement edge $g \xrightarrow{e_g} n$.

**Definition 7.43** An *adj-add* (*adj-addx*) operation at a node $n$ without a governor edge, but with landing site $l$ where $l$ is a non-filler, is a 1-change operation that adds a new governor edge of the form $g \xrightarrow{e_g} n$, where $g$ is a node which is dominated by $l$ in the deep tree or the surface tree such that an active (non-active) lexeme at $n$ has an adjunct frame that licenses the adjunct governor edge $g \xrightarrow{e_g} n$.

**Definition 7.44** A *fill-add* (*fill-addx*) operation at a node $n$ is an operation that adds a non-gapping filler node $f$ and a filler edge $l \xrightarrow{e} f$ to the graph, where $f$ and $l \xrightarrow{e} f$ must be licensed by a filler frame associated with an active (non-active) lexeme at a node $l$ that dominates $n$ in the deep tree or the surface tree.

**Definition 7.45** A *fill-chn* (*fill-chnx*) operation at a filler node $f$ with filler edge $l \xrightarrow{e_s} f$ and governor $g \xrightarrow{e_d} f$ is an operation that deletes the edge $l \xrightarrow{e_s} f$, and adds the edge $l' \xrightarrow{e_s'} f$ where $l'$ is a node that dominates $g$ and has an active (non-active) lexeme with a filler frame that licenses $l' \xrightarrow{e_s'} f$.

**Definition 7.46** A *src-add* operation at a filler node $f$ with filler edge $l \xrightarrow{e_f} f$, but without a filler source edge, is a 1-change operation that adds a new filler source edge $s \xrightarrow{e_s} f$ which is licensed by the filler frame in $l$ that generated $f$.

**Definition 7.47** A *gap-add* (*gap-addx*) operation at a node $n$ without a landing site is an operation that adds a gapping filler $f$, a filler edge $l \xrightarrow{e_f} f$, and a landing edge $f \xrightarrow{e_l} n$, where $l$ is yield-adjacent to $n$ with an active (non-active) lexeme that has a gapping frame that licenses the created gapping filler $f$ and the created filler and landing edge.

**Definition 7.48** A *glsite-add* operation at a node $n$ without a landing site is an operation that creates a landing edge $f \xrightarrow{e_l} n$ where $f$ is a gapping filler with some gapping dependent that is yield-adjacent to $n$, and where $e_l$ is the landing edge type licensed by the gapping frame for $f$.

**Definition 7.49** Let $G$ be a dependency graph containing a governor $g$. A dependency edge $g \xrightarrow{e} d$ in $G$ is said to *block* another edge $g \xrightarrow{e'} d'$ provided $g$ has a cost function $c$ that produces an error cost if the edge $g \xrightarrow{e'} d'$ is added to $G$, but does not produce an error cost if $G$ is left unchanged or if $g \xrightarrow{e'} d'$ replaces $g \xrightarrow{e} d$ in $G$.

**Definition 7.50** A *gcomp-add* (*gcomp-addx*) operation at a node $n$ without a governor edge, but with landing edge $l \xrightarrow{e_l} n$ where $l$ is a gapping filler, is an operation that (a) identifies a proper subsource $g'$ in the source tree for $l$ which has an active (non-active) complement frame that licenses some complement edge $g' \xrightarrow{e_g} n$; (b) identifies a dependent edge $g' \xrightarrow{e_g'} n'$ in the source tree such that $g' \xrightarrow{e_g'} n'$ blocks $g' \xrightarrow{e_g} n$; (c) creates an internal filler $g$ for $g'$ in the gapping tree, if absent; and (d) adds a new complement edge $g \xrightarrow{e_g} n$ and a new replacement edge $n' \xrightarrow{e_r} n$ where $e_r$ is the replacement edge type licensed by $l$.

**Definition 7.51** A *gadj-add* (*gadj-addx*) operation at a node $n$ without a governor edge, but with landing edge $l \xrightarrow{e_l} n$ where $l$ is a gapping filler, is an operation that (a) identifies a node $g$ in the existing gapping tree for $l$ such that $n$ has an active (non-active) adjunct frame that licenses some adjunct edge $g \xrightarrow{e_g} n$; (b) identifies a dependent edge $g' \xrightarrow{e_g'} n'$ in the source tree such that $g' \xrightarrow{e_g'} n'$ blocks $g' \xrightarrow{e_g} n$, where $g'$ is the subsource of $g$; and (c) adds a new adjunct edge $g \xrightarrow{e_g} n$ and a new replacement edge $n' \xrightarrow{e_r} n$ where $e_r$ is the replacement edge type licensed by $l$.

**Definition 7.52** A *gap-par* operation at a gapping filler $n$ is an operation that ensures parallelism between the source tree and the gapping tree for $n$ by adding and deleting internal fillers and boundary fillers as necessary.

We have now defined all the elementary parsing operations that we need. Figure 7.10 categorizes the parsing operations according to whether they apply to ordinary nodes, fillers, or gaps. It is quite conceivable that the human brain may be capable of learning these parsing operations from an even simpler set of operations, but in the absence of a theory that explains how this learning mechanism works, it is convenient to assume that the elementary parsing operations are innate.

**Remark 7.53** Our set of parsing operations is complete, ie, given any two well-formed dependency analyses of the same speech signal, it is possible

| | **delete** | **add** | **change** |
|---|---|---|---|
| **ordinary node** | | | |
| lexeme | * | * | lex-chn |
| landing site | lsite-del | lsite-add/addx | * |
| governor (comp) | gov-del | comp-add/addx | * |
| governor (adj) | gov-del | adj-add/addx | * |
| **filler** | | | |
| filler | fill-del | fill-add/addx | fill-chn/chnx |
| source | src-del | src-add | * |
| governor (comp) | gov-del | comp-add/addx | * |
| governor (adj) | gov-del | adj-add/addx | * |
| **gap** | | | |
| filler | fill-del | gap-add/addx | fill-chn/chnx |
| source | src-del | src-add | * |
| gap governor (comp) | gov-del | comp-add/addx | * |
| gap governor (adj) | gov-del | adj-add/addx | * |
| landed node lsite | lsite-del | glsite-add | * |
| landed node gov (comp) | gov-del | gcomp-add/addx | * |
| landed node gov (adj) | gov-del | gadj-add/addx | * |
| nodes in gapping tree | gap-par | gap-par | gap-par |

Figure 7.10: The elementary parsing operations that apply to different kinds of nodes.

to find a sequence of parsing operations that transforms one analysis into the other. In particular, any dependency analysis licensed by the grammar can be constructed by means of the parsing operations from the empty analysis. To see this, note that we can use lsite-del, gov-del, fill-del, src-del and gap-par to delete all edges and filler nodes from the old analysis, use lex-chn to set the new active lexeme at each node, and use the various add operations and gap-par to create the edges and filler nodes in the new analysis. Obviously, this is a very inefficient way of transforming the old analysis into the new one, and we can shorten the transformation significantly by changing only those lexemes, filler nodes, and edges that are not shared by both analyses.

We will now look at the time complexities of the elementary parsing operations. Since the time complexity depends on the structure of the graphs under consideration, we will consider four scenarios with different linguistically motivated restrictions on the graph structures with respect to their complements and adjuncts, potential landing sites, island constraints, and continuity (cf. Kromann 2001). As we will see, these restrictions significantly decrease the time complexity of the parsing operations.

**Definition 7.54** The time complexity of the elementary parsing operations will be evaluated in the following four scenarios, which place different restrictions on the growth of various graph properties as a function of the number of nodes in the graph:

> **S1 (none)**: no restrictions at all.
>
> **S2 (branching)**: any given node in the graph must have bounded degree (ie, it must be connected to at most $O(1)$ edges) and have at most $O(\log n)$ parents in the deep tree and in the surface tree (ie, the deep tree and the surface tree must have height $O(\log n)$).
>
> **S3 (islands)**: in addition to satisfying S2, island constraints must ensure that there is at most one channel for extraction from unbounded depth in each node, ie, that a node has at most $O(\log n)$ possible governors when its landing site is given.
>
> **S4 (continuity)**: in addition to satisfying S2, the graph must be continuous, ie, a node's governor must always coincide with its landing site.

**Remark 7.55** The assumptions in scenarios S2–S4 capture important regularities in human language. For example, we almost never observe words with more than 5 complements or 5 adjuncts (fewer than 0.026% of all words in the Danish Dependency Treebank have more than 10 dependents), and the assumption about bounded degree is therefore reasonable from a linguistic point of view.

The landing site assumption in S2 captures the observation that the number of potential landing sites for a word tends to grow significantly slower than the number of nodes in the graph, ie, a word tends to be yield-adjacent to only very few other words in the graph. We have assumed that the number of yield-adjacent words is at most $O(\log n)$, but because the Danish Dependency Treebank lacks dependencies at the discourse level, the treebank cannot be used to check the empirical validity of this bound.[1]

The island constraint assumption in scenario S3 is essentially a consequence of the island constraints proposed by Ross (1967), since these island constraints have the effect that there is essentially only one channel for extraction from unbounded depth.[2] The adjunct island constraint means that extraction from unbounded depth can only take place via complement edges, and the complex NP constraint means that the extraction path cannot contain nouns dominating verbs; even extraction from nouns with nested prepositional complements seems to be highly restricted. That is, we claim that the purpose of island constraints is to ensure that a node with a given landing site cannot have more than $O(d)$ potential governors that do not violate an island constraint, where $d$ is the depth of the graph. Since $d$ is bounded by $O(\log n)$ by the landing site assumption in scenario S2, it follows that the number of potential governors is bounded by $O(\log n)$. It would be interesting to verify the island constraint assumption empirically on treebank data, but this is outside the scope of this dissertation.

Discontinuous constructions are found in virtually all human languages, so the continuity restriction in scenario S4 is obviously too strong to be linguistically plausible. For example, approximately 0.93% of all dependencies in the Danish Dependency Treebank are discontinuous. But S4 can perhaps be viewed as a linguistically defensible approximation to natural languages, and because most dependencies are continuous, the time complexity for scenario S4 may provide a better estimate for the average-case time complexity of local optimality parsing than scenario S3.

[1]If the landing site assumption in S2 is valid, we cannot have linearly nested PPs of arbitrary length (eg, “the man in the chair in the room in the house in the forest. . . ”). It is not clear that humans are capable of understanding arbitrarily long nested PPs, but if they are, the landing site assumption is too strong. However, while the landing site assumption is probably important for keeping down the worst-case time complexity of the parser, it may be possible to loosen it without affecting the average-case time complexity by assuming that the parser uses a best-first search strategy to find optimal parsing operations. For example, if an error-driven parser tries the lowest landing site first and it results in an error-free analysis, there is no reason for it to proceed to a higher landing site. With this strategy, the number of potential landing sites is less important. A constraint that penalizes having a large word distance between a word and its governor and landing site is probably also strong enough to force the average-case time complexity down to the estimate we have presented here.

[2]The observation that there is usually only one “spine” of extraction in human languages has been proposed independently by Christian Wartena, in a talk given in the Mathematics of Language workshop at ESSLLI 2001 (cf. Michaelis and Wartena 1997, 1998).

**Remark 7.56** Figure 7.11 shows the maximal number of graphs that can be produced by a family of parsing operations at a given node, and the resulting time complexities in scenarios S1–S4.

In our estimates, we assume the following upper bounds. We assume that a lexeme's complement frames specify at most $\boldsymbol{c}$ different complement roles, that its adjunct frames specify at most $\boldsymbol{a}$ different adjunct roles, that its landing frames specify at most $\boldsymbol{s}$ different surface roles, that it has at most $\boldsymbol{f}$ filler frames, and that a filler frame matches at most $\boldsymbol{o}$ different filler sources; since all these properties are independent of the graph structure, they are $O(1)$ in all scenarios. We assume that a node can have at most $\boldsymbol{x}$ non-active lexemes; since this property does not depend on $n$, it is $O(1)$ in all scenarios. We assume that a node has at most $\boldsymbol{d}$ dependents; $d$ is $O(n)$ in scenario S1 since we can have a graph where the first node has all other nodes as its adjuncts, but in scenarios S2–S4, $d$ is $O(1)$ because of the assumption about bounded degree. We assume that a node has at most $\boldsymbol{p}$ transitive parents in the deep tree and the surface tree; $p$ is $O(n)$ in scenario S1 because we can have a graph where each node depends on the previous node, but in scenarios S2–S4, $p$ is $O(\log n)$ because of the assumption that the height of the tree is $O(\log n)$. We assume that a node has at most $\boldsymbol{l}$ potential landing sites; $l$ is $O(n)$ in scenario S1, for in a graph consisting of a sequence of nodes depending on the previous node followed by a root, the root is yield-adjacent to every other node in the graph and therefore has every other node as a potential landing site; but in scenarios S2–S4, the height of the graph is bounded by $O(\log n)$, so the number of potential landing sites is at most $O(2 \log n) = O(\log n)$. Finally, we assume that a node has at most $\boldsymbol{g}$ potential governors, given a landing site; $g$ is $O(n)$ in scenario S1 and S2 because a node that is yield-adjacent to the root node can have every other node as its potential governor; in scenario S3, the island assumption ensures that $g$ is $O(\log n)$; and in scenario S4, $g = 1$ because the governor must coincide with the landing site.

The maximal number of graphs produced by a particular family of parsing operations at a given node is calculated as follows. *lex-chn*: There are at most $x$ different lex-chn operations, since the node has at most $x$ lexemes. **-del*: The lsite-del, gov-del, fill-del, and src-del operations for the node are unique because they specify a well-defined set of edges that must be deleted. *lsite-add/addx*: There are at most $ls$ lsite-add operations, since the node has at most $l$ potential landing sites, whose active lexemes license

| | max.ops. | S1 none | S2 balanced | S3 islands | S4 continuity |
|---|---|---|---|---|---|
| croles\|lexeme | $c$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| aroles\|lexeme | $a$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| sroles\|lexeme | $s$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| fframes\|lexeme | $f$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| fsrc\|fframe | $o$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| lexemes\|node | $x$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| degree\|node | $d$ | $O(n)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| dparents\|node | $p$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| lsites\|node | $l$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| govs\|lsite | $g$ | $O(n)$ | $O(n)$ | $O(\log n)$ | $O(1)$ |
| lex-chn | $x$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| lsite-del | 1 | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| gov-del | 1 | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| fill-del | 1 | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| src-del | 1 | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| lsite-add | $ls$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| lsite-addx | $lsx$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| comp-add | $gc$ | $O(n)$ | $O(n)$ | $O(\log n)$ | $O(1)$ |
| comp-addx | $gcx$ | $O(n)$ | $O(n)$ | $O(\log n)$ | $O(1)$ |
| adj-add | $ga$ | $O(n)$ | $O(n)$ | $O(\log n)$ | $O(1)$ |
| adj-addx | $gax$ | $O(n)$ | $O(n)$ | $O(\log n)$ | $O(1)$ |
| fill-add | $pf$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| fill-addx | $pfx$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| fill-chn | $pf$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| fill-chnx | $pfx$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| src-add | $o$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| gap-add | $lf$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| gap-addx | $lfx$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| glsite-add | $l$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| gcomp-add | $gcd$ | $O(n^2)$ | $O(n)$ | $O(\log n)$ | $O(1)$ |
| gcomp-addx | $gcdx$ | $O(n^2)$ | $O(n)$ | $O(\log n)$ | $O(1)$ |
| gadj-add | $gad$ | $O(n^2)$ | $O(n)$ | $O(\log n)$ | $O(1)$ |
| gadj-addx | $gadx$ | $O(n^2)$ | $O(n)$ | $O(\log n)$ | $O(1)$ |
| gap-par | 1 | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| worst | | $O(n^2)$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ |

Figure 7.11: Table showing the maximal number of parsing operations at a given node in the four scenarios.

at most $s$ different surface roles. Similarly, there are at most $lsx$ lsite-addx operations, since each landing site has at most $x$ non-active lexemes. *comp-add/addx*: There are at most $gc$ comp-add operations, since the node has at most $g$ potential governors given its landing site, and the active lexeme in a governor licenses at most $c$ different complement roles. Similarly, there are at most $gcx$ comp-addx operations, since a governor has at most $x$ non-active lexemes. *adj-add/addx*: There are at most $ga$ adj-add operations, since the node has at most $g$ potential governors given its landing site, and the active lexeme in the node licenses at most $a$ different adjunct roles. Similarly, there are at most $gax$ adj-addx operations, since the node has at most $x$ non-active lexemes. *fill-add/addx*: There are at most $pf$ fill-add operations, since the node has at most $p$ parents, and the active lexeme in a parent has at most $f$ filler frames. Similarly, there are at most $pfx$ fill-addx operations, since a parent has at most $x$ non-active lexemes. *fill-chn/chnx*: There are at most $pf$ fill-chn operations, since the governor of the filler has at most $p$ parents, and the active lexeme in each parent has at most $f$ filler frames. Similarly, there are at most $pfx$ fill-chnx operations, since each parent has at most $x$ non-active lexemes. *src-add*: There are at most $o$ src-add operations, since the filler frame of the node's filler licensor licenses at most $o$ different filler sources. *gap-add/addx*: There are at most $lf$ gap-add operations, since a node has at most $l$ potential landing sites which license at most $f$ different gapping fillers; similarly, there are at most $lfx$ different gap-addx operations because each node has at most $x$ non-active lexemes. *glsite-add*: There are at most $l$ glsite-add operations, since a node has at most $l$ potential landing sites and the landing site edge is uniquely determined by the gapping filler. *gcomp-add/addx*: There are at most $gcd$ gcomp-add operations, since the node has at most $g$ potential governors given its landing site, the active lexeme in the governor licenses at most $c$ different complement roles, and the governor has at most $d$ other dependents that the node may replace. Similarly, there are at most $gcdx$ gcomp-addx operations, since the governor has at most $x$ non-active lexemes. *gadj-add/addx*: There are at most $gad$ gadj-add operations, since the node has at most $g$ potential governors given its landing site, the node's active lexeme licenses at most $a$ different adjunct roles, and the governor has at most $d$ other dependents that the node may replace. Similarly, there are at most $gadx$ gadj-addx operations, since the governor has at most $x$ non-active lexemes. *gap-par*: Since there is only one way of ensuring parallelism between a source tree and a gapping tree,

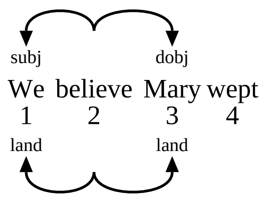


Figure 7.12: Analysis of "We believe Mary wept" after addition of "wept".

the gap-par operation results in a unique graph and is therefore $O(1)$ in all scenarios.

In Definition 7.30, we gave an informal definition of $k$-error operations on the basis of $k$-change operations on dependency trees. We will now restate this definition formally so that it is compatible with the elementary parsing operations for DG graphs introduced in this section.

**Definition 7.57** Let $\pi = (\pi_1, \ldots, \pi_k)$ be a sequence of $k$ elementary parsing operations that apply to the nodes $n_1, \ldots, n_k$ in a graph $G$ to produce a sequence $(G_0, \ldots, G_k)$ of graphs given by $G_0 = G$ and $G_i = \pi_i(G_{i-1})$. Then $\pi$ is called a *k-error operation* provided (a) $n_1$ is blamed by a cost function that signals a dynamic error in $G$, but no error in $G_1$ (cf. Definition 7.27); (b) each $n_i$ with $i > 1$ is affected by the parsing operation $\pi_{i-1}$ (cf. Definition 7.29); (c) each $n_i$ is blamed in $G_{i-1}$ by some cost function that signals a static error in $G_{i-1}$, but no error in $G_i$; and (d) no elementary parsing operation $\pi_i$ deletes edges, filler nodes, or lexeme choices introduced by another parsing operation $\pi_j$.

The following example demonstrates how $k$-error operations work.

**Example 7.58** Figure 7.12 shows the analysis of "We believe Mary wept" right after the word "wept" has been added to the graph. Since parsing is incremental, "Mary" is analyzed as the direct object of "believes" at this point. Figure 7.13 shows the search graph for $k$-error repair, ie, the sequences of elementary parsing operations that are compatible with a $k$-error operation. Each node in the search graph corresponds to a unique analysis (search state) labelled from 0 to 15, where 0 is the initial graph. Each edge in the search graph corresponds to an elementary parsing operation, and the edge label indicates the changes with respect to lexemes,

$$\begin{tikzcd}
 & 1 & & & & 6 \arrow[r, "-3\text{land}2"] & 9 \arrow[dr, "3\text{land}4"] & & \\
0 \arrow[ur, "2\text{land}4"] \arrow[dr, "4\text{land}2"'] & & & & & & & 12 & \\
 & 2 \arrow[r, "4\text{vobj}2"] & 3 \arrow[r, "2[\text{subj+vobj}]"] & 4 \arrow[r, "-3\text{dobj}2"] & 5 \arrow[uur, "3\text{subj}4"] \arrow[r, "-3\text{land}2"] \arrow[ddr, "3\text{subj}2"] & 7 \arrow[r, "3\text{land}4"] & 10 \arrow[ur, "3\text{subj}4"] & & \\
 & & & & & & & 13 \arrow[r, "-1\text{land}2"] & 15 \\
 & & & & & 8 \arrow[r, "-1\text{subj}2"] & 11 \arrow[ur, "1\text{subj}4"] \arrow[dr, "1\text{dobj}2"] & & \\
 & & & & & & & 14 &
\end{tikzcd}$$

Figure 7.13: Search graph for $k$-error operations applied to Figure 7.12.

edges, and filler nodes effected by the elementary parsing operation. The search graph can be a non-tree, as evidenced by search state 12 which can be reached both via search state 9 and search state 10 (ie, search state 12 represents a *transposition*).

The chart in Figure 7.14 gives detailed information about each search state, listing its parent id, the elementary parsing operation used to create the state, the error that was resolved by the parsing operation, the errors in the new graph (with new errors in italic), and the nodes affected by the parsing operation. For simplicity, we have assumed that the lexical entry for “believe” licenses two complement frames “subj+dobj” and “subj+vobj”, that “wept” licenses a single complement frame “subj”, and that no other words take any complements. Furthermore, we will assume that a missing landing site or governor at a node $n$ triggers the error $n^{\text{noland}}$ or $n^{\text{nogov}}$, that a landing site $l$ with a landed node $n$ without a governor triggers the error $l^{\text{badland-}n}$, that a governor $g$ with a complement $c$ that is not licensed by the active complement frame triggers the error $g^{\text{badcf-}c}$, that an island $i$ that detects an illegal extracted edge $h \xrightarrow{\text{s}}$ triggers the error $i^{\text{ext}(h \xrightarrow{\text{s}})}$, that a landing site $l$ with a discontinuous surface yield triggers the error $l^{\text{badyield}}$, that a bad word order at a landing site $l$ triggers the error $l^{\text{badwo}}$, that a governor $g$ with two complements $c_1$ and $c_2$ that fill the same complement role trigger the error $g^{\text{dblcf-}c_1+c_2}$, and that a dependent $d$ with a governor $g$ but no landing site triggers the error $d^{\text{badgov-}g}$.

For example, search state 1 is created by applying an lsite-add(2 land 4) operation to state 0, thereby resolving the error $2^{\text{noland}}$ and producing a new error $4^{\text{badland-2}}$ because node 4 lacks a governor. The operation affects nodes 2 and 4, ie, the next elementary parsing operation in the $k$-error sequence must apply to either node 2 or 4, and resolve at least one of the errors listed in the chart for state 1. Figure 7.12 and 7.15 show the analyses

| id | pid | operation | resolve | errors | affect |
|---|---|---|---|---|---|
| 0 | | | | $2^{\text{noland}}$ $2^{\text{nogov}}$ $4^{\text{noland}}$ $4^{\text{nogov}}$ | |
| 1 | 0 | lsite-add(2 land 4) | $2^{\text{noland}}$ | $2^{\text{nogov}}$ $4^{\text{noland}}$ $4^{\text{nogov}}$ $4^{\textit{badland-2}}$ | 2 4 |
| 2 | 0 | lsite-add(4 land 2) | $4^{\text{noland}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ $2^{\textit{badland-4}}$ $4^{\text{nogov}}$ | 2 4 |
| 3 | 2 | comp-addx(4 vobj 2) | $4^{\text{nogov}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ $2^{\textit{badcf-4}}$ | 2 4 |
| 4 | 3 | lex-chn(2[subj+vobj]) | $2^{\text{badcf-4}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ $2^{\textit{badcf-3}}$ | 2 3 |
| 5 | 4 | gov-del(3 dobj 2) | $2^{\text{badcf-3}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ $2^{\textit{badland-3}}$ $3^{\textit{nogov}}$ | 2 3 |
| 6 | 5 | comp-add(3 subj 4) | $3^{\text{nogov}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ $4^{\textit{ext(3 land 2)}}$ | 2 3 4 |
| 7 | 5 | lsite-del(3 land 2) | $2^{\text{badland-3}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ $2^{\textit{badyield}}$ $3^{\textit{noland}}$ $3^{\text{nogov}}$ | 2 3 |
| 8 | 5 | comp-addx(3 subj 2) | $3^{\text{nogov}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ $2^{\textit{badwo}}$ $2^{\textit{dblcf-1+3}}$ | 1 2 3 |
| 9 | 6 | lsite-del(3 land 2) | $4^{\text{ext(3 land 2)}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ $3^{\textit{noland}}$ $3^{\textit{badgov-4}}$ | 2 3 4 |
| 10 | 7 | lsite-add(3 land 4) | $3^{\text{noland}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ $3^{\textit{nogov}}$ $4^{\textit{badland-3}}$ | 3 4 |
| 11 | 8 | gov-del(1 subj 2) | $4^{\text{dblcf-1+3}}$ | $1^{\textit{nogov}}$ $2^{\text{noland}}$ $2^{\text{nogov}}$ $2^{\textit{badland-1}}$ | 1 2 |
| 12 | 9 | lsite-add(3 land 4) | $3^{\text{noland}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ | 3 4 |
| 12 | 10 | comp-add(3 subj 4) | $3^{\text{nogov}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ | 3 4 |
| 13 | 11 | comp-add(1 subj 4) | $1^{\text{nogov}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ $4^{\textit{ext(1 land 4)}}$ | 1 4 |
| 14 | 11 | comp-addx(1 dobj 2) | $1^{\text{nogov}}$ | $2^{\text{noland}}$ $2^{\text{nogov}}$ $2^{\textit{badcf-1}}$ | 1 2 |
| 15 | 13 | lsite-del(1 land 2) | $4^{\text{ext(1 land 4)}}$ | $1^{\text{noland}}$ $2^{\text{noland}}$ $2^{\text{nogov}}$ $4^{\textit{badland-1}}$ | 1 2 4 |

Figure 7.14: Chart with all *k*-error operations for Figure 7.12 (left).

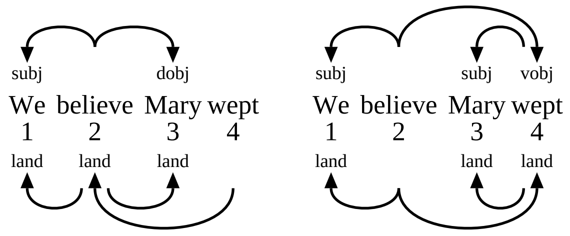


Figure 7.15: The analyses for states 1 (left) and 12 (right) in Figure 7.14.

corresponding to states 1, 12, 14, and 15.

From the worst-case number of graphs produced by individual parsing operations, we can now estimate the time complexity of composite parsing operations such as *k*-change, *k*-rconnect, *k*-displace, and *k*-error.

**Remark 7.59** To estimate the maximal number of graphs produced by each composite parsing operation, let *m* denote the maximal number of graphs

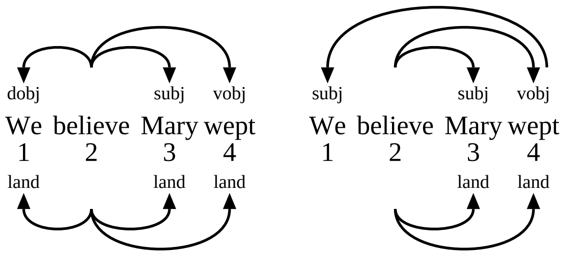


Figure 7.16: The analyses for states 14 (left) and 15 (right) in Figure 7.14.

| | **max.ops.** | **S1** **none** | **S2** **balanced** | **S3** **islands** | **S4** **continuity** |
|---|---|---|---|---|---|
| graphs/op | $m$ | $O(n^2)$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ |
| upd.cost/op | $u$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(1)$ |
| blame/err | $b$ | $O(n)$ | $O(\log n)$ | $O(\log n)$ | $O(1)$ |
| dyn.errors | $e$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| $k$-change | $n^k m^k$ | $O(n^{3k})$ | $O(n^{2k})$ | $O(n^k \log^k n)$ | $O(n^k \log^k n)$ |
| $k$-econnect | $(k-1)!ed^{k-1}m^k$ | $O(n^{3k-1} \log n)$ | $O(n^k \log n)$ | $O(\log^{k+1} n)$ | $O(\log^{k+1} n)$ |
| $k$-error | $e(bm)^k u^{k-1}$ | $O(n^{4k-1} \log n)$ | $O(n^k \log^{2k} n)$ | $O(\log^{3k} n)$ | $O(\log^{k+1} n)$ |

Figure 7.17: Upper bounds for the number of graphs produced by three families of parsing operations.

that can be produced by one application of an elementary parsing operation, let $u$ be the maximal number of nodes where the cost needs to be recomputed after one application of an elementary parsing operation, let $b$ denote the maximal number of nodes that can be blamed by a single error, and let $e$ denote the maximal number of simultaneous dynamic error nodes in a graph at any time. Figure 7.17 shows the maximal number of graphs produced by each composite parsing operation, and the upper bounds for the variables $m$, $u$, $b$, and $e$ in scenarios S1–S4.

To estimate the number of graphs produced by the different composite parsing operations, we note that a $k$-change operation can produce at most $n^k m^k$ different graphs, since for each elementary parsing operation we have a choice between $n$ nodes and at most $m$ different parsing operations. Similarly, a $k$-econnect operation can produce at most $(k-1)!ed^{k-1}m^k$ different graphs, since we must first choose one of the at most $e$ initial error nodes and one of the at most $m$ different parsing operations at that node, and for the $i$th subsequent parsing operation, we must choose one of the at most $(i-1)d$ nodes in the neighbourhoods of the first $i-1$ nodes and one of the at most $m$ parsing operations at that node, resulting in the upper bound $em \cdot (1dm) \cdots ((k-1)dm) = (k-1)!ed^{k-1}m^k$. Finally, a $k$-error operation can produce at most $e(bm)^k u^{k-1}$ different graphs, since we must first choose one of the at most $e$ dynamic errors and one of the at most $b$ nodes blamed by the error, then choose one of the at most $m$ different elementary parsing operations at the initial node, and for each of the $k-1$ subsequent parsing operations, we must choose one of the at most $ub$ different affected nodes produced by the preceding parsing operation and one of the at most $m$ different elementary parsing operations at the chosen node, resulting in the upper bound $eb \cdot m \cdot (ubm)^{k-1} = e(bm)^k u^{k-1}$.

We will now calculate upper bounds for the variables $m, u, b, e$. Since $m$ is the sum of the number of graphs produced by each family of elementary parsing operations, $m$ has the same complexity as the worst elementary parsing operation in Figure 7.11, ie, $m$ is $O(n^2)$ in scenario S1, $O(n)$ in S2, and $O(\log n)$ in S3 and S4. To obtain an upper bound for $u$, note that the cost operators listed in Figure 2.2, 2.3, and 3.1 are designed so that a change at a node can only affect its immediate neighbours in the graph, its adjacent nodes in the linear order, and the nodes on its extraction path. This means that a change at a node can affect the cost functions of at most $d+2+E$ where $E$ is the maximal length of an extraction path if the cost functions consist of a single cost operator, and at most $(d+2+E)^\kappa$ nodes if the cost functions consist of at most $\kappa$ cost operators. Noting that $u$ is always bounded by $n$, and that $E$ is at most $O(p)$ with $E = 1$ in S4, and combining these observations with the upper bounds for $d$ and $p$, we see that $u$ is $O(n)$ in S1, $O(\log^\kappa n)$ in S2–S3, and $O(1)$ in S4. However, in practice we believe that $\kappa$ is effectively 1 for any reasonable grammar — ie, if cost updates are managed carefully, we only need to recompute the costs of at most $O(\log n)$ nodes after each elementary operation. For this reason, we will assume $\kappa = 1$. To obtain an upper bound for $b$, we note that a node that is blamed by a cost function must be reachable by that cost function, ie, $b$ must be bounded by $u$ so that $O(b) = O(u)$. This gives the stated bounds for $b$. Finally, to obtain an upper bound for $e$, we will assume that all cost functions are dynamic and that $n$ is proportional with the elapsed time $t$. Footnote 5 on p. 105 shows that a dynamic error at a particular node is in the worst case repaired at the times $t_0^i$ for $i = 1, \ldots, \infty$, ie, the dynamic error occurs at most $O(\log n)$ times during the entire parsing. Since the parsing consists of at least $n$ operations, we see that the probability of a repair of a given dynamic error in a given parsing operation is at most $O(n^{-1} \log n)$. Since there are $n$ different nodes with $O(n)$ different cost functions, we see that the expected number $e$ of dynamic error nodes in the graph at any given time is at most $O(n \cdot n^{-1} \log n) = O(\log n)$.

**Remark 7.60** The upper bounds for $k$-error operations shown in Figure 7.17 are highly pessimistic. In practice, the requirement that an elementary parsing operation must be affected by the preceding parsing operation and resolve an error rules out most of the possibilities. For example, we obviously cannot resolve an agreement error or a missing landing site by adding a new complement to a node. This is also the reason why the search graph for $k$-error parsing in Figure 7.13 is relatively small — and far smaller than the corresponding search graph for $k$-econnect operations. Our upper bounds do not take this into account, and for this reason, $k$-error appears to be more inefficient than $k$-econnect in terms of worst-case time complexity, although in practice, we believe that $k$-error is far better than its worst-case estimates, and significantly more efficient than $k$-econnect.

**Remark 7.61** We can create a local optimality parser by searching for local improvements within a given family of parsing operations such as $k$-change, $k$-econnect, or $k$-error. To compute the time complexity of $k$-change, $k$-econnect, and $k$-error parsing, we note that for each graph produced by one of these operations, we need to recompute the cost of at most $u$ different nodes. Assuming that each cost function can be computed in constant time, a cost computation therefore takes $O(u)$ time. Moreover, following Remark 7.34, we will assume that a parser always terminates after at most $O(n)$ operations, ie, the time complexity of the parser can be computed by multiplying the time complexity of the family of parsing operations with $n$ and the time needed to update the costs, ie, by $O(un)$. This results in the worst-case time complexities shown in Figure 7.18.

**Remark 7.62** The upper bounds for $k$-change, $k$-econnect, and $k$-error parsing in Figure 7.18 show that the restrictions we have placed on the graphs in scenarios S1–S4 are important for making these parsers computationally efficient: in scenario S2, where we impose bounds on the height of the graph and the degree of nodes in the graph, the worst-case time complexity of $k$-error parsing falls from $O(n^{4k+1} \log n)$ to $O(n^{k+1} \log^{2k+1} n)$. In

| | **S1**<br>**none** | **S2**<br>**balanced** | **S3**<br>**islands** | **S4**<br>**continuity** |
|---|---|---|---|---|
| $k$-change parsing | $O(n^{3k+2})$ | $O(n^{2k+1}\log n)$ | $O(n^{k+1}\log^{k+1} n)$ | $O(n^{k+1}\log^{k} n)$ |
| $k$-econnect parsing | $O(n^{3k+1}\log n)$ | $O(n^{k+1}\log^{2} n)$ | $O(n\log^{k+2} n)$ | $O(n\log^{k+1} n)$ |
| $k$-error parsing | $O(n^{4k+1}\log n)$ | $O(n^{k+1}\log^{2k+1} n)$ | $O(n\log^{3k+1} n)$ | $O(n\log^{k+1} n)$ |

Figure 7.18: Time complexity for $k$-change, $k$-econnect, and $k$-error parsing.

scenario S3, where we also require the existence of island constraints that restrict the possible extraction paths to a singular spine of extraction, the time complexity falls to $O(n\log^{3k+1} n)$. This means that if human parsing is based on $k$-error parsing, there is strong evolutionary pressure on human languages to develop island constraints, thereby giving an alternative explanation for the existence of island constraints: they are not hard-coded in the human brain, but a consequence of the evolutionary pressure on all languages for making parsing as computationally efficient as possible.

So far, we have defined $k$-error parsing so that the parser must try all $k$-error parsing operations available to it in order to find the best possible improvement of the current analysis. However, if the parser tries to eliminate a given error and quickly finds a parsing operation that resolves it while improving the total cost of the analysis, it may be unnecessary to search through the entire neighbourhood for an even better alternative analysis. To address this problem, we will introduce a family of greedy $k$-error operations that can be used as an alternative to $k$-error.

**Definition 7.63** Let $a$ be an analysis, and let $\prec$ be a search-ordering on the set of all $k$-error operations on $a$. Given a dynamic error $e$ in $a$, we will say that a $k$-error operation $\pi$ is *greedy* with respect to $\prec$ and $e$ in $a$ if (a) it resolves $e$ while improving the total cost of the graph, and (b) it is the first parsing operation with this property in the order given by $\prec$.

**Remark 7.64** There are many ways to define a search-ordering on the set of $k$-error operations. For example, we get a best-first search if we use the cost of the resulting analyses as the primary sorting criterion, and an iterative deepening search if we use the number of parsing operations as the primary sorting criterion. We can get a probabilistically guided search by keeping track of past successes and failures so that we can estimate the probability that a given sequence of parsing operations will resolve a given error, or calculate the expected gain of applying a given sequence of elementary parsing operations. We can also use a combination of these criteria to define the search order. Finding the best search strategy is a matter of experimentation which we will leave to future research.

**Remark 7.65** The *k*-error search can be improved in other ways as well. To avoid performing the same search twice, we can use a transposition table to keep track of sequences of parsing operations that are equivalent (equivalence is quite easy to test since each *k*-error parsing sequence is characterized uniquely by the set of edges and filler nodes that are deleted, the set of edges and filler nodes that are added, and the set of lexeme changes). Likewise, we can constrain the memory resources allocated to the search by discarding the least plausible choice points, resulting in a beam search. Finally, we can get an any-time algorithm by stopping the search at any desired time and using the currently best improvement — this allows us to terminate the search quickly when computational resources are scarce, or in response to external events.

**Remark 7.66** We can also make greedy *k*-error parsing faster by using a first-improvement strategy where we try to resolve dynamic errors with large error ratios relative to the threshold before we proceed to smaller ones. This strategy requires us to stop the search once we have resolved an error and found an improved analysis. Stopping the search after the first improvement rather than proceeding until we know that no other improvement is better is a sensible strategy, since the parser will have to confront the remaining dynamic errors in the next parsing step anyway. In most applications, we expect the advantage in speed will outweigh the slightly increased risk of arriving at a suboptimal error-free analysis.

In this section, we have defined the elementary parsing operations used in DG, and computed their time complexity in four different scenarios. We have also defined *k*-error operations formally, and computed their time complexity. We have shown that in a linguistically realistic scenario with balanced graphs and island constraints (S3), the worst-case time complexity of *k*-error parsing is $O(n \log^{3k+1} n)$ — ie, *k*-error operations are (at least theoretically) a computationally efficient basis for local search. Finally, we have sketched how the average-case processing time of *k*-error operations can be improved by means of a number of different search strategies. In the following, we will argue that DG parsing is compatible with the psycholinguistic evidence about head-driven parsing, and that $k$-error operations are powerful enough to handle a wide variety of linguistic phenomena.

## 7.4 A head-driven model of human parsing

***Summary.*** *We present a DG-based model of human parsing that includes a formal top node that can act as the landing site for unanalyzed nodes and a set of restrictions on landing sites for unanalyzed nodes. We then present a number of psycholinguistic experiments that have been claimed to provide counter-evidence against all forms of head-driven parsers. We argue that this conclusion is based on overly restrictive assumptions about head-driven parsing, and describe how the DG parser accounts for the experimental results.*

The DG parser has many properties that makes it an attractive candidate for psycholinguistic parsing: it is strictly incremental, it is based on serial parsing with repair (cf. Hopf et al. 2003), it deals robustly with ungrammatical input, it has almost-linear time complexity and linear space complexity, and it leads to a natural explanation of island constraints and garden-paths. It is outside the scope of this dissertation to provide a full psycholinguistic evaluation of DG. However, since DG is head-driven — ie, it never attempts to predict edges between two nodes before both nodes have been encountered in the speech signal — and since many outstanding psycholinguists have produced evidence that seems to contradict head-driven parsing (Bader and Lasser 1994; Mecklinger et al. 1995; Konieczny 1996; Kamide and Mitchell 1999; Miyamoto et al. 1999; Aoshima et al. 2003), we will explain why the DG parser belongs to a family of head-driven parsers that was not anticipated by these authors, and outline how DG can account for the apparent counter-evidence. However, the psycholinguistic experiments cited above place so many restrictions on head-driven parsing that they must be credited as a major inspiration in the construction of the parsing model presented in this section. In the following, we will present the assumptions behind our DG model of human parsing, including the introduction of a formal top node in the graph and restrictions on the creation and deletion of temporary landing edges.

**Remark 7.67** We will assume that a DG graph always contains exactly one dummy node in initial position called TOP. Moreover, we will assume that any utterance can attach itself to TOP as a root adjunct, that TOP licenses any root adjunct as a landed node with edge type land, and that the parsing operations are designed so that TOP surface dominates all nodes in the graph at all times (cf. Remark 7.72).

**Definition 7.68** A landing edge $l \xrightarrow{e} n$ where $e$ is a subtype of the surface edge type tland is called a *temporary landing edge* with *temporary landing site* $l$ and *temporary landed node* $n$. The temporary landing edge is called *well-formed* if $l$ precedes $n$.

Temporary landing edges are licensed by a landing frame associated with the temporary landing site. In language learning, we are interested in identifying the lowest possible node that can function as a temporary landing site for a node that comes before its permanent landing site.

**Definition 7.69** Let $n$ be a non-top node in a graph $G$. The lowest surface edge $s \xrightarrow{e} s'$ with $s < n \leq s'$ is called the *roof edge* for $n$, and $s$ is called the *preliminary landing site* for $n$.

Because the surface tree is continuous and TOP surface dominates and precedes all other nodes, every non-top node has a unique roof edge and preliminary landing site.

**Remark 7.70** Temporary landing frames can be deduced automatically from a dependency treebank by assuming that if the permanent landing site does not coincide with the preliminary landing site, then the preliminary landing site is used as a temporary landing site for the node until the permanent landing site and governor is added to the graph. The treebank can also be used to calculate word order probabilities for temporary landed nodes relative to other landed nodes at the landing site.

**Example 7.71** Figure 7.19 shows a deep tree without fillers (top) and the associated preliminary landing sites (bottom) marked with dotted tland edges if the preliminary landing site does not coincide with the permanent landing site, and with solid land edges otherwise. The tree formed by the preliminary landing edges can be viewed as the minimal surface tree in which all landing edges point forward.

We can now propose a set of modifications to the parser that makes it compatible with the psycholinguistic evidence about head-driven parsing.

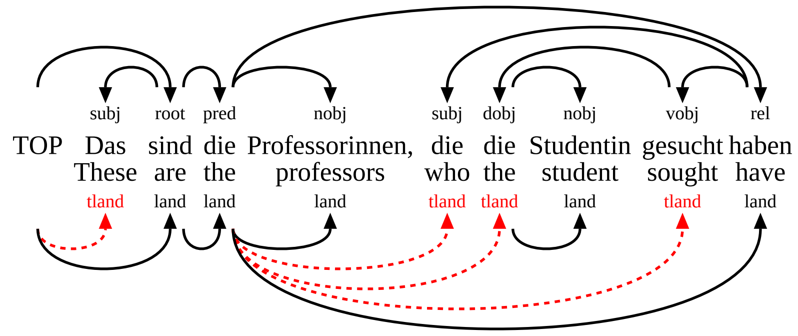

subj
root
pred
nobj
subj
dobj
nobj
vobj
rel
TOP Das sind die Professorinnen, die die Studentin gesucht haben
These are the professors who the student sought have
tland
land
land
land
tland
tland
land
tland
land

Figure 7.19: A dependency tree (top) with preliminary landing sites (bottom).

**Remark 7.72** We will assume that the DG parser is modified so that it always respects the constraints stated below. The first two constraints ensure that all nodes in the graph are surface dominated by TOP at all times.

- **HP1. Node addition at lowest temporary landing site**: when adding a node $n$ to the graph, the parser must create the surface edge $l \xrightarrow{\text{tland}} n$ where $l$ is the lowest temporary landing site dominated by the right boundary of TOP, or TOP if no such temporary landing site exists.
- **HP2. Landing site deletion ban**: an lsite-del operation cannot delete the landing site of a node $n$ unless it is immediately followed by an operation that creates a new landing site for $n$.

The following constraint ensures that temporary landing sites can only appear in the right periphery of TOP (ie, at a node that non-strictly dominates the right boundary of TOP). The underlying intuition is that at temporary landing sites, we are waiting for the arrival of a new landed node that will make an analysis of the temporary landed nodes possible, and if the graph is changed in a way where the temporary landing site is no longer at the right periphery, then continuity makes it unavailable as a landing site for a newly arrived node — ie, the temporary landed nodes will remain unanalyzed forever.

- **HP3. Right-peripherality of temporary landing sites**: landing site operations must ensure that all temporary landing sites are located in the right periphery of TOP.

The following constraint ensures that a node cannot lack a governor without having a temporary landing site (this does not preclude the possibility that a temporary landed node has a governor).

- **HP4. Temporary nature of ungoverned nodes**: a sequence of parsing operations is illegal if it results in a node that has a non-temporary landing site, but lacks a governor.

The following constraint ensures that once a node $n$ is analyzed as a temporary landed node, this analysis cannot be changed before all landed nodes at the landing site have a governor, with two exceptions: we assume that the reanalysis is safe if the new landing site precedes $n$, or if $n$ is the only left landed node of its new landing site.

- **HP5. Temporary landed node endurance**: a temporary landing site $l$ for a node $n$ cannot be replaced with a new landing site $l'$ unless (a) $l'$ precedes $n$; (b) $l'$ follows $n$ and $l'$ has no other left landed nodes or dependents; or (c) $l'$ follows $n$ and all temporary landed nodes at $l$ have a governor.

Finally, the following modification of the $\mathsf{extract}(a)$ cost operator ensures that we ignore island constraint violations at the top edge $l \xrightarrow{e_s} l'$ of the extraction path for a temporary landed node $n$ with $n < l'$ — ie, the island cost is temporarily ignored if it would disappear if $l'$ was analyzed as the permanent landing site for $n$.

- **HP6. Extraction path for temporary landed nodes**: in the extraction path for a temporary landed node $n$ with temporary landing site $l$, any edge from $l$ to a node $l'$ with $n < l'$ is invisible to the $\mathsf{extract}(a)$ cost operator.

The psycholinguistic evidence described in the following remarks place severe constraints on the kinds of head-driven parsers that can be used as models of human parsing, and rules out a great number of head-driven parsers proposed in the literature. However, the claim that this evidence rules out *all* head-driven parsers and proves that parsing must be predictive (ie, that the parser must predict the existence of heads before they have been encountered in the input) is based on the false premise that head-driven parsers always favour attachment to the first appearing potential governor in the speech signal. This is not necessarily true for head-driven parsers like the DG parser that take landing sites into account, and these parsers therefore deserve serious consideration as potential models of human parsing. Indeed, since it is widely recognized that it is difficult to formulate a predictive parsing model without overprediction and underprediction,[3] the lack of a predictive mechanism in the DG parsing model can actually be interpreted as an advantage.

**Remark 7.73** Bader and Lasser (1994) argue against head-driven parsing on the basis of local ambiguities in German such as (67) and (67′), where the initial pronoun can be interpreted either as the subject of the main verb or as the direct object of the subordinate verb, respectively. Their measurements show that (67′) leads to strong garden-path effects, whereas (67) does not lead to any processing difficulty. They assume that a head-driven parser leads to the opposite prediction, ie, that the subordinate verb is the preferred attachment site because it precedes the main verb.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| (67) | ..., daß | sie$_{\text{nom}}$ | ... | [zu | fragen] | erlaubt | hat. |
| | ..., that | she | ... | to | ask | permitted | has. |

‘that she has given permission to ask ...’

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| (67′) | ..., daß | [sie$_{\text{acc}}$ | ... | zu | fragen] | erlaubt | worden | ist. |
| | ..., that | her | ... | to | ask | permitted | been | is. |

‘that permission has been given to ask her ...”

However, these results are not incompatible with the DG model of human parsing. Figure 7.20 shows two stages in the DG parse of the shared part of (67) and (67′). The edge labels indicate the step at which the edge was created and possibly deleted, incremented for each edge modification — for example, the edge label “tland:2–9” indicates that the edge was created in step 2 and deleted in step 9.

In step 1, the word “sie” is added to the graph and analyzed as a temporary landed node of “daß” because there is no suitable governor in the graph yet. In step 2, the word “zu” is added to the graph and analyzed as a temporary landed node of “daß”. In steps 3–4, the word “fragen” is added to the graph and analyzed as a landed verbal object of “zu”. At this point, the parser can reanalyze “sie” as the direct object of “fragen” with landing

[3]In a review article, Phillips and Lau (2004) write about predictive parsers that “a long-standing challenge for this type of parser has been to constrain the predictive mechanisms in such a way that they do not overgenerate possible structure”.

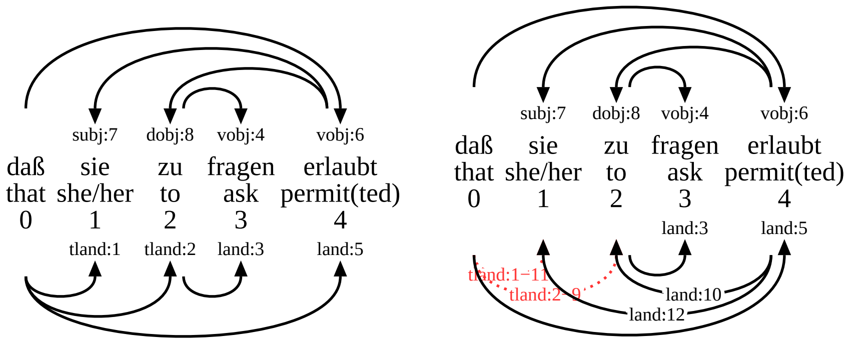


Figure 7.20: Two steps in the DG parse of (67) and (67′).

site "zu". While this reanalysis resolves the error that "sie" does not have a governor (an error with a low dynamic error cost because subjects often do not receive a governor that quickly), it also turns "zu" into the only temporary landed node at "daß". This is bad because "daß" is rarely observed as a preliminary landing site for "zu" infinitives, and for that reason, the parser sticks with its original analysis. In steps 5–6, the word "erlaubt" is added to the graph and analyzed as a landed verbal object of "daß".[4] Now "sie" can be analyzed as the subject of "erlaubt" in step 7, and "zu" can be analyzed as the direct object of "erlaubt" in step 8. The resulting analysis is shown in Figure 7.20 (left). Now both temporary landed nodes have a governor, so HP5 does not block a modification of their landing sites. In steps 9–10, "zu" is therefore reanalyzed as a permanent landed node of "erlaubt", and in steps 11–12, "sie" is reanalyzed as a permanent landed node of "erlaubt". The resulting analysis is shown in Figure 7.20 (right).

Figure 7.21 shows the subsequent steps in the DG derivation of (67). In steps 13–14, the word "hat" is added to the graph and analyzed as a landed verbal object of "daß". Since "daß" cannot have two verbal objects, "erlaubt" is reanalyzed as a perfect tense verb that functions as landed verbal object of "hat" in steps 15–18. Since perfect tense verbs cannot have a nonfiller subject, "sie" is reanalyzed as landed verbal subject of "hat" in steps 19–22. At this point, the parser can also analyze "sie" as the direct object of "fragen", but this is suboptimal because "hat" would then lack a subject.

[4]Bader and Lasser do not seem to have noticed that "erlaubt" is ambiguous between a perfect tense reading and a third person present tense reading.

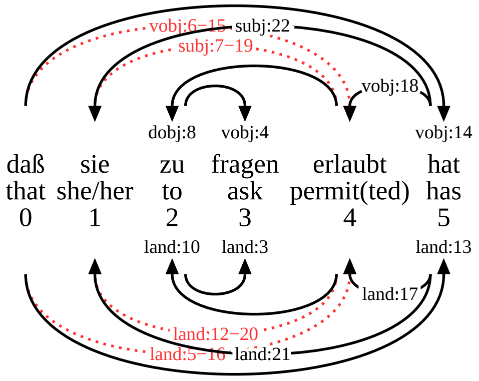


Figure 7.21: The subsequent steps in the DG parse of (67).

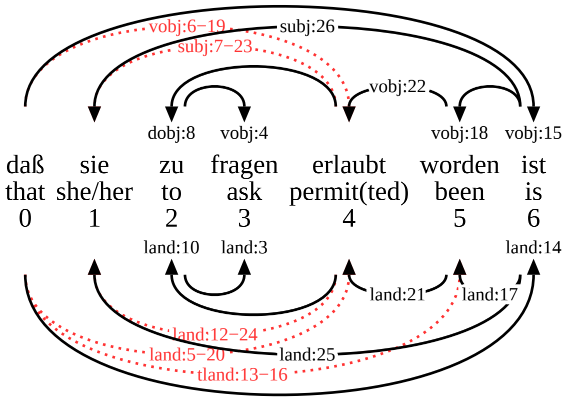


Figure 7.22: The subsequent steps in the A-parse of (67′).

Finally, the parser will analyze “sie” as the filler subject of “erlaubt” (not shown in Figure 7.21). The analysis of (67) is therefore unproblematic in the DG parser.

In contrast, even the two most optimistic and direct DG parses of (67′) shown in Figure 7.22 and 7.23 require much more extensive reanalysis. In step 13, the word “worden” is added to the graph and analyzed as a temporary landed node of “daß”. In steps 14–15, the word “ist” is added to the graph and analyzed as a landed verbal object of “daß”. In steps 16–18, “worden” is reanalyzed as the landed verbal object of “ist”. Alternatively, the parser could reanalyze “erlaubt” as the verbal object of “ist”, but this

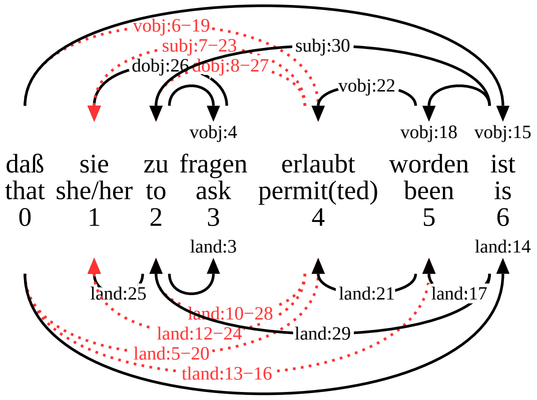


Figure 7.23: The subsequent steps in the B-parse of (67′).

would lead the parser astray. In steps 19–22, “erlaubt” is reanalyzed as the landed verbal object of “worden”. The parser now has two choices that will lead to reasonable analyses. In analysis A (shown in Figure 7.22), “sie” is reanalyzed as the subject of “hat” and “erlaubt” is reanalyzed as an indirect-object passive in steps 23–26. Alternatively in analysis B (shown in Figure 7.23), “sie” is reanalyzed as the direct object of “fragen” with landing site “zu” in steps 23–26; in steps 27–30, “zu” is then reanalyzed as the subject of “ist”, and “erlaubt” is reanalyzed as a direct-object passive.

Compared with the DG parse of (67), which requires 10 edge modifications after the addition of “hat”, the A and B parses of (67′) require 14 and 18 edge modifications after the addition of “worden ist”, and the derivation contains a larger number of choice points. Without a large-scale implementation of DG, it is difficult to determine whether the search graph for (67′) is so much larger than the search graph for (67) that the DG parser would predict a strong garden-path, but it would not be surprising if it did lead to this prediction. That is, the DG model of human parsing may well be compatible with the psycholinguistic evidence presented by Bader and Lasser.

**Remark 7.74** Mecklinger et al. (1995) argue against head-driven parsing on the basis of a self-paced reading and ERP-response experiment in which they examine local ambiguities in German such as (68) and (68′), where the relative pronoun is locally ambiguous between a subject reading and an

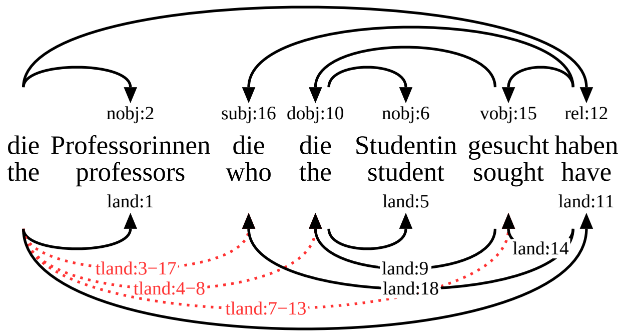


Figure 7.24: The DG parse of (68).

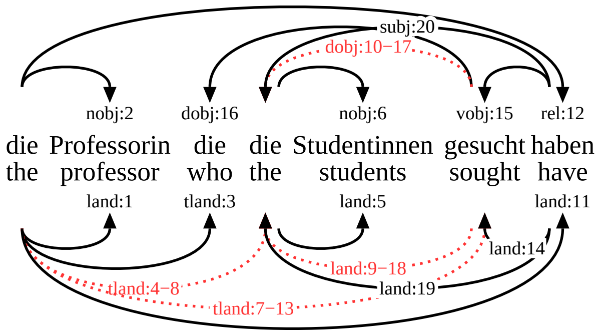


Figure 7.25: The DG parse of (68′).

object reading, but where number agreement at the relative verb forces one of the two readings. Their measurements show an increased processing load at object relatives compared to subject relatives, which they interpret as a predictive commitment to a subject reading.

(68) Das sind die Professorinnen, die die Studentin gesucht haben.
These are the professors who the student sought have$_{pl}$.
‘These are the professors who have sought the student.’

(68′) Das ist die Professorin, die die Studentinnen gesucht haben.
This is the professor who the students sought have$_{pl}$.
‘This is the professor who the students have sought.’

The DG parses of (68) and (68′) are shown in Figure 7.24 and 7.25. In both parses, the phrase “die Studentin(nen)” is analyzed as a direct object of "gesucht" in steps 8–10 right after "gesucht" has been added to the graph, in accordance with HP5(b). The parser cannot assign a subject to "gesucht" because perfect tense verbs do not license non-filler subjects, and word order constraints at the temporary landing site "die Professorin" prevents it from analyzing the initial relative pronoun "die" as the direct object of "gesucht". In (68), the direct object analysis turns out to be correct, whereas in (68′), "die Studentinnen" must be reanalyzed as the subject of "haben", thereby accounting for the increased processing load observed by Mecklinger et al..

**Remark 7.75** According to Konieczny (1996, p. 102), Steinhauer and Friederici have conducted an ERP-study of sentences with non-canonical word order such as (69), where an accusative NP comes before the nominative subject NP. Their results show that the non-canonical word order triggers an increased P600 at the accusative NP "den Professor". Since the P600 is believed to signal syntactic reanalysis, they interpret this as evidence for predictive attachment processes before the arrival of the verb.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| (69) | Daß | den | Professor | der | Student | gesucht | hat,... |
| | That | the$_{\text{dat}}$ | professor | the$_{\text{nom}}$ | student | sought | has,... |

'That the student has sought the professor,...'

Since "den" is analyzed as a temporary landed node of "daß" as soon as it is added to the graph, the temporary landing site "daß" will immediately detect the unusual word order of having an accusative NP as the first temporary landed node. This may trigger a small error at the temporary landing site, which may have the effect that the parser will try slightly harder to search for an alternative landing site. The DG parser may therefore well be compatible with the effect observed by Steinhauer and Friederici.

**Remark 7.76** Kamide and Mitchell (1999) argue against head-driven parsing on the basis of a questionnaire study and a self-paced reading experiment with Japanese attachment ambiguities such as (70), where a dative NP can attach either to the relative verb (low attachment) or to the main verb (high attachment), provided the verb licenses a dative object; the verbs "kasita" ("lent") and "miseta" ("showed") license a dative object, "yabutta" ("tore") does not. Their results show that the dominant reading is to interpret the dative NP as the indirect object of the main verb, and that examples that are only compatible with the LA reading have an increased

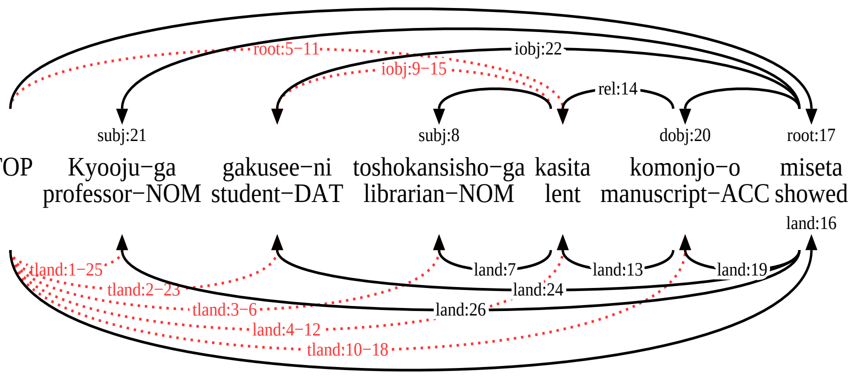


Figure 7.26: The DG parse of an LA+HA version of (70).

processing load compared to examples that are compatible with the HA reading. In their view, "this suggests that the parser must have been unravelling a commitment already made prior to this point – namely a link between the dative NP and the as-yet-unread matrix verb".

(70) Kyooju-ga gakusee-ni toshokansisho-ga kasita/yabutta
Professor$_{nom}$ student$_{dat}$ librarian$_{nom}$ lent/tore
mezurasii komonjo-o miseta/yabutta.
unusual ancient manuscript$_{acc}$ showed/tore.
'The professor showed/tore [the student]$_{HA}$ the unusual ancient manuscript which the librarian had lent/torn [the student]$_{LA}$.'

The DG parse of an ambiguous LA+HA version of (70) is shown in Figure 7.26, and the DG parse of an unambiguous LA version of (70) is shown in Figure 7.27. In both parses, "gakusee-ni" is initially analyzed as the indirect object of the first verb "kasita" in step 9, but because of HP5, its temporary landing site TOP remains unchanged.[5] For this reason, the attachment is undone in step 15 after "kasita" has been reanalyzed as a relative verb governed by "komonjo-o" in steps 13–14, otherwise the extraction out of the relative clause would violate the adjunct island constraint. After step 21, where the second verb "miseta/yabutta" has been added to the graph and "komonjo-o" has been analyzed as the direct object and "kyooju-ga"

[5]The initial analysis of "gakusee-ni" as the indirect object of the first verb accounts for the filled gap effect observed by Aoshima et al. (2003) in their first experiment.

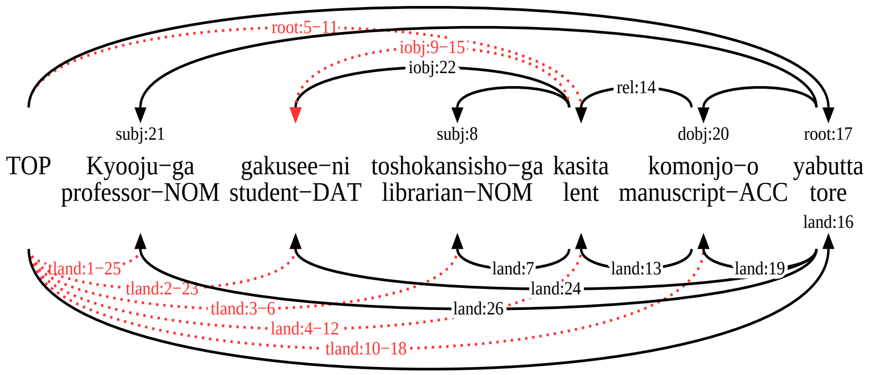


Figure 7.27: The DG parse of an LA version of (70).

as the subject of the second verb, the parser must find a governor for "gakusee-ni". In the LA+HA sentence, "gakusee-ni" can be analyzed as the indirect object of both verbs. However, at this point, it still has "TOP" as its temporary landing site, so because of the adjunct island constraint, the search for a potential governor is likely to stop at the main verb. For this reason, a preference for the HA reading can be expected. In the LA sentence, the unavailability of the main verb as the governor of "gakusee-ni" forces the parser to try the relative verb "kasita" as a potential governor; in steps 23–24, the island violation is resolved by reanalysing "gakusee-ni" as a landed node of "kasita", according to HP5(c). The behaviour predicted by the DG parser is therefore compatible with the effects reported by Kamide and Mitchell.

**Remark 7.77** Miyamoto et al. (1999) argue against head-driven parsing on the basis of a self-paced reading experiment with Japanese attachment ambiguities of the form "RC $N_3$ P $N_2$ P $N_1$" exemplified by (71), where a relative clause is followed by three potential attachment sites $N_3$, $N_2$, $N_1$.

(71) [Eda-ga oreteiru] / [Paati-de atta] / [Gakkou-made notta] shigemi-no
[branch broken] / [party met] / [school-to rode] bush
yoko-no hito-no ushiro-no jitensha-wa kireide ooki-katta.
beside person behind bicycle pretty big-was.
'The bicycle behind the person beside the bush that [has a broken branch] / [I met at the party] / [I rode to school] was pretty and big.'

The preferred reading (low, middle, and high attachment) is controlled

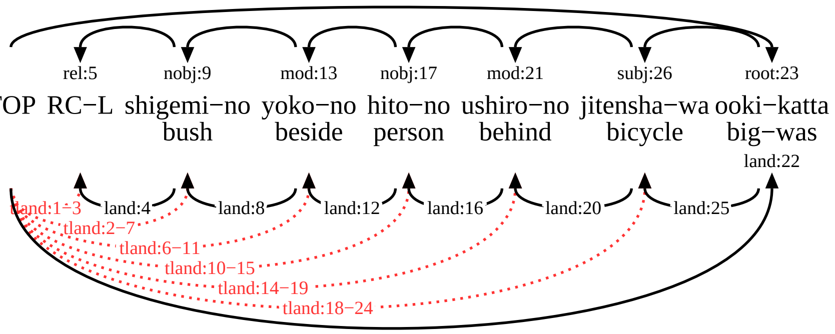


Figure 7.28: The DG parse of the LA version of (71).

by making only one of the three attachment sites semantically compatible with the relative clause. The experiments by Miyamoto et al. show that the LA reading is faster than the HA and MA readings in the $N_3$ region (with no difference between HA and MA), and that the HA reading is faster than the MA and LA readings in the $N_2$ and $N_1$ regions (with no difference between MA and LA). Miyamoto et al. explain this as a U-shaped preference that results from the interaction between a preference for LA (to make attachments as local as possible) and another preference for HA (to make attachments as close to the predicate as possible), so that MA is less favored than both LA and HA — a result that they view as incompatible with head-driven parsing.

The DG parses for the LA, MA, and HA versions of (71) are shown in Figure 7.28, 7.29, and 7.30. The most important difference between the parses is that in the LA parse, the relative clause can be reanalyzed as a permanent landed node of the noun it modifies as soon as the noun appears because of HP5(b), but because of HP5(c), this reanalysis of landing site is not possible before the appearance of the main verb in the MA and HA parses.

While we cannot offer detailed predictions about the time course of human parsing, we can at least show that it is quite possible for a DG parser to lead to the reading time rankings reported by Miyamoto et al., thereby removing the basis for their argument against head-driven parsing. For example, the DG parser will predict the observed rankings if we assume that the events *C* and *T* defined below always cause a fixed delay. *C*: following Miyamoto et al., we will assume that every preposition with a complex

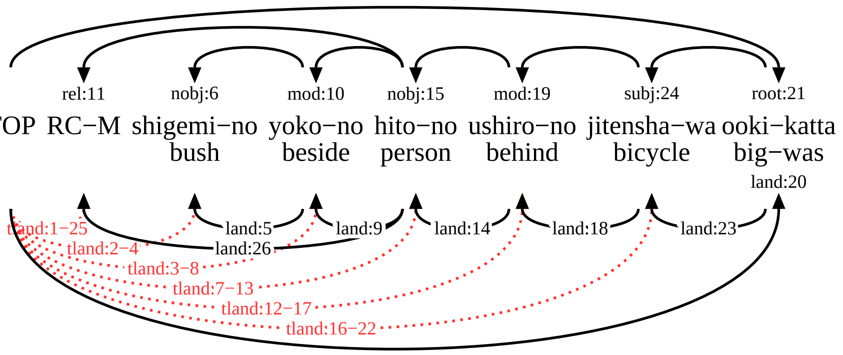


Figure 7.29: The DG parse of the MA version of (71).

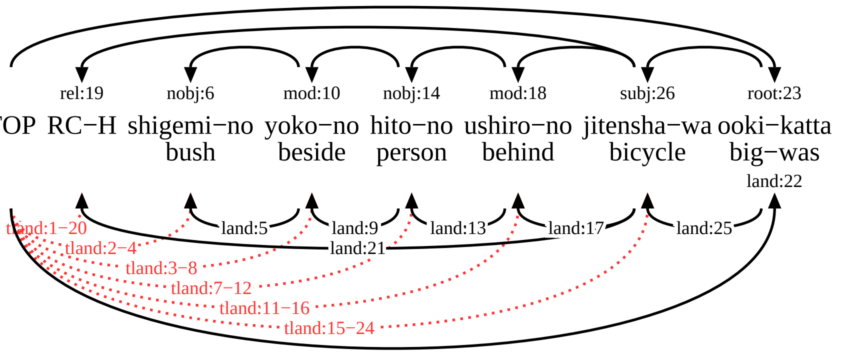


Figure 7.30: The DG parse of the HA version of (71).

argument (ie, a complement or governor that contains an embedded relative clause) causes a delay because of discourse complexity in the semantic module. *T*: we will assume that a non-transient increase in the number of temporary landed nodes at an existing temporary landing site causes a delay (ie, if a new temporary landed node is added and another is deleted immediately, there is no delay).

The table in Figure 7.31 records the number of times the events *C* and *T* occur in different regions of the input in the three parses. In the $N_3$ P region, the parser creates two temporary landed nodes in the MA and HA sentences, but only one temporary landed node in the LA sentence. In the $N_2$ P region, only the first preposition can be interpreted; it has a complex $N_3$ argument in the LA sentence, and a complex $N_2$ argument in

| | $N_3$ P | $N_2$ P | $N_1$ |
|---|---|---|---|
| **LA** | | $C$ | $C+C$ |
| **MA** | $T$ | $C$ | $C+C$ |
| **HA** | $T$ | | $C$ |
| **Ordering** | LA $<$ MA $=$ HA | HA $<$ LA $=$ MA | HA $<$ LA $=$ MA |

Figure 7.31: Hypothesized time-consuming events in different readings, and the predicted ordering with respect to reading time.

the MA sentence. Finally, in the $N_1$ region, both prepositions can be interpreted; in the LA and MA readings, both prepositions have a complex argument, whereas in the HA reading, only the second preposition has a complex argument. The resulting rankings, shown in the bottom line of the table, correspond to the U-shaped preference curve observed by Miyamoto et al.. Thus, although we do not know whether $C$ and $T$ are reasonable assumptions about human parsing, they do prove that there are DG parsing models that may be compatible with the evidence produced by Miyamoto et al.

In this section, we have presented a number of psycholinguistic experiments that have been claimed to provide evidence against all forms of head-driven parsing. We have shown that this claim is too strong: while the psycholinguistic evidence disproves many of the head-driven parsers proposed in the literature, the evidence does not rule out families of head-driven parsers that take landing sites into account. In particular, we have shown how the DG parser accounts for much of the apparent psycholinguistic counter-evidence. For this reason, head-driven parsing deserves serious consideration as an alternative to predictive parsing.

## 7.5 Evaluation of the DG system

***Summary***. *We describe five different strategies for evaluating the DG system, and the steps we have taken in this direction. We then describe a preliminary evaluation of a simplified DG parser with* 3*-displace and a small manually created grammar for German which shows that k-error parsing is powerful enough to deal with a wide range of word order phenomena, but that the simplified implementation is too inefficient to work with a grammar that is induced from a treebank. For this reason, a real evaluation of DG must await a large-scale implementation of the theory.*

The DG system can be evaluated from many different perspectives, including linguistic, psycholinguistic, statistical, and computational perspectives. In the following, we will describe the different evaluation strategies that the DG parser can be subjected to, and our work in this direction so far. Since most of the evaluation strategies require a computational implementation of the DG theory, and our implementation is still incomplete, a thorough evaluation of DG is unfortunately not possible yet.

**Remark 7.78 (linguistic evaluation)** The DG system can be evaluated with respect to its linguistic coverage, ie, with respect to the linguistic constructions that can be successfully analyzed, learned, and produced. For example, we can test whether a parser with a statistically induced DG grammar is capable of handling the wide range of constructions described in chapter 4: topicalizations, extrapositions, scramblings, control constructions, relatives, parasitic gaps, elliptic coordinations, punctuation, lexical alternations, etc.

**Remark 7.79 (psycholinguistic evaluation)** DG leads to specific predictions about repairs, garden-paths, and processing complexity. This allows the DG parser to be tested against the psycholinguistic evidence about human parsing (cf. Gernsbacher 1994), ie, to test whether DG makes the right predictions about the presence or absence of strong garden-paths, and whether the number of different graphs produced during DG parsing correlates with the processing times observed in eye-tracking experiments.

**Remark 7.80 (analysis of precision, recall, and generative adequacy)** The DG system can be evaluated with respect to its labelled and unlabelled *precision* and *recall*, which measure the system's ability to induce a grammar from a gold-standard and reproduce the gold-standard annotations from this grammar. They are defined as $|P \cap G|/|P|$ and $|P \cap G|/|G|$, respectively, where $G$ is the set of edges in the gold standard, and $P$ is the set of edges produced by the parser. Precision and recall are a natural evaluation metric when comparing different DG language models and different sets of DG parsing operations with each other, or when comparing the DG parser with statistical state-of-the-art dependency parsers such as the head-driven probabilistic context free parser by Collins (1997), the memory-based projective dependency parser by Nivre et al. (2004), and the unsupervised statistical dependency parser by Klein and Manning (2004).

In order to test the system's ability to avoid generating analyses which humans deem ungrammatical, we should also test its *generative adequacy*, which we tentatively define as the probability that the local neighbourhood around a word in a randomly generated analysis will be considered grammatical by an average human speaker.

**Remark 7.81 (error analysis)** The discrepancies between the analyses produced by a statistical DG system and the gold standard can be divided into *grammar errors* where the gold-standard analyses fails to be optimal with respect to the induced grammar, *parser errors* where the parser's analyses are suboptimal with respect to the induced grammar, *treebank errors* where the gold-standard contains errors of annotation, and *real ambiguities* where the discrepancies between the gold-standard and the parser are caused by real ambiguities in the text that cannot be easily resolved (ie, the parser's analysis and the gold-standard analysis are equally sensible). An examination of where these errors occur and how often is important for determining where the DG system fails, and how it can be improved.

**Remark 7.82 (complexity analysis)** A statistically induced grammar can also be used to evaluate the computational properties of the parser, such as the average-case time complexity for different families of parsing operations or the average size of the search space in $k$-error operations as a function of $k$. The time-complexity is particularly relevant for DG, where the parser is assumed to operate on entire discourses rather than isolated sentences. This also means that the parser should ideally be tested on a discourse treebank constructed along the lines presented in section 4.5.[6]

We have already performed most of the evaluations that can be performed without a computational implementation of the theory: in chapter 4 we have provided analyses for a wide range of linguistic phenomena; in section 7.4 we have evaluated the DG parser with respect to the psycholinguistic evidence about head-driven parsing; and in section 7.3 we have computed the worst-case time complexity of different versions of the DG parser. However, most aspects of the five evaluation strategies require a fully working DG system. Although we do not have a fully working

[6]The Center for Computational Modelling at the Copenhagen Business School is currently planning the creation of a discourse treebank on top of the Danish Dependency Treebank.

system yet, we have taken many of the required steps: we have created the DG-based Danish Dependency Treebank as our training material and gold-standard (cf. section 4.8), we have created a small-scale Prolog implementation of the DG theory (Kromann 2001), and we are currently working on a large-scale Perl implementation of the theory within our DTAG treebank tool (Kromann 2005).[7]

**Remark 7.83** In our efforts to evaluate DG, we have written a DG parser in Prolog with three simplifications that make the parser easier to implement: the parser does not handle secondary dependencies, it is based on $k$-displace with $k = 1, 2, 3$, and it uses a best-improvement brute-force search strategy in each parsing step. We have evaluated the parser on two different grammars: a small manually created grammar for German and a large statistical grammar induced from the Danish Dependency Treebank. The statistical grammar is based on a simplified version of the probabilistic language model presented in section 6.2, but with empirical probabilities with deleted interpolation (cf. Collins 1997) rather than hierarchical partition models.

**Remark 7.84** The manual DG grammar contains complement, adjunct, and landing frames as well as cost functions that encode word order, number and case agreement, selectional restrictions, island constraints, a weak preference for non-extraction, and a weak preference for short linear distances between nodes and their governors and landing sites. The evaluation of the simplified parser with the manual grammar (previously reported in Kromann 2001) showed that 3-displace is powerful enough to parse a wide range of word-order phenomena and ambiguities, including topicalizations and partial verb phrases such as (72), extrapositions and scramblings such as (73), cross-serial dependencies such as (74), and local ambiguities such as (75) and (76).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| (72) | Ein | Märchen | erzählen | wird | er | seiner | Tochter | können. |
| | A | fairy-tale | tell | will | he | his | daughter | be-able-to. |

[7]DTAG is written in Perl and PostScript and currently consists of 27,000 lines of source code. The full DG implementation is expected to require approximately 50,000 lines of source code. It will therefore take some time to complete the implementation. Compared with a probabilistic context-free parser such as (Collins 1997) or a probabilistic projective dependency parser such as (Nivre et al. 2004), the DG implementation suffers from the complexity of the underlying linguistic formalism.

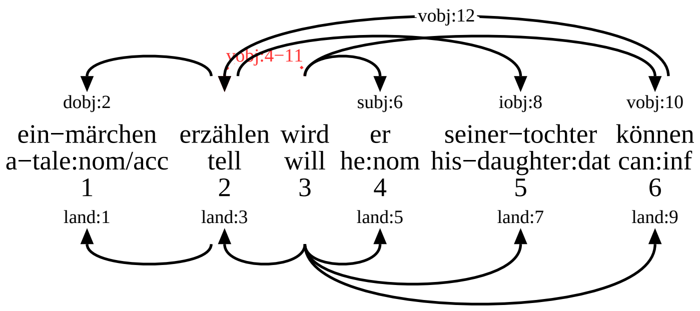


Figure 7.32: Simplified DG parse of the partial verb phrase (72).

‘He will be able to tell his daughter a fairy-tale.’

(73)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| weil | dem | Publikum | niemand | versprochen | hat, | die | Walzer | zu | spielen |
| because | the-DAT | audience | nobody-NOM | promised | has, | the | waltzes | to | play |

‘because nobody has promised the audience to play the waltzes’

(74)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| omdat | Piet | Marie | de | kinderen | zag | laten | zwemmen |
| because | Piet | Marie | the | children | saw | let | swim |

‘because Piet saw Marie let the children swim’

(75)

| | | | | | |
|---|---|---|---|---|---|
| Die | Walzer | haben | die | Musiker | erfreut. |
| The | waltzes | have | the | musicians | pleased. |

‘The waltzes pleased the musicians.’

(76)

| | | | | | |
|---|---|---|---|---|---|
| Die | Walzer | haben | die | Musiker | gespielt. |
| The | waltzes | have | the | musicians | played. |

‘The musicians played the waltzes.’

As an illustration of how the DG parser handles these sentences, Figure 7.32 shows how the partial verb phrase (72) is parsed by the parser. The left column in Figure 7.33 shows the parsing operation that was selected as the minimal-cost operation in each step, the right column shows all the alternative operations, and the cost associated with each operation is shown in subscript. At each step, the selected operation is superior to the alternatives because of the number of roots, case markings, or selectional restrictions. Initially, the parser has to read words 1 (“ein-märchen”) and 2 (“erzählen”) before it can add any edges, resulting in a graph with

| **step** | **best operation** | **alternative operations** |
|---|---|---|
| | $\mathbf{[1]}_{7.96}$ | |
| | $\mathbf{[2]}_{15.92}$ | |
| 1–2 | $\mathbf{[1^{land\text{-}2}_{dobj\text{-}2}]}_{7.96}$ | $[1^{land\text{-}2}_{iobj\text{-}2}]_{10.96}$ $[3]_{24.18}$ |
| | $\mathbf{[3]}_{16.22}$ | |
| 3–4 | $\mathbf{[2^{land\text{-}3}_{vobj\text{-}3}]}_{8.26}$ | $[2^{land\text{-}3}]_{12.22}$ $[4]_{24.18}$ |
| | $\mathbf{[4]}_{16.22}$ | |
| 5–6 | $\mathbf{[4^{land\text{-}3}_{subj\text{-}3}]}_{7.96}$ | $[4^{land\text{-}3}_{iobj\text{-}2}]_{10.26}$ $[4^{land\text{-}3}_{dobj\text{-}2}\ 1^{land\text{-}3}_{subj\text{-}3}]_{10.96}$ $[4^{land\text{-}3}]_{12.22}$ $[4^{land\text{-}3}_{dobj\text{-}2}\ 1^{land\text{-}2}_{iobj\text{-}2}]_{14.26}$ $[4^{land\text{-}3}_{dobj\text{-}2}\ 1^{land\text{-}3}_{iobj\text{-}2}]_{14.26}$ $[5]_{24.18}$ |
| | $\mathbf{[5]}_{15.92}$ | |
| 7–8 | $\mathbf{[5^{land\text{-}3}_{iobj\text{-}2}]}_{7.96}$ | $[5^{land\text{-}3}]_{11.92}$ $[5^{land\text{-}3}_{dobj\text{-}2}\ 1^{land\text{-}2}_{iobj\text{-}2}]_{13.96}$ $[5^{land\text{-}3}_{dobj\text{-}2}\ 1^{land\text{-}3}_{iobj\text{-}2}]_{13.96}$ $[5^{land\text{-}3}_{subj\text{-}3}\ 4^{land\text{-}3}_{iobj\text{-}2}]_{14.96}$ $[6]_{23.88}$ |
| | $\mathbf{[6]}_{15.92}$ | |
| 9–12 | $\mathbf{[6^{land\text{-}3}_{vobj\text{-}3}\ 2^{land\text{-}3}_{vobj\text{-}6}]}_{7.98}$ | $[3^{land\text{-}6}]_{11.92}$ $[6^{land\text{-}3}]_{11.92}$ |

Figure 7.33: Individual parsing operations in the parse of (72).

cost 15.92. In steps 1–2, it identifies identifies 1 as a landed direct object of 2, etc. A reanalysis takes place in steps 9–12, where 6 ("können") replaces 2 ("erzählen") as the direct object of 3 ("wird"), and 2 is reanalyzed as a verbal object of 3. The parser, grammar, and the detailed parses of the other tested sentences with graphical visualizations are available from http://www.id.cbs.dk/∼mtk/dg.

Obviously, we cannot conclude from these experiments that local optimality parsing with $k$-displace works in all cases just because it works in a few cases. However, the results are important because they show that the repairs needed for incremental serial parsing of many difficult word order phenomena are $k$-displace operations with a relatively low $k$. Since $k$-displace operations are a special kind of $k$-error operations, this indicates that $k$-error operations can deal robustly with many different word orders.

**Remark 7.85** We have also tested the simplified DG parser on a large-scale statistical grammar extracted from the Danish Dependency Treebank. The grammar has been created by extracting complement, adjunct, and landing frames from the treebank at the level of word classes. The resulting grammar is highly ambiguous — for example, there are 84 different complement frames for verbs, 17 for prepositions, 13 for nouns, and 10 for adjectives. Similarly, there are 11 different adjunct frames for verbs, 10 for prepositions, 13 for nouns, and 12 for adjectives.

However, with this level of ambiguity, the simplified parser cannot process sentences with more than 8–10 words. This is because it performs a brute-force search for local improvements, and because it computes the cost of the graph by reevaluating every node in the graph, which gives it a theoretical time complexity of $O(n^5)$. This shows that a simple-minded implementation of the DG theory does not work: in order to attain the theoretical worst-case time complexity of $O(n \log^{3k+1} n)$ for $k$-error parsing, we cannot use a brute-force search strategy in the local neighbourhoods, but need to use a best-first search strategy with an efficient implementation of graph updates and cost computations.

**Remark 7.86** Since DG is a relatively complicated syntax formalism and the DG parser is quite difficult to implement, it is reasonable to ask whether the efforts we have put into DG are justified, given the existence of highly successful broad-coverage statistical dependency parsers such as Collins (1997), Charniak (2000), Yamada and Matsumoto (2003), Nivre and Scholz (2004), and Nivre and Nilsson (2005), and rule-based dependency parsers such as Tapanainen (1999), Schneider et al. (2004), and Foth et al. (2004).

At this point, it is by no means clear whether a DG system will turn out to be better than these systems with respect to precision, recall, and speed. However, although DG still remains to be implemented and evaluated, it also has some interesting features that does distinguish it from most of these parsers, thereby making it worth exploring further. First of all, the DG parser is based on a linguistically sophisticated probabilistic language model that models both discontinuous dependencies, secondary dependencies, gapping coordinations, and island constraints. Secondly, the use of local search in the DG parser means that it can cope with language models where exact parsing is NP-hard — it is therefore conceivable that it can be extended to known NP-hard problems, such as word alignment and machine translation, without serious degradation of the time complexity. Finally, DG parsing holds some promise as a model of human parsing. In these respects, DG at least has a potential to address some problems that not all of the other parsers have the potential to address.

In this section, we have described five strategies for evaluating the DG parser — linguistic evaluation, psycholinguistic evaluation, analysis of precision and recall, error analysis, and complexity analysis. We have described the steps we have taken in this direction so far, and used our simplified DG parser to show that $k$-displace operations (and hence $k$-error operations) are powerful enough to deal with many challenging word order phenomena. However, we have also shown that the poor time complexity of the simplified parser makes it unsuitable for a large-scale evaluation of DG parsing. For this reason, most aspects of the five evaluation strategies have to await the completion of a full implementation of DG. Finally, we have argued that although DG is a complicated theory and the DG parser is difficult to implement, its potential to solve some important problems in parsing and psycholinguistics means that it is a framework that deserves further exploration.

## 7.6 Summary

In this chapter, we have extended the local optimality parsing algorithm so that it can deal with unsegmented speech signals with multiple speakers and noise. We have argued that incremental parsing provides important information about the parsing operations employed by humans, and we have proposed different families of parsing operations ($k$-connect, $k$-displace, $k$-error) that can be used to model human parsing. We have given a formal definition of the elementary parsing operations used in DG parsing and $k$-error operations, and estimated their theoretical worst-case time complexity in different scenarios. These estimates show that $k$-error parsing has worst-case time complexity $O(n \log^{3k+1} n)$ if the grammar defines a suitable set of island constraints, thereby explaining why island constraints seem to be universally present in all languages. We have also proposed a DG model of human parsing which uses temporary landing sites to explain the psycholinguistic results that have been argued to provide counter-evidence against head-driven parsing. Finally, we have described the ways in which the DG parser should be evaluated once it has been implemented, and the steps we have taken in this direction so far.

# Part V

# Conclusion

# Chapter 8

# Conclusion

*We shall not cease from exploration*
*And the end of all our exploring*
*Will be to arrive where we started*
*And know the place for the first time.*

T. S. Eliot, "Four Quartets"

In this chapter, we describe the current state of the field, what has been achieved in this dissertation, and what remains to be done. We also describe what we view as the most important new contributions in the dissertation, and what we view as the most important weaknesses.

## 8.1 The current state of the field

One of the central challenges in computational linguistics today is to formulate models of human language processing that are adequate from both a linguistic, statistical, computational, and psycholinguistic point of view — ie, models that can: (a) provide linguistically satisfactory analyses of natural language constructions and their grammaticality; (b) produce good statistical estimates of the relative frequency with which different constructions occur; (c) provide computationally efficient algorithms for parsing, generation, and grammar learning of text and speech; and (d) explain the psycholinguistic evidence about human language processing.

There are many ways in which a theory may fail to resolve these four issues. Some of the most successful statistical parsing models are based on context-free grammar, which is rather crude from a linguistic point of view. Most sophisticated linguistic theories (including GB, LFG, HPSG, TAG, CCG, and Word Grammar) were originally devised as rule-based theories where probabilities or utilities do not play any role. Prompted by the need to perform disambiguation in parsing, many of these theories have subsequently been equipped with probability models with state-of-the-art performance in statistical parsing. However, these probability models often embody statistical independence assumptions that are rather crude from a linguistic point of view, which means that the probability models often fail to have the same linguistic qualities as the underlying formalisms.

Computational efficiency is a challenge for most sophisticated linguistic theories, and even for context-free grammars, parsers are currently confined to parsing sentence-sized, pre-tokenized, pre-tagged text segments where low-polynomial average-time complexity is enough to ensure reasonable computational performance. However, as the field shifts its attention to discourse and speech processing with robust, massively ambiguous statistically induced grammars, we will need to find computationally efficient algorithms for parsing general discourse-sized, unsegmented text and speech — a task which probably requires algorithms with almost-linear average time complexity in order to ensure reasonable computational performance.[1]

The highly fragmentary and incomplete nature of the psycholinguistic evidence about human language processing makes it tempting to simply ignore the evidence altogether, and very few theories in linguistics and computational linguistics today have actually attempted to provide a full or even partial account of the psycholinguistic evidence. However, since the available psycholinguistic evidence is capable of ruling out many models of human language processing, it can be expected to provide important information about nature's solution to the many unsolved challenges in computational linguistics, and it is therefore increasingly viewed as an area that deserves serious attention.

In linguistics and computational linguistics today, the situation there-

[1]An $O(n^3)$ algorithm that works well for sentences with $n = 100$ is unlikely to be computationally feasible for discourses with $n = 100,000$.

fore seems to be that we are at a rather advanced stage with respect to the rule-based modelling of syntactic constructions and their grammaticality, but that we are only beginning to fill the gaps in our understanding of how to model the frequency of syntactic constructions and rephrase our notions of grammaticality in terms of frequency; of how to model the parsing of discourse-sized, unsegmented text and speech in a computationally efficient way; and of how to relate our linguistic models to the psycholinguistic evidence about human language processing.

## 8.2 What has been achieved

In this dissertation, we have presented a new linguistic theory, called Discontinuous Grammar, which we believe may have the long-term potential to resolve the linguistic, statistical, computational, and psycholinguistic issues (a)–(d) described above. The theory attempts to model how humans represent grammars and linguistic analyses, and how humans parse written and spoken language. It also includes a method for inducing probabilistic language models from annotated corpora, which is interesting both from a linguistic perspective because it can be viewed as a first step towards a full model of human language learning, and from a technological perspective because of its potential usefulness in natural language systems.

The new scientific contributions made in this dissertation fall in five main areas: the dependency formalism and the associated linguistic analyses, the local optimality parser, the statistical XHPM estimation algorithm, the probabilistic language model, and the axiomatic model of human communication. These are discussed in detail below.

### The dependency formalism and the associated linguistic analyses

Rather than basing our work on an existing syntax formalism, we have developed a new dependency-based syntax formalism. Most of the design choices in this formalism have been dictated by the desire to provide the best possible basis for serial parsing with repair and for statistical language modelling. For example, we have chosen a dependency-based rather than phrase-structure based design because phrase-structure based formalisms lead to unnecessarily complex repair operations, ie, to large edit distances between alternative readings in ambiguous sentences.

Most of the underlying intuitions behind the formalism have been borrowed from other syntax formalisms: the distinction between a continuous surface tree that determines word order and a possibly discontinuous deep tree that determines dependency relations; the complement-adjunct distinction; the principle of compositionality; the upwards movement principle; the notion of island constraints; the notion that the word order is controlled by landing sites; the use of fillers to encode secondary dependencies; the use of a productive lexicon with inheritance and lexical transformations; etc. Many of our analyses of particular linguistic phenomena are also inspired by the analyses made within other linguistic frameworks, even though the actual formulation may be slightly different because of the difference between the underlying frameworks.

However, our formalism also differs from most other syntax formalisms in important respects: eg, by its conception of grammar as a utility function expressed in terms of cost functions that can be used to encode probability models and violable linguistic constraints; by its linguistically motivated inventory of cost operators; by the details of its theory of movement and island constraints; by its analyses of sharing and gapping coordinations; by its analyses of parasitic gaps; by its analyses of inflectional morphology and lexical transformations such as passives and expletives; by its mechanisms for dealing with punctuation; by its mechanisms for dynamic feature functions and time-dependent cost functions; and by its conception of discourse structure, syntactic structure, and morphological structure as a single dependency structure augmented by anaphoric links.

In the dissertation, we have argued that the formalism is expressive enough to account for a wide range of linguistic phenomena, including various word order constructions (topicalizations, extrapositions, scramblings), filler constructions (control, relatives, and parasitic gaps), elliptic coordinations (sharing and gapping), punctuation, discourse structure and anaphora, as well as lexical alternations and other aspects of morphology. We have also tried to demonstrate that the grammar can be expressed either in a rule-based manner using hand-crafted cost functions, or, preferably, in a probabilistic manner using a probabilistic language model with the same set of conditioning variables as in the manually crafted cost functions. We have thereby substantiated the theory's claim to a reasonable degree of linguistic adequacy, our first desideratum for a linguistic theory.

## The local optimality parser

Rather than using an exact parsing algorithm where we ensure the computational feasibility by restricting the linguistic expressiveness of the syntax formalism or the probabilistic language model, we have argued that it is better to use a fast heuristic parsing algorithm that can be combined with arbitrarily complex language models, at the risk of sometimes ending up with a globally suboptimal parse (ie, a garden-path analysis). We have argued that this risk is not as large as it may seem at first sight because ambiguities tend to be local, ie, alternative analyses in a global or local ambiguity rarely differ with respect to more than 3 dependencies.

We have shown that it is possible to find heuristic parsing algorithms with the desired properties. In particular, we have proposed an incremental serial parsing algorithm with repair, called local optimality parsing, which constructs an analysis by means of a sequence of locally optimal structure-changing operations that only affect a small number of nodes in each step. We have proposed different families of parsing operations, such as $k$-connect, $k$-displace, and $k$-error, and argued that $k$-error seems to be powerful enough to be a useful basis for local optimality parsing. We have shown that local optimality parsing with $k$-error operations has worst-case time complexity $O(n \log^{3k+1} n)$ in a scenario with island constraints and balanced graphs, and $O(n^{k+1} \log^{2k+1} n)$ in a scenario without island constraints — that is, local optimality parsing does lead to almost-linear parsing, but island constraints are essential for achieving the almost-linear behaviour. If our model is true, this explains why island constraints are a universal property of all languages. We have also sketched how local optimality parsing can be extended to deal with segmentation, morphological analysis, and multi-speaker spoken dialogue.

In order to determine the model's psycholinguistic plausibility, we have examined the psycholinguistic experiments that have been claimed to disprove head-driven parsing. We have shown that, although our parsing model is head-driven, its use of temporary landing sites means that it can be refined in a way that makes it compatible with the apparent counter-evidence. On this basis, we have argued that there is a family of head-driven parsers that still deserve serious consideration as a psycholinguistic model of parsing, especially because it is difficult to design predictive, non-head-driven parsers so that they always make the right predictions.

In this way, we have shown that it is possible to construct a heuristic parsing algorithm with an almost-linear worst-case time complexity, and that this algorithm has a reasonable claim to psycholinguistic adequacy, at least with respect to the psycholinguistic evidence that we have examined.

### The probabilistic language model

We have proposed a generative probabilistic language model for DG. Although the model is inspired by the generative language models for CFGs proposed by Eisner (1996), Collins (1997), and Charniak (2000), there are important differences. First of all, the DG model includes specialized submodels for phenomena that are not directly modelled by the CFG models, such as discontinuous word order, secondary dependencies, gapping coordinations, punctuation marks, anaphora, and deep roots. Moreover, the generation of word order is no longer coupled to the generation of complements and adjuncts. Finally, the statistical estimation is mostly based on XHPM estimation with empirical priors, rather than on empirical estimation with linear interpolation or other methods of back-off.

### The statistical XHPM estimation algorithm

Most linguistic entities — including lexemes, dependency roles, ontological classes, inflections, etc. — are organized by means of classification hierarchies, and linguists routinely formulate their analyses in terms of the abstract classes in these hierarchies. Nevertheless, the existing statistical techniques in computational language modelling are mostly based on empirical estimators which cannot take advantage of the abstract classes provided by the hierarchies.

For this reason, we have proposed a new family of statistical distributions, called HPM and XHPM distributions, which can be used to model conditional random variables with a mixture of hierarchically organized variables, continuous variables, empirical variables, and variables with arbitrary distributions. XHPM distributions can be viewed as a correction to a prior distribution which redistributes probability mass to regions where the empirical probability is significantly higher than stipulated by the prior distribution. The prior distribution is then used to distribute the probability mass among the individual outcomes within each region. The regions are defined in terms of rest classes of the form $c_0 - (c_1 \cup \ldots \cup c_n)$ where $c_0, \ldots, c_n$ are classes in the hierarchies.

The estimation method proposed by Li and Abe (1998) can be viewed as a special, highly constrained instance of HPM distributions where the hierarchy is a tree rather than a general directed acyclic graph, where the prior distribution is uniform rather than arbitrary, and where all regions are given by classes in the hierarchy rather than by general rest classes.

### The axiomatic model of human communication

We have proposed an axiomatic model of human communication in which human speakers are capable of comparing the relative well-formedness of any two collections of linguistic analyses. We have proven that — given a simple, linguistically plausible set of assumptions about how human speakers compare these collections of analyses — the underlying preference ordering satisfies the axioms of utility and can therefore be expressed by means of a utility function. This proof has important theoretical ramifications, because it shows that if our axiomatic model is a reasonable idealization of the behaviour of human speakers, then human grammars can be encoded by means of utility or probability functions. That is, quantitative models of human language are not only necessary from an applied point of view, they are an issue of central theoretical importance in linguistics.

### Weaknesses

There are a number of weaknesses in the dissertation which external constraints have prevented me from addressing. The most serious weakness is the lack of a computational implementation which could be used to evaluate the theory with respect to its linguistic and computational adequacy; instead, I have only provided a blueprint for a computational implementation, and written the first 27,000 lines of source code in the DTAG program. Although linguistic theories should be allowed to exist for some time without an implementation and computational evaluation, I hope to eventually live up to my own ideals in this respect.

The DG theory presented in this dissertation has other shortcomings of a less serious nature. It stops short of providing a theory for anaphora, but should be extended so that it provides a linguistic analysis of anaphora, and a way of dealing with them in the lexicon, the parser, and the probabilistic language model. The probabilistic language model proposed in chapter 6 could be extended in many ways, as detailed at the end of the chapter. DG theory would also benefit from a thorough comparison with other theories in linguistics, computational linguistics, and psycholinguistics, both with respect to formal machinery, parsing and learning algorithms, linguistic analyses of particular phenomena, and empirical findings about human language processing.

Finally, a book should be written at least twice, and this book suffers from having been written only once.

## The way ahead

There are many issues that have not been addressed in this dissertation, but the syntax formalism proposed in this dissertation does at least offer partial progress towards achieving the ultimate goals outlined at the beginning of this chapter: (a) the DG formalism seems to provide a useful basis for the syntactic description of human languages; (b) it seems to provide a good framework for formulating linguistically adequate probabilistic language models; (c) it can be coupled with a parsing algorithm with almost-linear time complexity; and (d) the parsing model seems to be compatible with at least some of the psycholinguistic evidence about human parsing. We therefore believe that the syntax formalism proposed in this dissertation may be a step in the right direction which deserves further exploration.

# Part VI

# Appendices

# Appendix A

# DG language model

## A.1 Language model

### Model dg

```
# Define DG language model

model(dg,
    pri1a, pri2a, pri2b, pri2c, pri2d, pri3a, pri4a,
    sec1a, sec1b, sec2a, sec2b, sec3a, sec3b, sec4a, sec4b);

usemodel(dg);
```

## A.2 Submodels

### Submodel pri1a

```
# pri1a($this): primary expansion 1a of node $this where landing
# site and relative word order are identified

# Calculate normed probability
complex(pri1a, [$this],
    # Find landing site for node and landing frame with
    # previously generated landed nodes
    var($lsite, out(landing, $this)),
    var($lframe, list(sort($n, val(linorder, $n),
        ($lsite | in(landing, $lsite))
            + where($n, lt(val(genorder, $n),
                val(genorder, $this)))))),
    var($pos, node2pos($this, $lframe)),
    # Calculate prob. of current landing site and word order
    prob(pri1a_un, $this, $lsite, $lframe, $pos)

    # Normalize with summed probabilities for all other landing
    # sites and word orders
    / sum($lsite, trans($t, out(deep, $t), $this),
        # Find landing frame for this landing site
        var($lframe, list(sort($n, val(linorder, $n),
            ($lsite | in(landing, $lsite))
                + where($n, lt(val(genorder, $n),
                    val(genorder, $this))))),

        # Sum over all possible word orders at landing site
        sum($pos, sequence(0, abs($lframe)),
            prob(pri1a_un, $this, $lsite, $lframe, $pos))))
)
-> where(out(landing, $this));
```

## Submodel pri1a-un

```
# pri1a_un($this, $lsite, $lframe, $pos): calculate unnormed
# landing site and word order probability (primary1a) associated
# with node $this

complex(pri1a_un, [$this, $lsite, $lframe, $pos],
    # Find all island nodes
    var($islanddeps,
        list(trans($t, out(deep, $t) - $lsite, $this))),

    # Find product of all extraction probabilities
    product($islanddep, $islanddeps,
        var($island, out(deep, $islanddep)),
        min(1, prob(ldepext, $islanddep, $island, $this)
            / prob(ldep, $islanddep)))

    # Multiply with word order weights
    * product($ipos, sequence(0, abs($lframe)),
        # Find nearest locally landed complements
        var(prevc, last(member($lframe)
            - after(pos2node($ipos, $lframe))) - $lsite),
        var(nextc, first(member($lframe)
            - before(pos2node($ipos + 1, $lframe))) - $lsite),

        # Find nearest landed nodes between locally landed comps.
        var(preva, pos2node($ipos, $lframe) - ($lsite | $prevc)),
        var(nexta, pos2node($ipos + 1, $lframe)
            - ($lsite | $nextc)),

        # Find prob. of inserting $this (if $pos = $ipos) or STOP
        ifelse($pos == $ipos,
            prob(worder, $this, $lsite, $prevc, $preva,
                $nexta, $nextc) / prob(lnode, $this),
            prob(worder, STOP, $lsite, $prevc, $preva,
                $nexta, $nextc))
);
```

## Submodel pri2a

```
# pri2a($this): primary expansion 2a of node $this where
# complement frame is selected

complex(pri2a, [$this],
    prob(cframe, $this)
)
-> where($this - filler);
```

## Submodel pri2b

```
# pri2b($this): primary expansion 2b of node $this where
# fillers are generated. For each potential filler source,
# we determine the probability of generating a filler. The
# potential filler sources for a node are hardcoded into the
# model, and are computed by first finding a "neighbour" to
# the filler source (which is either the filler licensor,
# its governor, or the governor's governor if the filler
# licensor heads a relative clause attached to a subject),
# and then finding a source as any dependent or landed node
# at the neighbour, any parent of the neighbour, or any node
# that is extracted through the neighbour.

complex(pri2b, [$this],
    # Find potential neighbours of filler source, and potential
    # filler sources
    var($neighbours, list(
        $this
        | out(deep, $this)
        | out(subj, out(rel, $this)))),
    var($sources, list(
        in(deep|landing, member($neighbours))
        | out(deep, member($neighbours))
        | extract(member($neighbours)))),

    # Sort filler sources into left and right side of licensor
    # in decreasing distance to licensor, and unordered sources
    # in generation order
    var($lsources, sort($n, dist($n, $this),
        member($sources) + before($this))),
    var($rsources, sort($n, -dist($n, $this),
        member($sources) + after($this))),
    var($usources, sort($n, val($n, genorder),
        member($sources) - out(surf, $n))),

    # Find all possible filler edge type outcomes observed in the
    # "ftype" distribution for node "this" and attribute "ftype"
    var($ftypes, list(outcome(ftype, this, ftype))),

    # For each filler edge type and left filler source, decide
    # whether filler licensor generates filler conditioned on the
    # decisions made for the previous filler source
    product($src, member($lsources),
        var($prev, prev(member($lsources), $src)),
        product($ftype, member($ftypes),
            prob(fgen, $this, $src, $ftype, $prev)
        )
    )

    # For each filler edge type and right filler source, decide
    # whether filler licensor generates filler conditioned on the
    # decisions made for the previous filler source
    * product($src, member($rsources),
        var($prev, next(member($rsources), $src)),
        product($ftype, member($ftypes),
            prob(fgen, $this, $src, $ftype, $prev)
        )
    )

    # For each filler edge type and unordered filler source, decide
    # whether filler licensor generates filler conditioned on the
    # decisions made for the previous filler source
    * product($src, member($usources),
        var($prev, next(member($usources), $src)),
        product($ftype, member($ftypes),
            prob(fgen, $this, $src, $ftype, $prev)
        )
     )
)
-> where($this - filler);
```

## Submodel pri2c

```
# pri2c($this): primary expansion 2c of node $this where
# fillers are consumed or passed on to a complement. For each
# potential filler, we determine the probability of consuming the
# filler as a complement or an adjunct. Fillers are ordered in
# generation order.

complex(pri2c, [$this],
    # Find fillers passing through node $this sorted in
    # generation order
    var($fillers, sort($n, val(genorder, $n),
        list(filler + (in(deep, $this) | extract($this)))))

    # Find product of probabilities for each filler
    product($filler, member($fillers),
        # Find complement roles, empty complement roles, and
        # adjunct roles
        var($croles, sort(etype(in(comp, $this)))),
        var($emptycroles, list(member($croles)
            + where($crole, lt(val(genorder, in($crole, $this)),
                val(genorder, $filler))))),
        var($aroles, list(outcome(dtype, this, dtype) + adjunct)),
        var($dtype, etype(out(deep, $filler))),

        # Find probability of selected method of disposal
        prob(fdispose, $filler, $this, $dtype,
            abs($filler + in(deep, $this)), 1)
        / (    # Consume as complement...
            sum($crole, member($emptycroles),
                prob(fdispose, $filler, $this, $crole, 1, 1))

            # or pass on to complement...
            + sum($crole, member($croles),
                prob(fdispose, $filler, $this, $crole, 0, 1))

            # or for each adjunct role
            + sum($arole, member($aroles),
                # consume as adjunct...
                prob(fdispose, $filler, $this, $arole, 0, 1)

                # or pass on to adjunct
                + prob(fdispose, $filler, $this, $arole, 1, 1))))
)
-> where($this - filler);
```

## Submodel pri2d

```
# pri2d($this): primary expansion 2d of node $this where
# complements are generated and expanded.

complex(pri2d, [$this],
    # Find complements not satisfied by a filler
    var($freecomps, sort($n, val(genorder, $n),
        list(in(deep, $this) - filler))),

    # Find product of probabilities for generating each complement
    product($comp, member($freecomps),
        # Find probability of complement lexeme
        prob(comp, $comp, $this, etype(out(deep, $comp))))
)
-> where($this - filler);
```

## Submodel pri3a

```
# pri3a($this): primary expansion 3a of node $this where
# non-punctuation adjuncts are generated and expanded.

complex(pri3a, [$this],
    # Find probabilities of generated non-filler adjuncts
    product($adj, in(adj-pnct, $this) - filler,
        # First choose an adjunct role: if the governor has passed
        # a filler to the adjunct, then the adjunct role has already
        # been generated
        ifelse(filler + extract(out(deep, $adj)),
            1,
            prob(arole, $this, etype(out(deep, $adj))))

        # Choose adjunct lexeme given gov. lexeme and adjunct role
        * prob(adj, $adj, $this, etype(out(deep, $adj)))
    )

    # Find stopping probability
    * prob(arole, $this, STOP)
)
-> where($this - filler);
```

## Submodel pri4a

```
# pri4a($this): primary expansion 4a of node $this where
# gapping conjuncts are generated and expanded.
```

```
complex(pri4a, [$this],
    # Find gapping conjuncts
    var($gaps, list(in(conj, $this) + where($n, out(gap, $n)))),

    # Generate number of gapping conjuncts
    prob(gapnum, $this, abs($gaps))

    # Generate gapping conjuncts
    * product($gap, member($gaps),
        # Choose number of gapping dependents in each conjunct
        var($gapdeps, list(in(gapd, $gap))),
        prob(gapdepnum, $gap, $this, abs($gapdeps))

        # Generate gapping dependents
        * product($gapdep, member($gapdeps),
            # Generate replacement path...
            var($repl, out(repl, $gapdep)),
            prob(gappath, $gapdep, $this, path($this, $repl))

            # and lexeme for gapping dependent
            ifelse(out(comp, $repl),
                prob(comp, $gapdep, out(comp, $repl),
                    etype(out(comp, $repl))),
                prob(adj, $gapdep, out(adj, $repl),
                    etype(out(adj, $repl))))
        )
    )
)
-> where($this - (filler - gap));
```

## Submodel sec1a

```
# sec1a($this): generate deep roots if node is at the right
# periphery

complex(sec1a, [$this],
    # Generate number of deep roots
    var($droots, in(surface, $this) + where($n, -out(deep, $n))),
    prob(drootnum, $this, abs($droots))

    # Generate deep roots
    * product($droot, member($droots),
        # Generate deep root
        prob(droot, $this, $droot, etype(out(surface, $droot)))
    )
)
-> where($this + dominate(rightp(TOP, -before(any))));
```

## Submodel sec3a

```
# sec3a($this): generate punctuation marks at landing site $this

complex(sec3a, [$this],
    # Find landing frame for $this, with and without punctuation
    var($lframe, list(sort($n, val(linorder, $n),
        $lsite | in(landing, $lsite)))),
    var($lframe_np, list(member($lframe) - punctuation)),

    # Process punctuation fields in landing frame (ie,
    # each position between non-punctuation nodes in lframe)
    product($pos, sequence(0, abs($lframe_np)),
        # Find previous and next non-punctuation node
        var($prevnp, pos2node($pos, $lframe_np)),
        var($nextnp, pos2node($pos + 1, $lframe_np)),

        # Find oldest non-punctuation node to left and right
        var($oldestl, last(sort($n, val(genorder, $n),
            member($lframe_np) - $nextnp - after($nextnp)))),
        var($oldestr, last(sort($n, val(genorder, $n),
            member($lframe_np) - $prevnp - before($prevnp)))),

        # Find left and right punctuation field at
        # outside periphery
        var($outl, ifelse($pos > 0, '',
            list(leftp($this, -any, in(pnct, any))))),
        var($outr, ifelse($pos < abs($lframe_np), '',
            list(rightp($this, -any, in(pnct, any))
                + where($n, out(pnct, $n) + dominate($this))))),

        # Find punctuation field at given position
        var($pnctfield, list(member($lframe) + in(pnct, any)
            + after($prevnp) + before($nextnp))),

        # Process each punctuation mark in punctuation field
        product($pnct, member($pnctfield),
            # Find last open begin marker
            var($open, last(member($lframe) + in(pnct, $this)
                + before($pnct) + beginmarker
                    - where($n, out(match, $n) + before($pnct)
                        + in(pnct, $this)))),
            # Find directly preceding punctuation mark
            var($prevp, last(member($lframe) + in(pnct, any)
                + before($pnct) + after($prevnp))),

            # Find probability of punctuation mark
            prob(pnct, $this, $pnct, $prevnp, $nextnp,
                $prevp, $oldestl, $oldestr, $open, $outl, $outr)
        )

        # Find probability of stopping
        * prob(pnct, $this, STOP, $prevnp, $nextnp,
            last($pnctfield), $oldestl, $oldestr,
            $open, $outl, $outr)
    )
)
-> where($this - punctuation);
```

## A.3 Distributions

### Distribution adj

```
# adj($dep, $gov, $dtype): choice of adjunct $dep given
# governor $gov and adjunct role $dtype, modelled by an
# XHPM distribution

distribution(adj, [$dep, $gov, $dtype],
    xhpm()

    # Training material computed from $dep
    -> train(
        var($gov, out(deep, $dep)),
        var($dtype, etype(out(deep, $dep))))
    -> trainwhere(out(adj, $dep))

    # Augmentation
    -> augmentation(
        id(dep, $dep)
        + id(gov, $gov)
        + attr(gov, lex, val(lexeme, $gov))
        + attr(dep, lex, val(lexeme, $dep))
        + attr(dep, dtype, $dtype))

    # Specification of the probability model:
    # (dep.lex|gov.lex, dep.dtype)
    -> e(dep, dtype)
    -> c(any, any)
    -> given(gov, lex)
    -> given(dep, dtype)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype, hierarchy_edge)

    # Specification of prior distribution
    -> cnew_prior(
        prob(lexeme, rename(dep, this, subaug(dep, lex, aug()))))
    -> cgiven_prior(
        prob(lexeme, rename(gov, this, subaug(gov, lex, aug())))
        * prob(dtype, rename(dep, this, subaug(dep, dtype, aug()))))
);
```

## Distribution arole

```
# arole($this, $dtype): choice of adjunct role $dtype given
# governor $this, modelled by an XHPM distribution

distribution(arole, [$this, $dtype],
    xhpm()

    # Training material computed from $this
    -> train(
        nvar($dtype, STOP | etype(in(adj, $this))))
    -> trainwhere(in(adj, $this))

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, lex, val(lexeme, $this))
        + attr(this, dtype, $dtype))

    # Specification of the probability model: (this.dtype|this.lex)
    -> c(any, any)
    -> given(this, lex)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype, hierarchy_edge)

    # Specification of prior distribution
    -> cnew_prior(
        prob(dtype, subaug(this, dtype, aug())))
    -> cgiven_prior(
        prob(lexeme, subaug(this, lex, aug())))
)
-> where($this - filler);
```

## Distribution cframe

```
# cframe($this): ultimate complement frame for node $this
# modelled by an empirical distribution

distribution(cframe, [$this],
    xhpm()

    # Training material
    -> trainwhere($this - filler)

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, lex, val(lexeme, $this))
        + attr(this, dtype, etype(out(deep, $this)))
        + attr(this, stype, etype(out(landing, $this)))
        + attr(this, cframe,
            join(' ', sort(etype(in(comp, $this))))))

    # Variables and hierarchies
    -> c(this, lex|dtype|stype)
    -> x(this, cframe)
    -> given(this, lex|dtype|stype)
    -> hierarchy(lex, hierarchy_lexeme)
    -> hierarchy(dtype|stype, hierarchy_edge)

    # Prior distributions and estimation methods
    -> cgiven_prior(
        prob(node, subaug(this, lex|dtype|stype, aug())))
    -> xnew_est(empirical)
);
```

## Distribution comp

```
# comp($dep, $gov, $dtype): choice of complement $dep given
# governor $gov and complement role $dtype, modelled by an
# XHPM distribution

distribution(comp, [$dep, $gov, $dtype],
    xhpm()

    # Training material computed from $dep
    -> train(
        var($gov, out(deep, $dep)),
        var($dtype, etype(out(deep, $dep))))
    -> trainwhere(out(comp, $dep))
    # Augmentation
    -> augmentation(
        id(dep, $dep)
        + id(gov, $gov)
        + attr(gov, lex, val(lexeme, $gov))
        + attr(dep, lex, val(lexeme, $dep))
        + attr(dep, dtype, $dtype))

    # Specification of the probability model:
    # (dep.lex|gov.lex, dep.dtype)
    -> e(dep, dtype)
    -> c(any, any)
    -> given(gov, lex)
    -> given(dep, dtype)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype, hierarchy_edge)

    # Specification of prior distribution
    -> cnew_prior(
        prob(lexeme, rename(dep, this, subaug(dep, lex, aug()))))
    -> cgiven_prior(
        prob(lexeme, rename(gov, this, subaug(gov, lex, aug())))
        * prob(dtype, rename(dep, this, subaug(dep, dtype, aug()))))
);
```

## Distribution droot

```
# droot($this, $lsite, $stype): choice of deep root $this given
# landing site $lsite and landing edge type $stype, modelled by
# an XHPM distribution

distribution(droot, [$this, $lsite, $stype],
    xhpm()

    # Training material computed from $this
    -> train(
        var($lsite, out(surface, $this)),
        var($stype, etype(out(surface, $this))))
    -> trainwhere(out(surface, $this) + where($n, -out(deep, $this)))

    # Augmentation
    -> augmentation(
        id(this, $this)
        + id(lsite, $lsite)
        + attr(this, lex, val(lexeme, $this))
        + attr(lsite, lex, val(lexeme, $lsite))
        + attr(this, stype, etype(out(surface, $this))))

    # Specification of the probability model:
    -> c(any, any)
    -> given(lsite, lex)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, stype, hierarchy_edge)

    # Specification of prior distribution
    -> cnew_prior(
        prob(lexeme, subaug(this, lex, aug()))
        * prob(etype, rename(this, stype, etype,
            subaug(this, stype, aug()))))
    -> cgiven_prior(
        prob(lexeme, rename(lsite, this,
            subaug(lsite, lex, aug()))))
);
```

## Distribution drootnum

```
# drootnum($this, $nroots): choice of number of deep roots
# given node $this at the right periphery, modelled by an
# XHPM distribution

distribution(drootnum, [$this, $nroots],
    xhpm()

    # Training material computed from $this
    -> train(var($nroots,
        abs(in(surface, $this) + where($n, -out(deep, $n)))))
    -> trainwhere($this + dominate(rightp(TOP, -before(any), -any)))

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, lex, val(lexeme, $this))
        + attr(this, nroots, $nroots))

    # Specification of the probability model: (nroots|lex)
    -> x(this, nroots)
    -> c(this, lex)
    -> given(this, lex)
    -> hierarchy(any, lex, hierarchy_lexeme)

    # Specification of prior distribution
    -> cgiven_prior(prob(lexeme, subaug(this, lex, aug())))
    -> xnew_est(empirical)
);
```

## Distribution dtype

```
# dtype($this): dependent role of node $this modelled by an
# empirical distribution without smoothing

distribution(dtype, [$this],
    empirical()

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, dtype, etype(out(deep, $this))))

    # Hierarchy for deep edges
    -> hierarchy(any, dtype, hierarchy_edge)
);
```

## Distribution etype

```
# etype($this, $etype): edge type $etype for an edge at node
# $this modelled by an empirical distribution

distribution(etype, [$this, $etype],
    empirical()

    # Training material computed from $this
    -> train(nvar($etype, etype(out(any, $this))))

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, etype, $etype))

    # Hierarchy
    -> hierarchy(any, lex, hierarchy_edge)
);
```

## Distribution fdispose

```
# fdispose($filler, $disposer, $dtype, $method, $disposed):
# models the decision $decision (1 = dispose, 0 = do not dispose)
# made by a node $disposer when asked to dispose of a filler
# $filler to a dependent role $dtype using the consumption
# method $consume (1 = consume, 0 = pass on), modelled by an
# XHPM distribution.

distribution(fdispose, [$filler, $disposer, $dtype, $consume],
    xhpm()

    # Compute training material from $filler
    -> train(
        # Find all disposers
        nvar($disposer, dominate($filler)
            - dominate(out(fill, $filler)))

        # Find dtype for consumption/passing-on
        nvar($dtype, etype(in(deep, $disposer)
            + where($D, $D + (dominate($filler) | $filler)))),

        # Determine whether filler was consumed or passed-on
        var($consume, abs($disposer + out(deep, $filler))),
    )
    -> trainwhere($filler + filler)

    # Augmentation
    -> augmentation(
        id(filler, $filler) + id(disposer, $disposer)
        + attr(filler, dtype, $dtype)
        + attr(filler, ftype, etype(out(fill, $filler)))
        + attr(filler, consume, $consume)
        + attr(filler, lex, val(lexeme, $filler))
        + attr(filler, liclex, val(lexeme, out(fill, $filler)))
        + attr(filler, path, path(out(fill, $filler), $filler))
        + attr(disposer, lex, val(lexeme, $disposer)))

    # Variables and hierarchies
    -> x(filler, consume)
    -> c(any, any)
    -> new(filler, consume)
    -> new(filler, dtype)
    -> given(any, any)
    -> hierarchy(any, lex|liclex, hierarchy_lexeme)
    -> hierarchy(any, dtype|ftype, hierarchy_edge)

    # Prior distributions and estimators
    -> xnew_est(empirical)
    -> cnew_prior(prob(dtype, rename(filler, this,
        subaug(filler, dtype, aug()))))
    -> cgiven_prior(
        prob(lexeme, rename(filler, this,
            subaug(filler, lex, aug()))))
        * prob(lexeme, rename(filler, this,
            rename(filler, liclex, lex,
                subaug(filler, liclex, aug()))))
        * prob(lexeme, rename(disposer, this,
                subaug(disposer, lex, aug())))
        * prob(ftype, rename(filler, this,
            subaug(filler, ftype, aug())))
        * prob(fpath, rename(filler, this,
            subaug(filler, path, aug()))))
);
```

## Distribution fgen

```
# fgen($this, $src, $ftype, $prev): choosing whether the filler
# licensor $this should generate a filler for filler source $src
# with filler type $ftype, given the previous filler source $prev
# and its generated fillers, modelled by an XHPM distribution.

distribution(fgen, [$this, $src, $ftype, $prev],
    xhpm()

    # Training material constructed from $this for all non-filler
    # nodes
    -> train(
        # Find potential neighbours of filler source, and
        # potential filler sources
        var($neighbours, list(
            $this
            | out(deep, $this)
            | out(subj, out(rel, $this)))),
        var($sources, list(
            in(deep|landing, member($neighbours))
            | out(deep, member($neighbours))
            | extract(member($neighbours)))),

        # Sort filler sources into left and right side of
        # licensor in decreasing distance to licensor
        var($lsources, sort($n, dist($n, $this),
            member($sources) + before($this))),
        var($rsources, sort($n, -dist($n, $this),
            member($sources) + after($this))),
        var($usources, sort($n, val($n, genorder),
            member($sources) - out(surf, $n))),

        # Find all possible filler edge type outcomes observed
        # in the "ftype" distribution for node "this" and
        # attribute "ftype"
        var($ftypes, list(outcome(ftype, this, ftype))),

        # Non-deterministically run through all possible
        # combinations of filler sources and filler types
        ((nvar($src, member($lsources))
            + var($prev, prev(member($lsources), $src)))
        |(nvar($src, member($rsources))
            + var($prev, next(member($rsources), $src)))
        |(nvar($src, member($usources))
            + var($prev, prev(member($usources), $src)))),
        nvar($ftype, member($ftypes)))
    -> trainwhere($this - filler)

    # Augmentation
    -> augmentation(
        id(this, $this) + id(src, $src) + id(prev, $prev)

        # this: (a) lexeme; (b) deep role; (c) complement frame;
        + attr(this, lex, val(lexeme, $this))
        + attr(this, dtype, etype(out(deep, $this)))
        + attr(this, stype, etype(landing(deep, $this)))
        + attr(this, cframe,
            join(' ', sort(etype(in(comp, $this)))))

        # src: (a) ftype; (b) deep role; (c) landing field;
        # (d) local landing; (e) upwards part of path from $this
        # to $src; (f) downwards part of path from $this to $src;
        # (g) position relative to $this; (h) whether filler is
        # created or not
        + attr(src, ftype, $ftype)
        + attr(src, field,
            abs($src + before(out(landing, $src))))
        + attr(src, local,
            abs(out(landing, $src) + out(deep, $src)))
        + attr(src, uppath, uppath($this, $src))
        + attr(src, downpath, downpath($this, $src))
        + attr(src, pos, abs($src + before($this)))
        + attr(src, created,
            abs($src + out(source, in($ftype, $this))))

        # prev: (a) used as filler source for filler with type
        # $ftype by $this; (b) used as filler source for filler
        # with type $ftype by other filler licensor; (c) whether
        # uppath for $prev is the same as for $src
        + attr(prev, this_created,
            abs($prev + out(source, in($ftype, $this))))
        + attr(prev, other_created,
            abs($prev + out(source, in($ftype, any - $this))))
        + attr(prev, sameuppath,
            uppath($this, $prev) == uppath($this, $src))
    )

    # Variables
    -> e(src, ftype)
    -> x(src, created)
    -> c(any, any)
    -> new(src, created)
    -> given(any, any)


    # Hierarchies
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype|stype, hierarchy_edge)

    # Prior distributions and estimators
    -> xnew_est(empirical)
    -> cgiven_prior(this, cframe,
        prob(cframe, subaug(this, cframe|lex|dtype|stype, aug())))
    -> cgiven_prior(this, lex|dtype|stype,
        prob(node, subaug(this, lex|dtype|stype, aug())))
    -> cgiven_prior(any, any, empirical)
);
```

## Distribution fpath

```
# fpath($this): observed path from filler to filler licensor
# modelled by an empirical distribution

distribution(fpath, [$this],
    empirical()

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, path,
            path(out(fill, $this), $this)))
)
-> where($this + filler);
```

## Distribution ftype

```
# ftype($this): observed filler type modelled by an empirical
# distribution
```

```
distribution(ftype, [$this],
    empirical()

    # Training material
    -> trainwhere(out(fill, $this))

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, ftype,
            etype(out(fill, $this))))
);
```

### Distribution gapdepnum

```
# gapdepnum($gap, $gov, $ngapdeps): choice of number of gapping
# dependents in a gapping conjunct $gap with governor $gov,
# modelled by an XHPM distribution

distribution(gapdepnum, [$gap, $gov, $ngapdeps],
    xhpm()

    # Training material computed from $this
    -> train(
        var($gov, out(deep, $gap)),
        var($ngapdeps, abs(in(gapd, $gap))))
    -> trainwhere($this + gap)

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, lex, val(lexeme, $gov))
        + attr(this, ngapdeps, $ngapdeps))

    # Specification of the probability model: (ngaps|lex)
    -> x(this, ngapdeps)
    -> c(this, lex)
    -> given(this, lex)
    -> hierarchy(any, lex, hierarchy_lexeme)

    # Specification of prior distribution
    -> cgiven_prior(prob(lexeme, subaug(this, lex, aug())))
    -> xnew_est(empirical)
);
```

## Distribution gapnum

```
# gapnum($this, $ngaps): choice of number of gapping conjuncts
# given gap governor $this, modelled by an XHPM distribution

distribution(gapnum, [$this, $ngaps],
    xhpm()

    # Training material computed from $this
    -> train(
        var($ngaps, abs(in(conj, $this) + where($n, out(gap, $n)))))
    -> trainwhere($this - (filler - gap))

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, lex, val(lexeme, $this))
        + attr(this, ngaps, $ngaps))

    # Specification of the probability model: (ngaps|lex)
    -> x(this, ngaps)
    -> c(this, lex)
    -> given(this, lex)
    -> hierarchy(any, lex, hierarchy_lexeme)

    # Specification of prior distribution
    -> cgiven_prior(prob(lexeme, subaug(this, lex, aug())))
    -> xnew_est(empirical)
);
```

## Distribution gappath

```
# gappath($gapdep, $first, $path): choice of replacement path in
# gapping coordination with gapping dependent $gapdep and first
# conjunct $first, modelled by an XHPM distribution

distribution(gappath, [$gapdep, $first, $path],
    xhpm()

    # Training material computed from $gapdep
    -> train(
        var($first, out(deep, out(gapd, $gapdep))),
        var($path, path($first, out(repl, $gapdep))))
    -> trainwhere(out(gapd, $gapdep))

    # Augmentation
    -> augmentation(
        id(this, $gapdep)
        + attr(this, lex, val(lexeme, $first))
        + attr(this, path, $path))

    # Specification of the probability model: (ngaps|lex)
    -> x(this, path)
    -> c(this, lex)
    -> given(this, lex)
    -> hierarchy(any, lex, hierarchy_lexeme)

    # Specification of prior distribution
    -> cgiven_prior(prob(lexeme, subaug(gov, lex, aug())))
    -> xnew_est(empirical)
);
```

## Distribution landa

```
# landa($this, $lsite, $landa): a local landed complement $landc
# next to node $this at landing site $lsite modelled by an XHPM
# distribution

distribution(landa, [$this, $lsite, $landc],
    xhpm()

    # Training material constructed from $this for all nodes
    # that have a landing site
    -> train(
        # Find landing site for node and landing frame with
        # previously generated landed nodes
        var($lsite, out(landing, $this)),
        var($lframe, list(sort($n, val(linorder, $n),
            ($lsite | in(landing, $lsite))
                + where($n, lt(val(genorder, $n),
                    val(genorder, $this)))))),
        var($pos, node2pos($this, $lframe)),

        # Find nearest locally landed complements
        var($prevc, last(member($lframe)
            - after(pos2node($pos, $lframe))) - $lsite),
        var($nextc, first(member($lframe)
            - before(pos2node($pos + 1, $lframe))) - $lsite),

        # Find nearest landed nodes between locally landed comps.
        var($preva,
            pos2node($pos, $lframe) - ($lsite | $prevc)),
        var($nexta,
            pos2node($pos + 1, $lframe) - ($lsite | $nextc))
        # Find landa
        nvar($landa, $preva|$nexta))
    -> trainwhere(out(landing, $this))

    # Augmentation
    -> augmentation(
        id(lsite, $lsite) + id(landa, $landa)
        + concat(setof($n, lsite|landa,
            attr($n, lex, val(lexeme, $n))
            + attr($n, stype, etype(out(landing, $n)))
            + attr($n, dtype, etype(out(deep, $n)))
            + attr($n, local,
                abs(out(landing, $n) + out(deep, $n))))))

    # Variables and hierarchies
    -> c(any, any)
    -> given(lsite, any)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype, hierarchy_edge)
    -> hierarchy(any, stype, hierarchy_edge)

    # Prior distributions
    -> cnew_prior(prob(lnode,
        rename(landc, this, subaug(landc, any, aug()))))
    -> cgiven_prior(prob(lnode,
        rename(lsite, this, subaug(lsite, any, aug()))))
);
```

## Distribution landc

```
# landc($landc, $lsite): a local landed complement $landc at
# landing site $lsite modelled by an XHPM distribution

distribution(landc, [$this, $lsite, $landc],
    xhpm()

    # Training material constructed from $this for all nodes
    # that have a landing site
    -> train(
        # Find landing site for node and landing frame with
        # previously generated landed nodes
        var($lsite, out(landing, $this)),
        var($lframe, list(sort($n, val(linorder, $n),
            ($lsite | in(landing, $lsite))
                + where($n, lt(val(genorder, $n),
                    val(genorder, $this)))))),
        var($pos, node2pos($this, $lframe)),

        # Find nearest locally landed complements
        var($prevc, last(member($lframe)
            - after(pos2node($pos, $lframe))) - $lsite),
        var($nextc, first(member($lframe)
            - before(pos2node($pos + 1, $lframe))) - $lsite),

        # Find landc
        nvar($landc, $prevc|$nextc))
    -> trainwhere(out(landing, $this))

    # Augmentation
    -> augmentation(
        id(lsite, $lsite) + id(landc, $landc)
        + concat(setof($n, lsite|landc,
            attr($n, lex, val(lexeme, $n))
            + attr($n, stype, etype(out(landing, $n)))
            + attr($n, dtype, etype(out(deep, $n)))
            + attr($n, local,
                abs(out(landing, $n) + out(deep, $n))))))

    # Variables and hierarchies
    -> c(any, any)
    -> given(lsite, any)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype, hierarchy_edge)
    -> hierarchy(any, stype, hierarchy_edge)

    # Prior distributions
    -> cnew_prior(prob(lnode,
        rename(landc, this, subaug(landc, any, aug()))))
    -> cgiven_prior(prob(lnode,
        rename(lsite, this, subaug(lsite, any, aug()))))
);
```

## Distribution ldep

```
# ldep($dep, $gov): a dependency between a dependent $dep and a
# governor $gov modelled by an XHPM distribution

distribution(ldep, [$dep, $gov],
    xhpm()

    # Training material computed from $dep
    -> train(var($gov, out(deep, $dep)))
    -> trainwhere(out(deep, $dep))
    # Augmentation
    -> augmentation(
        id(dep, $dep)
        + id(gov, $gov)
        + concat(setof($n, dep|gov,
            attr($n, lex, val(lexeme, $n))
            + attr($n, dtype, etype(out(deep, $n)))
            + attr($n, stype, etype(out(landing, $n))))))

    # Specification of the probability model
    -> c(any, any)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype, hierarchy_edge)
    -> hierarchy(any, stype, hierarchy_edge)

    # Specification of prior distribution
    -> cnew_prior(
        prob(node, rename(dep, this, subaug(dep, any, aug())))
        * prob(node, rename(gov, this, subaug(gov, any, aug()))))
);
```

## Distribution ldepext

```
# ldepext($dep, $gov, $ext): a dependency between a dependent $dep
# and a governor $gov extracted through a node $ext modelled by
# an XHPM distribution

distribution(ldepext, [$dep, $gov, $ext],
    xhpm()

    # Training material computed from $dep
    -> train(
        var($gov, out(deep, $dep)),
        nvar($ext, extract(out(deep, $dep))))
    -> trainwhere(extract(out(deep, $dep)))

    # Augmentation
    -> augmentation(
        id(dep, $dep)
        + id(gov, $gov)
        + id(ext, $ext)
        + concat(setof($n, dep|gov|ext,
            attr($n, lex, val(lexeme, $n))
            + attr($n, dtype, etype(out(deep, $n)))
            + attr($n, stype, etype(out(landing, $n))))))
    # Specification of the probability model:
    # (gov.{lex,dtype,stype}, dep.{lex,dtype,stype}
    #     |ext.{lex,dtype,stype})
    -> c(any, any)
    -> given(ext, any)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype, hierarchy_edge)
    -> hierarchy(any, stype, hierarchy_edge)

    # Specification of prior distribution
    -> cnew_prior(
        prob(ldep, subaug(dep|gov, any, aug())))
    -> cgiven_prior(
        prob(node, rename(ext, this, subaug(ext, any, aug()))))
);
```

## Distribution lexeme

```
# lexeme($this): lexeme associated with a node $this modelled
# by an empirical distribution

distribution(lexeme, [$this],
    empirical()

    # Augmentation
    -> augmentation(
        id(this, $this) + attr(this, lex, val(lexeme, $this)))

    # Hierarchies
    -> hierarchy(any, lex, hierarchy_lexeme)

    # Unknown lexemes (lexemes with frequency <= 5)
    -> known_mincount(5)
    -> unknown_marker('UNK')
);
```

## Distribution lnode

```
# lnode($this): a node $this with lexeme, deep role, surface
# role, and locality flag modelled by an XHPM distribution

distribution(lnode, [$this],
    xhpm()

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, lex, val(lexeme, $this))
        + attr(this, dtype, etype(out(deep, $this)))
        + attr(this, stype, etype(out(landing, $this)))
        + attr(this, local,
            abs(out(landing, $this) + out(deep, $this))))

    # Variables and hierarchies
    -> c(this, any)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype, hierarchy_edge)
    -> hierarchy(any, stype, hierarchy_edge)

    # Prior distribution
    -> cnew_prior(
        prob(node, subaug(any, lex|dtype|stype, aug()))
        * prob(local, subaug(any, local, aug())))
);
```

## Distribution local

```
# local($this): local landing of node $this modelled by an
# empirical distribution

distribution(local, [$this],
    empirical()

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr($this, local,
            abs(out(landing, $this) + out(deep, $this))))
);
```

## Distribution node

```
# node($this): a node $this with lexeme, deep role, and surface
# role modelled by an XHPM distribution

distribution(node, [$this],
    xhpm()

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, lex, val(lexeme, $this))
        + attr(this, dtype, etype(out(deep, $this)))
        + attr(this, stype, etype(out(landing, $this))))

    # Variables and hierarchies
    -> c(this, lex|dtype|stype)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype, hierarchy_edge)
    -> hierarchy(any, stype, hierarchy_edge)

    # Prior distribution
    -> cnew_prior(
        prob(lexeme, subaug(any, lex, aug()))
        * prob(dtype, subaug(any, dtype, aug()))
        * prob(stype, subaug(any, stype, aug())))
);
```

## Distribution outl

```
# outl($this, $outl): model $outl value in punctuation decision
# by empirical distribution; the data are produced by the 'pnct'
# distribution

distribution(outl, [$this, $outl],
    empirical()

    # Augmentation
    -> augmentation(id(this, $this) + attr(this, outl, $outl))
);
```

## Distribution outr

```
# outr($this, $outl): model $outr value in punctuation decision
# by empirical distribution; the data are produced by the 'pnct'
# distribution

distribution(outr, [$this, $outr],
    empirical()

    # Augmentation
    -> augmentation(id(this, $this) + attr(this, outr, $outr))
);
```

## Distribution pnct

```
# pnct($this, $pnct, $prevnp, $nextnp, $prevp, $oldestl,
# $oldestr, $open, $outl, $outr): use an XHPM distribution to
# model the generation of a punctuation mark $pnct (or STOP) at a
# landing site $this between the landed non-punctuation nodes
# $prevnp and $nextnp, and immediately after the punctuation mark
# $prevp, where $oldestl is the oldest node in the landing field
# to the left, and $oldestr is the oldest node in the landing
# field to the right, $open is the last open begin marker at
# $this, and $outl and $outr are the punctuation marks from the
# outside periphery of $this that lie between $prevnp and $nextnp
# (ie, they are empty unless $prevnp or $nextnp mark the start or
# the end of the landing field).

distribution(pnct, [$this, $pnct, $prevnp, $nextnp, $prevp,
        $oldestl, $oldestr, $open, $outl, $outr],
    xhpm()

    # Training material constructed from $this
    -> train(
        # Find landing frame for $this, with and without punctuation
        var($lframe, list(sort($n, val(linorder, $n),
            $lsite | in(landing, $lsite)))),
        var($lframe_np, list(member($lframe) - punctuation)),

        # Process punctuation fields in landing frame (ie,
        # each position between non-punctuation nodes in lframe)
        nvar($pos, sequence(0, abs($lframe_np))),

        # Find previous and next non-punctuation node
        var($prevnp, pos2node($pos, $lframe_np)),
        var($nextnp, pos2node($pos + 1, $lframe_np)),

        # Find oldest non-punctuation node to left and right
        var($oldestl, last(sort($n, val(genorder, $n),
            member($lframe_np) - $nextnp - after($nextnp)))),
        var($oldestr, last(sort($n, val(genorder, $n),
            member($lframe_np) - $prevnp - before($prevnp)))),

        # Find left and right punctuation field at
        # outside periphery
        var($outl, ifelse($pos > 0, '',
            list(leftp($this, -any, in(pnct, any))))),
        var($outr, ifelse($pos < abs($lframe_np), '',
            list(rightp($this, -any, in(pnct, any))
                + where($n, out(pnct, $n) + dominate($this))))),

        # Find punctuation field
        var($pnctfield, list(member($lframe) + in(pnct, any)
            + after($prevnp) + before($nextnp))),
        # Process each punctuation mark in punctuation field
        nvar($pnct, member($pnctfield) | STOP),

        # Find last open begin marker
        var($open, last(member($lframe) + in(pnct, $this)
            + beginmarker - ($pnct | after($pnct)
                | where($n, out(match, $n) + in(pnct, $this)
                    - ($pnct | after($pnct)))))),

        # Find directly preceding punctuation mark
        var($prevp, last(member($lframe) + in(pnct, any)
            - before($prevnp) - after($nextnp)
            - after($pnct) - $pnct))
    )
    -> trainwhere($this - punctuation)
    -> feed(outl, id(this, $this) + attr(this, outl, $outl))
    -> feed(outr, id(this, $this) + attr(this, outr, $outr))

    # Augmentation
    -> augmentation(
        # Nodes
        id(this, $this) + id(pnct, $pnct)
        + id(prevp, $prevp) + id(open, $open)
        + id(prevnp, $prevnp) + id(nextnp, $nextnp)
        + id(oldestl, $oldestl) + id(oldestr, $oldestr)

        # Nodes modelled by 'lnode' distribution
        + concat(setof($n, this|pnct|prevp|open,
            attr($n, lex, val(lexeme, $n))
            + attr($n, dtype, etype(out(deep, $n)))
            + attr($n, stype, etype(out(surface, $n)))
            + attr($n, local, out(deep, $n) + out(surface, $n))))

        # Nodes modelled by 'dtype' distribution
        + concat(setof($n, prevnp|nextnp|oldestl|oldestr,
            attr($n, dtype, etype(out(deep, $n)))))

        # Outer peripheries for $this
        + attr(this, outl, $outl)
        + attr(this, outr, $outr)
    )

    # Variables and hierarchies
    -> c(any, any)
    -> given(pnct, any)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype|stype, hierarchy_edge)

    # Prior distributions
    -> cnew_prior(prob(lnode, rename(pnct, this,
        subaug(pnct, any, aug()))))
    -> cgiven_prior(
        # Nodes modelled by lnode distribution
        product($n, this|prevp|open,
            prob(lnode, rename($n, this,
                subaug($n, lex|dtype|stype|local, aug()))))

        # Nodes modelled by dtype distribution
        * product($n, prevnp|nextnp|oldestl|oldestr,
            prob(dtype, rename($n, this,
                subaug($n, dtype, aug()))))

        # Outer peripheries
        * prob(outl, subaug(this, outl, aug()))
        * prob(outr, subaug(this, outl, aug())))
);
```

## Distribution stype

```
# stype($this): surface role of node $this modelled by an
# empirical distribution

distribution(stype, [$this],
    empirical()

    # Augmentation
    -> augmentation(
        id(this, $this)
        + attr(this, stype, etype(out(landing, $this))))

    # Hierarchy
    -> hierarchy(any, stype, hierarchy_edge)
);
```

## Distribution worder

```
# worder($this, $lsite, $prevc, $preva, $nexta, $nextc): the
# insertion of a landed node between two local complements and
# twoneighbouring non-local complements (ie, landing frame
# (..., $prevc, ..., $preva, $this, $nexta, ..., $nextc))
# modelled by an XHPMdistribution
```

```
distribution(worder, [$this, $lsite, $prevc, $preva,
        $nexta, $nextc],
    xhpm()

    # Training material constructed from $this for all nodes that
    # have a landing site
    -> train(
        # Find landing site for node and landing frame with
        # previously generated landed nodes
        var($lsite, out(landing, $this)),
        var($lframe, list(sort($n, val(linorder, $n),
            ($lsite | in(landing, $lsite))
                + where($n, lt(val(genorder, $n),
                    val(genorder, $this)))))),
        var($npos, node2pos($this, $lframe)),
        nvar($pos, sequence(0, abs($lframe))),

        # Redefine $this to STOP if $pos != $npos
        var($this, ifelse($pos == $npos, $this, STOP)),

        # Find nearest locally landed complements
        var($prevc, last(member($lframe)
            - after(pos2node($pos, $lframe))) - $lsite),
        var($nextc, first(member($lframe)
            - before(pos2node($pos + 1, $lframe))) - $lsite),

        # Find nearest landed nodes between locally landed
        # complements
        var($preva,
            pos2node($pos, $lframe) - ($lsite | $prevc)),
        var($nexta,
            pos2node($pos + 1, $lframe) - ($lsite | $nextc)))
    -> trainwhere(out(landing, $this))

    # Augmentation
    -> augmentation(
        id(this, $this) + id(lsite, $lsite)
        + id(prevc, $prevc) + id(nextc, $nextc)
        + id(preva, $preva) + id(nexta, $nexta)
        + concat(
            setof($n, this|lsite|prevc|preva|nexta|nextc,
                attr($n, lex, val(lexeme, $n))
                + attr($n, stype, etype(out(landing, $n)))
                + attr($n, dtype, etype(out(deep, $n)))
                + attr($n, local,
                    abs(out(landing, $n) + out(deep, $n))))))
    # Variables and hierarchies
    -> c(any, any)
    -> given(any-this, any)
    -> hierarchy(any, lex, hierarchy_lexeme)
    -> hierarchy(any, dtype|stype, hierarchy_edge)

    # Prior distributions
    -> cnew_prior(prob(lnode, subaug(this, any, aug())))
    -> cgiven_prior(
        prob(landc,
            rename(prevc, this, subaug(prevc, any, aug())))
        * prob(landc,
            rename(nextc, this, subaug(nextc, any, aug())))
        * prob(landa,
            rename(preva, this, subaug(preva, any, aug())))
        * prob(landa,
            rename(nexta, this, subaug(nexta, any, aug()))))
);
```

# Appendix B

# Danish summary

## Status inden for forskningsfeltet

En af de vigtigste udfordringer i datalingvistik i dag er at formulere modeller for menneskelig sprogprocessering som er tilfredsstillende ud fra både en lingvistisk, statistisk, datalogisk og psykolingvistisk synsvinkel — dvs. modeller som leder til: (a) lingvistisk tilfredsstillende analyser af sproglige konstruktioner og deres grammatikalitet; (b) gode statistiske estimater for forskellige lingvistiske konstruktioners relative frekvens; (c) hurtige og effektive algoritmer til at parse, generere og lære talt og skrevet sprog; og (d) modeller af menneskelig sprogprocessering der stemmer overens med de psykolingvistiske observationer.

Der er mange måder hvorpå en teori kan være utilstrækkelig i forhold til disse fire udfordringer. Mange af de bedste statistiske parsingmodeller er baseret på kontekstfri grammatik, som er primitiv ud fra et lingvistisk synspunkt. Mange af de bedste lingvistiske teorier (herunder GB, L-FG, HPSG, TAG, CCG, og Word Grammar) er oprindeligt tænkt som regelbaserede teorier, hvor sandsynligheder og nytteværdier ikke spiller nogen rolle. Motiveret af behovet for disambiguering i parsing er mange af disse teorier senere blevet udstyret med sandsynlighedsmodeller der har ført til resultater på state-of-the-art niveau i statistisk parsing. Sandsynlighedsmodellerne i de regelbaserede teorier indeholder imidlertid ofte antagelser om statistisk uafhængighed som er forsimplede ud fra et lingvistisk synspunkt, hvilket bevirker at sandsynlighedsmodellerne sjældent har samme lingvistiske kvalitet som de underliggende formalismer.

Algoritmernes tidsforbrug er en udfordring i de fleste sofistikerede lingvistiske teorier, og selv inden for kontekstfri grammatik er de eksisterende parsere som regel begrænset til at parse præsegmenterede og prætokeniserede tekstsegmenter i sætningsstørrelse, hvor lav-polynomisk gennemsnitlig tidskompleksitet er nok til at sikre et tilstrækkeligt lavt tidsforbrug til at algoritmerne kan anvendes i praksis. Men efterhånden som forskningsfeltet skifter fokus til processering af diskurs og tale med robuste, massivt flertydige statistisk inducerede grammatikker, vil man få brug for at finde tidseffektive algoritmer til at parse generel, usegmenteret tekst og tale i diskursstørrelse — et problem som sandsynligvis kræver algoritmer med næsten-lineær gennemsnitlig tidskompleksitet for at sikre at algoritmerne er tilstrækkeligt hurtige til at kunne anvendes i praksis.[1]

Det er fristende at ignorere de psykolingvistiske observationer af menneskelig sprogprocessering fordi vores viden på dette område er fragmenteret og ufuldstændig. Måske derfor har få lingvistiske og datalingvistiske teorier forsøgt at give en fyldestgørende forklaring på de eksisterende psykolingvistiske observationer. Observationerne er imidlertid allerede nu tilstrækkeligt detaljerede til at udelukke mange modeller for menneskelig sprogprocessering. Observationerne giver dermed vigtig information om naturens løsning på de mange udfordringer som datalingvistikken står over for, og der er i feltet en voksende forståelse for at psykolingvistik bør opfattes som et område der fortjener større opmærksomhed.

I lingvistik og datalingvistik er man derfor nået langt med hensyn til den regelbaserede modellering af syntaktiske konstruktioner og deres grammatikalitet, mens man kun lige er begyndt at forstå hvordan man modellerer syntaktiske konstruktioners frekvens og definerer grammatikalitet vha. frekvens; hvordan man parser usegmenteret tekst og tale i diskursstørrelse på en tidseffektiv måde; og hvordan man relaterer de lingvistiske modeller af sprogbrugere til de psykolingvistiske observationer af menneskelig sprogprocessering.

[1]En $O(n^3)$ algoritme som har et rimeligt tidsforbrug ved sætninger med $n = 100$ kan være uanvendelig ved diskurser med $n = 100,000$.

## De opnåede resultater

Denne afhandling præsenterer en ny teori, *Diskontinuert Grammatik*, som vi mener har en chance for på længere sigt at kunne møde de lingvistiske, statistiske, datalogiske og psykolingvistiske udfordringer (a)–(d) vi netop har skitseret. Teorien modellerer hvordan mennesker repræsenterer lingvistiske analyser og grammatikker, og hvordan mennesker analyserer skrift og tale. Teorien indeholder også en metode til at inducere probabilistiske grammatikker fra lingvistisk annoterede korpora, hvilket er interessant både fra et lingvistisk perspektiv fordi det er første skridt mod en fuldstændig model for menneskelig sprogindlæring, og fra et teknologisk perspektiv på grund af de potentielle anvendelser i natursprogssystemer.

Denne afhandlings nye videnskabelige bidrag falder inden for fem hovedområder: dependensformalismen og de tilknyttede lingvistiske analyser, den lokale optimalitetsparser, den statistiske XHPM estimationsalgoritme, den probabilistiske sprogmodel, og den aksiomatiske model af sprogbrugere. Disse bidrag er beskrevet nærmere nedenfor.

### Dependensformalismen og de tilknyttede lingvistiske analyser

I stedet for at basere afhandlingen på en eksisterende syntaksformalisme, har vi udviklet en ny dependensbaseret formalisme. De fleste af vore designvalg har været dikteret af ønsket om at sikre det bedst mulige grundlag for seriel parsing med retteoperationer og for statistisk sprogmodellering. Vi har fx valgt et design baseret på dependensstruktur i stedet for frasestruktur fordi formalismer baseret på frasestruktur fører til unødvendigt komplicerede retteoperationer, dvs. til store retteafstande mellem konkurrerende læsninger i flertydige sætninger.

De fleste af intuitionerne bag formalismen har vi overtaget fra andre syntaksformalismer: distinktionen mellem et kontinuert overfladetræ der bestemmer ordstillingen og et muligvis diskontinuert dybdetræ der bestemmer funktor-argument strukturen; distinktionen mellem komplementer og adjunkter; kompositionalitetsprincippet; ideen om at ordstillingen bestemmes af landingsstedet; princippet om opadgående flytning; ideen om flytningsrestriktioner (island constraints); brugen af fillere til at indkode sekundære dependenser; brugen af et produktivt leksikon med nedarvning og leksikalske transformationer; osv. Mange af vore analyser af specifikke lingvistiske fænomener er inspireret af analyser der er blevet foreslået inden for andre lingvistiske teorier, selv om indsigterne ofte er formuleret på en anden måde pga. forskellene mellem de underliggende formalismer.

DG-formalismen adskiller sig dog også fra de fleste andre syntaksformalismer på vigtige områder: fx ved dens opfattelse af grammatikker som nyttefunktioner opbygget ud fra sandsynlighedsmodeller eller brydbare lingvistiske restriktioner; ved dens lingvistisk motiverede inventar af omkostningsoperatorer; ved detaljerne i dens teori om flytninger og flytningsrestriktioner; ved dens analyse af elliptiske koordinationer, parasitiske gaps, tegnsætning, morfologi, samt leksikalske transformationer i passiver og ekspletiver; ved dens dynamiske trækfunktioner og tidsafhængige omkostningsfunktioner; og ved dens opfattelse af diskursstruktur, syntaktisk struktur og morfologisk struktur som en samlet dependensstruktur suppleret med anaforiske relationer.

I afhandlingen har vi argumenteret for at formalismen kan håndtere en bred vifte af lingvistiske fænomener, herunder forskellige ordstillingsfænomener (topikalisering, ekstraposition og scrambling), filler-konstruktioner (kontrol, relativer og parasitiske gaps), elliptiske koordinationer (sharing og gapping), tegnsætning, diskursstruktur og anaforer, samt leksikalske alternationer og andre morfologiske aspekter. Vi har vist at grammatikken enten kan specificeres vha. manuelt skrevne regler eller, endnu bedre, vha. en probabilistisk grammatik induceret ud fra en træbank og en probabilistisk sprogmodel med samme forklarende variable som i en regelbaseret grammatik. Vi har dermed sandsynliggjort at teorien kan føre til tilfredsstillende lingvistiske analyser, vores første desideratum for en lingvistisk teori.

## Den lokale optimalitetsparser

I stedet for at anvende en eksakt parsingalgoritme hvor tidseffektiviteten sikres ved at begrænse de lingvistiske udtryksmuligheder i syntaksformalismen eller den probabilistiske sprogmodel, har vi argumenteret for at det er bedre at bruge en hurtig, heuristisk parsingalgoritme som kan kombineres med vilkårligt komplekse sprogmodeller, men med en risiko for indimellem at ende med en analyse der ikke er globalt optimal (dvs. en garden-path analyse). Vi har argumenteret for at denne risiko ikke er så stor som den umiddelbart kan virke, idet flertydigheder ofte er lokale, dvs. konkurrerende analyser i en globalt eller lokalt flertydig konstruktion sjældent afviger fra hinanden med mere end 3 dependenser.

Vi har vist at det er muligt at finde heuristiske parsingalgoritmer med de ønskede egenskaber. Vi har foreslået en inkrementel seriel parsingalgoritme med retteoperationer, kaldet lokal optimalitetsparsing, som konstruerer en analyse vha. en sekvens af lokalt optimale strukturmodificerende operationer som kun berører et lille antal knuder i hvert skridt. Vi har foreslået forskellige familier af parsingoperationer, så som $k$-connect, $k$-displace og $k$-error, og argumenteret for at $k$-error synes at være en brugbar basis for lokal optimalitetsparsing. Vi har vist at lokal optimalitetsparsing med $k$-error har værstefalds tidskompleksitet $O(n \log^{3k+1} n)$ i et scenarie med flytningsrestriktioner og balancerede grafer, og $O(n^{k+1} \log^{2k+1} n)$ i et scenarie uden flytningsrestriktioner — dvs. lokal optimalitetsparsing fører til næsten-lineær parsing, men flytningsrestriktioner er nødvendige for at opnå den næsten-lineære opførsel. Hvis vores model er sand, forklarer dette hvorfor flytningsrestriktioner findes universelt i alle sprog. Vi har også skitseret hvordan lokal optimalitetsparsing kan udvides til at håndtere segmentering, morfologisk analyse og talt dialog mellem flere sprogbrugere.

For at undersøge teoriens psykolingvistiske plausibilitet har vi gennemgået de psykolingvistiske eksperimenter som har været anført som argumenter imod kernedrevet parsing. Vi har vist at selv om DG-parseren er kernedrevet, kan den vha. midlertidige landingssteder forfines på en måde som kan forklare de psykolingvistiske observationer. På dette grundlag har vi argumenteret for at der findes en familie af kernedrevne parsere som stadig fortjener seriøs overvejelse som psykolingvistiske modeller for parsing, ikke mindst fordi det er vanskeligt at designe prædiktive, ikke-kernedrevne parsere så de altid laver de rigtige forudsigelser.

Vi har dermed vist at det er muligt at konstruere en heuristisk parsingalgoritme med en næsten-lineær værstefalds tidskompleksitet, og at denne algoritme er psykolingvistisk adækvat, i det mindste med hensyn til de psykolingvistiske eksperimenter vi har undersøgt.

## Den probabilistiske sprogmodel

Vi har foreslået en probabilistisk sprogmodel for DG. Selv om modellen er inspireret af de generative sprogmodeller for kontekstfrie grammatikker foreslået af Eisner (1996), Collins (1997) og Charniak (2000), er der vigtige forskelle. DG-modellen inkluderer først og fremmest specialiserede delmodeller for fænomener som ikke modelleres direkte i CFG-modellerne, som fx diskontinuert ordstilling, sekundære dependenser, gapping-koordinationer, tegnsætning, anaforer og dybe rødder. Genereringen af ordstilling er heller ikke koblet til genereringen af komplementer og adjunkter. Endelig er den statistiske estimationsmetode primært baseret på XHPM-estimation med aprioriske empiriske fordelinger, i stedet for på empirisk estimation med linær interpolation eller andre back-off metoder.

## Den statistiske XHPM estimationsalgoritme

De fleste lingvistiske entiteter — som fx leksemer, dependensroller, ontologiske klasser, bøjninger, osv. — kan organiseres vha. klassifikationshierarkier, og lingvister formulerer rutinemæssigt deres analyser vha. de abstrakte klasser i disse hierarkier. De eksisterende estimationsmetoder i statistiske sprogmodeller er ikke desto mindre i overvejende grad baseret på empiriske estimatorer som ikke kan drage fordel af de abstrakte klasser som hierarkierne indeholder.

Vi har derfor foreslået en ny familie af statistiske fordelinger, kaldet HPM- og XHPM-fordelinger, som kan modellere betingede stokastiske variable med en blanding af hierarkisk organiserede variable, kontinuerte variable, empiriske variable og variable med arbitrær fordeling. XHPM-fordelinger kan opfattes som korrektioner til aprioriske fordelinger der flytter sandsynlighedsmasse til områder hvor den empiriske sandsynlighed er signifikant højere end den aprioriske fordeling foreskriver. Den aprioriske fordeling benyttes derefter til at distribuere sandsynlighedsmassen mellem de enkelte udfald i hvert område. Områderne er defineret vha. restklasser af formen $c_0 - (c_1 \cup \ldots \cup c_n)$ hvor $c_0, \ldots, c_n$ er klasser i hierarkierne.

Den estimationsmetode som Li and Abe (1998) har foreslået, er et restriktivt specialtilfælde af HPM-fordelinger hvor hierarkiet er et træ i stedet for en generel orienteret acyklisk graf, hvor den aprioriske fordeling er uniform i stedet for arbitrær, og hvor alle områder er defineret som klasser i hierarkiet i stedet for som generelle restklasser.

## Den aksiomatiske model af sprogbrugere

Vi har foreslået en aksiomatisk model for menneskelig kommunikation hvor en sprogbruger er i stand til at sammenligne den relative velformethed af to vilkårlige samlinger af lingvistiske analyser. Givet et enkelt, lingvistisk plausibelt sæt af antagelser om hvordan sprogbrugeren sammenligner disse samlinger af analyser, har vi bevist at sprogbrugerens underliggende præferenceordning opfylder nytteteoriens aksiomer og derfor kan udtrykkes vha. en nyttefunktion. Dette bevis har vigtige teoretiske konsekvenser fordi det viser at hvis vores aksiomatiske model er en god idealisering af sprogbrugeres adfærd, så kan menneskelige grammatikker indkodes vha. nyttefunktioner, med nyttefunktioner defineret vha. sandsynlighedsmodeller som et særligt interessant specialtilfælde. Kvantitative modeller af menneskeligt sprog er således ikke blot nødvendige fra et teknologisk synspunkt, men bør også spille en central rolle i teoretisk lingvistik.

## Svagheder

Der er en række svagheder i denne afhandling som eksterne begrænsninger har forhindret mig i at udbedre. Den mest alvorlige svaghed er manglen på en maskinel implementering som kan bruges til at evaluere teorien med hensyn til dens lingvistiske og datalogiske egenskaber; i stedet har jeg kun lavet en detaljeret skitse for den maskinelle implementering og skrevet de første 27.000 liniers kildekode i DTAG-programmet. Selv om lingvistiske teorier bør kunne eksistere en vis tid uden maskinel implementering og evaluering, håber jeg engang at kunne leve op til mine egne idealer i denne henseende.

Den teori som afhandlingen beskriver, har andre, mindre alvorlige svagheder. Den indeholder ikke en teori for anaforer, men burde specificere i detaljer hvordan anaforer skal analyses og håndteres i leksikonnet, parseren og den probabilistiske sprogmodel. Den probabilistiske sprogmodel i kapitel 6 kan forbedres på mange måder, som skitseret i slutningen af kapitlet. DG kan også med fordel sammenlignes grundigt med andre teorier i lingvistik, datalingvistik og psykolingvistik, både med hensyn til syntaksformalismen, parsing- og indlæringsalgoritmerne, de lingvistiske analyser af konkrete fænomener og forklaringsmodellerne for de psykolingvistiske observationer af menneskelig sprogprocessering.

## Vejen frem

Der er mange emner som ikke er blevet berørt i denne afhandling. Men den syntaksformalisme vi har foreslået, ser ud til at besidde nogle af de ultimative egenskaber som vi opstillede i begyndelsen af resumeet: (a) DG-formalismen lader til at være en brugbar ramme for den syntaktiske beskrivelse af menneskelige sprog; (b) den lader til at være en brugbar ramme til at formulere lingvistisk adækvate probablistiske sprogmodeller; (c) den kan kobles med en parsingalgoritme med næsten-lineær tidskompleksitet; og (d) parsingmodellen synes at være kompatibel med i det mindste nogle af de psykolingvistiske observationer af menneskelig parsing. Vi tror derfor at den syntaksformalisme vi har foreslået i denne afhandling, kan være et skridt i den rigtige retning som fortjener yderligere udforskning.

# Bibliography

A. Abeillé and O. Rambow, editors. *Tree Adjoining Grammars. Formalisms, linguistic analysis, and processing*. CSLI Publications, 2000.

A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data structures and algorithms*. Addison-Wesley, 1987.

J. S. Allwood. The Complex NP Constraint in Swedish. University of Massachusetts Occasional Reports No. 2. University of Massachusetts, Dept. of Linguistics, 1976.

S. Aoshima, C. Phillips, and A. Weinberg. Theoretical implications of the parsing of Japanese wh-scrambling constructions. In G. Garding and M. Tsujimura, editors, *Proceedings of the 22nd West Coast Conference on Formal Linguistics*. Cascadilla Press, 2003.

N. Asher. *Reference to abstract objects in discourse*. Kluwer Academic Publishers, 1993.

N. Asher. Discourse structure and the logic of conversation. Current research in the semantics-pragmatics interface 1, Elsevier, 1999.

N. Asher. Computation and storage in discourse interpretation. In S. Nooteboom, editor, *Computation and storage in linguistics*. Kluwer Academic Publishers, forthcoming.

M. Bader and I. Lasser. German verb-final clauses and sentence processing: Evidence for immediate attachment. In Clifton et al. (1994).

J. Baldridge and A. Lascarides. Annotating discourse structure for robust semantic interpretation. In *Proceedings of the Sixth International Workshop on Computational Semantics, Tilburg, The Netherlands*, 2005.

A. Barron, J. Rissanen, and B. Yu. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44 (6), October 1998.

A. Basu, S. Ray, C. Park, and S. Basu. Improved power in multinomial goodness-of-fit tests. *Journal of the Royal Statistical Society Series D*, 51(3): 381–393, September 2002.

G. Bech. *Studien über das deutsche Verbum infinitum.* Number 139 in Linguistische Arbeiten. Niemeyer, Tübingen, 2nd edition, 1955/1983.

A. Belz. PCFG learning by nonterminal partition search. In P. Adriaans, H. Fernau, and M. van Zaanen, editors, *Grammatical inference: algorithms and applications. Proceedings of the 6th International Colloquium on Grammatical Inference (ICGI 2002)*, pages 14–27. Springer-Verlag, 2002.

T. G. Bever. The cognitive basis for linguistic structures. In J. R. Hayes, editor, *Cognition and the development of language*. Wiley, 1970.

E. Bick. A CG & PSG hybrid approach to automatic corpus annotation. In K. Simow and P. Osenova, editors, *Proceedings of SProLaC2003 (at Corpus Linguistics 2003, Lancaster)*, pages 1–12, 2003.

N. L. Biggs. *Discrete mathematics*. Oxford Science Publications, 1989.

R. Bod. Enriching linguistics with statistics: Performance models of natural language. ILLC Dissertation Series 1995-14, University of Amsterdam, The Netherlands, 1995.

R. Bod. Monte Carlo parsing. In Bunt and Tomita (1996), pages 255–280.

R. Bod. *Beyond grammar. An experience-based theory of language*. CSLI Publications, 1998.

R. Bod and R. Kaplan. A data-oriented parsing model for Lexical-Functional Grammar. In R. Bod, R. Scha, and K. Sima'an, editors, *Data-oriented parsing*. CSLI Publications, 2003.

A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. The Prague Dependency Treebank: three-level annotation scenario. In A. Abeillé, editor, *Treebanks: Building and using syntactically annotated corpora*. Kluwer Academic Publishers, 2001.

R. D. Borsley. *Syntactic theory*. Edward Arnold, 1991.

R. D. Borsley. *Modern phrase structure grammar*. Blackwell, 1996.

G. Bouma, R. Malouf, and I. Sag. Satisfying constraints on extraction and adjunction. *Natural Language and Linguistic Theory*, 2001.

S. Brants, S. Dipper, S. Hansen, W. Lezius, and G. Smith. The TIGER treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories, Sozopol*, 2002.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Wadsworth, 1984.

J. Bresnan. *Lexical-functional syntax*. Blackwell, 2001.

C. Brew. Stochastic HPSG. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, pages 83–89, 1995.

T. Briscoe, V. de Paiva, and A. Copestake, editors. *Inheritance, defaults, and the lexicon*. Cambridge University Press, 1993.

E. H. P. Brown. Report of the Oxford meeting, September 25-29, 1936. *Econometrica*, 5(4):361–383, October 1937.

T. Buch-Larsen, J. P. Nielsen, M. Guillen, and C. Bolancé. Kernel density estimation for heavy-tailed distributions using the Champernowne transformation. *Statistics*, 39(6):503–518, December 2005.

H. Bunt and M. Tomita, editors. *Recent advances in parsing technology*. Kluwer Academic Publishers, 1996.

L. Carlson and D. Marcu. Discourse tagging manual. ISI Tech Report ISI-TR-545, Information Sciences Institute, University of Southern California, July 2001.

L. Carlson, D. Marcu, and M. E. Okurowski. Building a discourse-tagged corpus in the framework of Rhetorical Structure Theory. In J. van Kuppevelt and R. Smith, editors, *Current directions in discourse and dialogue*, pages 85–112. Kluwer Academic Publishers, 2003.

D. G. Champernowne. The graduation of income distributions. *Econometrica*, 20(4):591–615, October 1952.

E. Charniak. A maximum-entropy-inspired parser. In *Proceedings of NAACL*, 2000.

N. Chomsky. *Syntactic structures*. Mouton, The Hague, 1957.

N. Chomsky. *Aspects of the theory of syntax*. MIT Press, 1965.

N. Chomsky. *Some consequences of the theory of Government and Binding*. MIT Press, Cambridge, Mass., 1982.

N. Chomsky and M. Halle. *The sound pattern of English*. Harper and Row, 1968.

P. Cimiano and S. Handschuh. Ontology-based linguistic annotation. In *Proceedings of the ACL 2003 Workshop on Linguistic Annotation: Getting the Model Right*, 2003.

C. Clifton, L. Frazier, and K. Rayner, editors. *Perspectives on sentence processing*. Lawrence Earlbaum Associates, 1994.

M. Collins. Three generative, lexicalized models for statistical parsing. In P. R. Cohen and W. Wahlster, editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23, Somerset, New Jersey, 1997. Association for Computational Linguistics.

F. Costa, P. Frasconi, V. Lombardo, and G. Soda. Towards incremental parsing of natural language using recursive neural networks. *Applied Intelligence*, 19(1–2):9–25, 2003.

P. W. Culicover and P. M. Postal. *Parasitic gaps*. MIT Press, 2001.

M. Dalrymple, R. M. Kaplan, J. T. Maxwell III, and A. Zaenen, editors. *Formal issues in Lexical-Functional Grammar*. CSLI Lecture Notes, no. 47, 1994.

M. Daniels and D. Meurers. Improving the efficiency of parsing with discontinuous constituents. In S. Wintner, editor, *Proceedings of NLULP'02: The 7th International Workshop on Natural Language Understanding and Logic Programming*, pages 49–68, 2002.

M. Daum. Dynamic dependency parsing. In *Proc. of ACL 2004 Workshop on Incremental Parsing, Barcelona, Spain*, 2004.

M. Daum and W. Menzel. Parsing natural language using guided local search. In F. van Harmelen, editor, *Proceedings 15th European Conference on Artificial Intelligence, Lyon, France, 2002*, pages 435–439, 2002.

A. Dikovsky. Grammars for local and long dependencies. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, 2001.

D. Duchier. Lexicalized syntax and topology for non-projective dependency grammar. In *Joint Conference on Formal Grammars and Mathematics of Language FGMOL-01, Helsinki, August 2001*, 2001.

D. Duchier and R. Debusmann. Topological dependency trees: A constraint-based account of linear precedence. In *39th Annual Meeting of the Association for Computational Linguistics (ACL 2001), Toulouse, France*, July 9-11 2001.

J. M. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING-96*, pages 340–345, 1996.

E. Engdahl. Restrictions on unbounded dependencies in Swedish. In E. Engdahl and E. Ejerhed, editors, *Readings on unbounded dependencies in Scandinavian languages*. Almquist and Wiksell, Stockholm, 1982.

E. Engdahl. Parasitic gaps. *Linguistic Inquiry*, 6(1):5–34, 1983.

N. Fabb. Compounding. In Spencer and Zwicky (1998).

C. Fellbaum, editor. *WordNet: An electronic lexical database*. MIT Press, 1998.

F. Ferreira and J. M. Henderson. Use of verb information in syntactic parsing: Evidence from eye movements and word-by-word self-paced reading. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 16:555–568, 1990.

J. D. Fodor and F. Ferreira, editors. *Reanalysis in sentence processing*. Kluwer Academic Publishers, 1998.

K. Foth, M. Daum, and W. Menzel. A broad coverage parser for German based on defeasible constraints. In H. Christiansen, P. R. Skadhauge, and J. Villadsen, editors, *Constraint Solving and Language Processing, Workshop Proceedings, Datalogiske Skrifter No. 99*, pages 88–101. Roskilde Universitetscenter, Denmark, 2004.

L. Frazier and K. Rayner. Making and correcting errors during sentence comprehension: Eye movements in the analysis of structurally ambiguous sentences. *Cognitive Psychology*, 14:178–210, 1982.

R. Freidin, editor. *Principles and parameters in comparative grammar*. MIT Press, 1991.

J. H. Friedman and N. I. Fisher. Bump hunting in high-dimensional data. *Statistics and Computing*, 9(2):123–143, 1999. ISSN 0960-3174.

G. Gazdar. Unbounded dependencies and coordinate structure. *Linguistic Inquiry*, 12:155–184, 1981.

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

K. Gerdes and S. Kahane. A formal dependency grammar using a topological hierarchy. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, 2001.

M. A. Gernsbacher, editor. *Handbook of psycholinguistics*. Academic Press, 1994.

L. Haegeman. *Introduction to Government and Binding theory*. Blackwell, 2nd edition, 1994/1991.

P. Halmos. *Measure theory*. Springer-Verlag, 1974/1950.

K. Harbusch and G. Kempen. A quantitative model of word order and movement in English, Dutch and German complement constructions. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING-2002), Taipei (Taiwan)*, pages 328–334. Morgan Kaufmann, 2002.

K. Hartmann. *Right node raising and gapping: interface conditions on prosodic deletion*. John Benjamins, 2000.

G. Helbig. *Probleme der Valenz- und Kasustheorie*. Max Niemeyer Verlag, 1992.

G. Helbig and W. Schenkel. *Wörterbuch zur Valenz und Distribution deutscher Verben*. Max Niemeyer Verlag, 1971/1969.

P. Hellwig. Dependency unification grammar. In V. Agel, L. M. Eichinger, H. Eroms, P. Hellwig, H. Heringer, and H. Lobin, editors, *Dependency and valency. An international handbook of contemporary research*. Mouton, 2003.

L. Heltoft and E. Hansen. Grammatik over det danske sprog, kapitel 1. Udkast. Institut for Sprog og Kultur, Roskilde Universitetscenter, 2000.

I. N. Herstein and J. Milnor. An axiomatic approach to measurable utility. *Econometrica, Journal of the Econometric Society*, 21(2), April 1953.

R. L. Hill and W. S. Murray. Commas and spaces: Effects of punctuation on eye movements and sentence parsing. In A. Kennedy, R. Radach, D. Heller, and J. Pynte, editors, *Reading as a perceptual process*. Elsevier Science Ltd., 2000.

C. Hockett. *A manual of phonology*. Indiana University publications in anthropology and linguistics, Memoir 11. Waverley Press, 1955.

C. Hockett. Two models of grammatical description. In M. Joos, editor, *Readings in linguistics*, pages 386–99. University of Chicago Press, 2nd edition, 1958.

J.-M. Hopf, M. Bader, M. Meng, and J. Bayer. Is human sentence parsing serial or parallel? Evidence from event-related brain potentials. *Cognitive Brain Research*, 15:165–177, 2003.

R. Hudson. *English Word Grammar*. Blackwell, 1990.

R. Hudson. Syntax without functional categories. UCL Working Papers in Linguistics 9, 1997.

R. Hudson. Trouble on the left periphery. Unpublished manuscript, 1998.

R. Hudson. An encyclopedia of English grammar and Word Grammar. http://www.phon.ucl.ac.uk/home/dick/enc-gen.htm, 2003.

R. Hudson. Are determiners heads? *Functions of Language*, 11(1), 2004.

D. S. Johnson and L. A. McGeoch. The Traveling Salesman Problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local search in combinatorial optimization*, pages 215–310. John-Wiley and Sons, Ltd., 1997.

K. Johnson. Restoring exotic coordinations to normalcy. *Linguistic Inquiry*, 33(1):97–156, 2002.

M. Johnson. Learning and parsing stochastic unification-based grammars. In B. Scholkopf, editor, *Learning theory and kernel machines*. Springer, 2003.

B. Jones. *What's the point? A (computational) theory of punctuation*. PhD thesis, Centre for Cognitive Science, University of Edinburgh, 1997.

D. Jurafsky and J. H. Martin. *Speech and language processing*. Prentice-Hall, 2000.

R. Kager. *Optimality Theory*. Cambridge University Press, 1999.

S. Kahane. Bubble trees and syntactic representations. In Becker and Krieger, editors, *Proc. 5th M. of Mathematics of Language*. DFKI, 1997.

Y. Kamide and D. C. Mitchell. Incremental pre-head attachment in Japanese parsing. *Language and Cognitive Processes*, 14:631–662, 1999.

H. Kamp and U. Reyle. *From discourse to logic*. Kluwer Academic publishers, 1993.

E. R. Kandel, J. H. Schwartz, and T. M. Jessell, editors. *Principles of neural science*. Appleton & Lange, 3rd edition, 1991.

F. Keller. *Gradience in grammar: Experimental and computational aspects of degrees of grammaticality*. PhD thesis, University of Edinburgh, 2000.

G. Kempen and K. Harbusch. Performance grammar: A declarative definition. In A. Nijholt, M. Theune, and H. Hondorp, editors, *Computational linguistics in the Netherlands 2001*, pages 148–162. Rodopi, 2002.

B. Keson and O. Norling-Christensen. PAROLE-DK. Det Danske Sprog- og Litteraturselskab, 1998.

D. Klein and C. D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL 2004)*, pages 479–486, 2004.

L. Konieczny. *Human sentence processing: A semantics-oriented parsing approach*. PhD thesis, Albert-Ludwigs-Universität zu Freiburg i. Br., 1996.

M. T. Kromann. Towards Discontinuous Grammar. In A. Todirascu, editor, *Proceedings of the ESSLLI 1999 Student Session*, pages 145–156, August 1999a.

M. T. Kromann. Towards discontinuous grammar formalisms with lexical notions of valency. Master's thesis. Department of Computational Linguistics, Copenhagen Business School, January 1999b.

M. T. Kromann. Optimality parsing and local cost functions in Discontinuous Grammar. In *Proceedings of the Joint Conference on Formal Grammar and Mathematics of Language (FGMOL-01), Helsinki, August 10-12, 2001. Electronic Notes in Theoretical Computer Science 53*, 2001.

M. T. Kromann. Elliptic VP coordinations in Danish 1. Survey T1 on Linguist-GRID.org. http://linguist-grid.org/cgi-bin/grid?view=report&test=1, 2002a.

M. T. Kromann. Linguist-GRID.org. Internet website for linguistic surveys. http://linguist-grid.org, 2002b.

M. T. Kromann. The Danish Dependency Treebank and the DTAG treebank tool. In J. Nivre and E. Hinrichs, editors, *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT 2003)*. Växjö University Press, 2003.

M. T. Kromann. Learning massively probabilistic grammars from treebanks using hierarchical partition models. *Acta Linguistica Hafniensia*, 36, 2004.

M. T. Kromann. DTAG treebank tool. http://www.id.cbs.dk/~mtk/dtag, 2005.

M. T. Kromann and S. K. Lynge. Danish Dependency Treebank v. 1.0. Department of Computational Linguistics, Copenhagen Business School, 2004.

M. T. Kromann, L. Mikkelsen, and S. K. Lynge. Danish Dependency Treebank. Annotation manual. Department of Computational Linguistics, Copenhagen Business School, 2003.

E. König, W. Lezius, and H. Voormann. TIGERSearch 2.1 user's manual. IMS, Univ. of Stuttgart, 2003.

A. Lascarides and N. Asher. Segmented Discourse Representation Theory: Dynamic semantics with discourse structure. In H. Bunt and R. Muskens, editors, *Computing meaning: Volume 3*. Kluwer Academic Publishers, in press.

A. Lavie and M. Tomita. GLR* — An efficient noise-skipping parsing algorithm for context-free grammars. In Bunt and Tomita (1996), pages 183–200.

R. L. Lewis. Reanalysis and limited repair parsing: Leaping off the garden path. In Fodor and Ferreira (1998).

R. L. Lewis. Specifying architectures for language processing: Process, control, and memory in parsing and interpretation. In M. Crocker, M. Pickering, and C. Clifton, editors, *Architectures and mechanisms for language processing*. Cambridge University Press, 1999.

R. L. Lewis. Falsifying serial and parallel parsing models: Empirical conundrums and an overlooked paradigm. *The Journal of Psycholinguistic Research*, 29:241–248, 2000.

H. Li and N. Abe. Generalizing case frames using a thesaurus and the MDL principle. *Computational Linguistics*, 24(2):217–244, 1998.

H. Li and N. Abe. Learning dependencies between case frame slots. *Computational Linguistics*, 25(2):283–291, 1999.

R. Lieber. *On the organization of the lexicon*. PhD thesis, Massachusetts Institute of Technology, 1980.

S. Lin. Computer solutions of the Travelling Salesman Problem. *Bell System Technical Journal*, 44(10):2245–69, December 1965.

V. Lombardo and L. Lesmo. Unit coordination and gapping in dependency theory. In A. Polguère and S. Kahane, editors, *Proceedings of the workshop "Processing of Dependency-Based Grammars", COLING-ACL 1998*, 1998.

W. C. Mann and S. A. Thompson. Rhetorical Structure Theory: Towards a functional theory of text organization. *Text*, 8(3):243–281, 1988.

C. D. Manning. Probabilistic syntax. In R. Bod, J. Hay, and S. Jannedy, editors, *Probabilistic linguistics*, pages 289–341. MIT Press, 2003.

C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

D. Marcu. *The theory and practice of discourse parsing and summarization*. MIT Press, 2000.

D. Marcu. Discourse structures: trees or graphs? Web comment, http://www.isi.edu/∼marcu/discourse, 2003.

M. Marcus. *A theory of syntactic recognition for natural language*. MIT Press, 1980.

M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The Penn Treebank: Annotating predicate argument structure. In *ARPA Human Language Technology Workshop*, 1994.

P. H. Matthews. *Morphology*. Cambridge University Press, 2nd edition, 1991.

J. T. Maxwell and R. M. Kaplan. The interface between phrasal and functional constraints. *Computational Intelligence*, 19(4):571–590, 1993.

A. Mecklinger, H. Schriefers, K. Steinhauer, and A. Friederici. Processing relative clauses varying on syntactic and semantic dimensions: An analysis with event-related potentials. *Memory and Cognition*, 23:477–494, 1995.

I. A. Melcuk. *Dependency syntax: Theory and practice*. State University of New York Press, 1988.

P. Merlo and S. Stevenson. Lexical syntax and parsing architecture. In

M. Crocker, M. Pickering, and C. Clifton, editors, *Architectures and mechanisms for language processing*, pages 161–188. Cambridge University Press, 2000.

M. Meteer, A. Taylor, R. MacIntyre, and R. Iyer. Dysfluency annotation stylebook for the Switchboard corpus. Distributed by LDC, 1995.

J. Michaelis and C. Wartena. How linguistic constraints on movement conspire to yield languages analyzable with a restricted form of LIGs. In G. M. Kruijff, G. V. Morrill, and R. T. Oehrle, editors, *Proceedings of Formal Grammar 1997. Linguistic aspects of logical and computational perspectives on language*, pages 158–168, 1997.

J. Michaelis and C. Wartena. Unidirectional inheritance of indices. A weakly context free facet of LIGs. In G. Bouma, G. M. Kruijff, and R. T. Oehrle, editors, *Proceedings of the FHCG'98*, pages 258–267, 1998.

G. Miller. The magical number seven, plus or minus two. *The Psychological Review*, 63(1):81–97, 1956.

E. T. Miyamoto, E. Gibson, N. J. Pearlmutter, T. Aikawa, and S. Miyagawa. A U-shaped relative clause attachment preference in Japanese. *Language and Cognitive Processes*, 14(5/6):663–686, 1999.

J. R. Munkres. *Topology. A first course*. Prentice Hall, 1975.

S. Müller. Continuous or discontinuous constituents? A comparison between syntactic analyses for constituent order and their processing systems. *Research on Language and Computation*, 2:209–257, 2004.

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

J. Nerbonne, K. Netter, and C. Pollard, editors. *German in Head-driven Phrase Structure Grammar*. University of Chicago Press, 1994.

P. Neuhaus and N. Bröker. The complexity of recognition of linguistically adequate dependency grammars. In P. R. Cohen and W. Wahlster, editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 337–343, Somerset, New Jersey, 1997. Association for Computational Linguistics.

J. Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03), Nancy, France, 23-25 April 2003*, pages 149–160, 2003.

J. Nivre. Incrementality in deterministic dependency parsing. In *Incremental parsing: bringing engineering and cognition together. Workshop at ACL-*

*2004, Barcelona, Spain, July 25, 2004*, 2004.

J. Nivre, J. Hall, and J. Nilsson. Memory-based dependency parsing. In H. T. Ng and E. Riloff, editors, *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL), May 6-7, 2004, Boston, Massachusetts*, pages 49–56, 2004.

J. Nivre and J. Nilsson. Pseudo-projective dependency parsing. In *Proceedings of ACL 2005*, 2005.

J. Nivre and M. Scholz. Deterministic dependency parsing of English text. In *Proceedings of COLING 2004, Geneva, Switzerland, August 23-27, 2004*, 2004.

R. Norberg. Credibility theory. In J. Teugels and B. Sundt, editors, *Encyclopedia of actuarial science*. Wiley, 2004.

G. Nunberg. *The linguistics of punctuation*. CSLI Publications, 1990.

C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization. Algorithms and complexity*. Dover Press, 1982.

P. Parikh. Communication, meaning, and interpretation. *Linguistics and Philosophy*, 23:185–212, 2000.

P. Parikh. *The use of language*. CSLI Publications, 2001.

A. R. Parker. Parasitic gaps in the Germanic languages. Master's thesis, Department of Linguistics, National University of Ireland, Dublin, 1999.

N. J. Pearlmutter and A. A. Mendelsohn. Serial versus parallel sentence comprehension. Manuscript in revision, 2000.

C. Phillips and E. Lau. Foundational issues (review article on Jackendoff's Foundations of language, 2002). *Journal of Linguistics*, 40:571–591, 2004.

L. Polanyi. The linguistic structure of discourse. In D. Schiffrin, D. Tannen, and H. E. Hamilton, editors, *Handbook of discourse analysis*, Blackwell Handbooks in Linguistics. Blackwell Publishers, 2003.

L. Polanyi, C. Culy, M. H. van den Berg, G. L. Thione, and D. Ahn. Sentential structure and discourse parsing. In *Proceedings of the ACL2004 Workshop on Discourse Annotation, Barcelona, Spain, July 25-26, 2004*, 2004.

L. Polanyi, M. H. van den Berg, and D. Ahn. Discourse structure and sentential information structure. An initial proposal. *Journal of Logic, Language and Information*, 2003.

C. Pollard and I. A. Sag. *Head-driven Phrase Structure Grammar*. University of Chicago Press, 1994.

K. R. Popper. *Logik der Forschung*. J.C.B. Mohr (Paul Siebeck), 1935/1982.

C. Potts. Parasitic gaps in the wild. Language Log, posted March 9. http://itre.cis.upenn.edu/~myl/languagelog/archives/000561.html, 2004.

D. M. W. Powers. Applications and explanations of Zipf's law. In *NeM-LaP3/CoNLL98: New methods in language processing and computational natural language learning*, ACL, pages 151–160, 1998.

A. Prince and P. Smolensky. Optimality Theory. Constraint interaction in generative grammar. Technical Report RuCCS TR-2, Rutgers Center for Cognitive Science, Rutgers University, 1993.

J. Pustejovsky. *The generative lexicon*. MIT Press, 1995.

J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

O. Rambow and A. K. Joshi. A processing model for free word order languages. In Clifton et al. (1994), pages 267–301.

F. P. Ramsey. Truth and probability. In R. B. Braithwaite, editor, *The foundations of mathematics and other logical essays*. Harcourt Brace Jovanovich, 1931.

M. Reape. Domain union and word order variation in German. In Nerbonne et al. (1994), pages 151–197.

J. A. Rice. *Mathematical statistics and data analysis*. Wadsworth & Brooks/Cole Advanced Books & Software, 1988.

J. Rissanen. Modeling by shortest data description. *Automatic*, 14:37–38, 1978.

B. Roark. Robust garden path parsing. *Natural Language Engineering*, 9(2): 1–24, 2003.

J. R. Ross. *Constraints on variables in syntax*. PhD thesis, MIT, 1967.

S. J. Russell and P. Norvig. *Artificial intelligence*. Prentice-Hall, 1995.

I. Sag, G. Gazdar, T. Wasow, and S. Weisler. Coordination and how to distinguish categories. *Natural Language and Linguistic Theory*, 3:117–171, 1985.

G. Sampson. Consistent annotation of speech-repair structures. In A. R. et al, editor, *Proceedings of the First International Conference on Language Resources and Evaluation, Granada*, pages 1279–82, 1998.

C. Samuelsson. Relating Turing's formula and Zipf's law. In *Proceedings of the 4th Workshop on Very Large Corpora*, Copenhagen, Denmark, 1996.

E. Sapir. *Language: An introduction to the study of speech*. Harcourt Brace, New York, 1921.

A. Sarkar and A. Joshi. Handling coordination in a Tree Adjoining Grammar. In *Proceedings of COLING 1996*, pages 610–615, 1996.

B. Say and V. Akman. Current approaches to punctuation in computational linguistics. *Computers and the Humanities*, 30(6):457–469, 1997.

P. Schachter. Constraints on coordination. *Language*, 53:86–103, 1977.

G. Schneider. Extracting and using trace-free functional dependencies from the Penn Treebank to reduce parsing complexity. In J. Nivre, editor, *Proceedings of TLT 2003, Växjö University*, 2003.

G. Schneider, J. Dowdall, and F. Rinaldi. A robust and hybrid deep-linguistic theory applied to large-scale parsing. In *COLING 2004, Robust methods in analysis of natural language data, Geneva, Switzerland, August 29*, 2004.

M. Schulz. Parsen natürlicher Sprache mit gesteuerter lokaler Suche. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, 2000.

K. Sima'an. Computational complexity of probabilistic disambiguation by means of tree grammars. In *Proceedings of COLING 96*, volume 2 of *COLING*, pages 1175–1180, 1996.

H. L. Somers. *Valency and case in computational linguistics*. Edinburgh University Press, 1987.

A. Sorace and F. Keller. Gradience in linguistic data. *Lingua*, 115(11):1497–1524, 2005.

A. Spencer. Morphophonological operations. In Spencer and Zwicky (1998).

A. Spencer and A. M. Zwicky, editors. *The handbook of morphology*. Blackwell Publishers, 1998.

S. Staab. GLR parsing of word lattices using a beam search method. In *Eurospeech-95: Proceedings of the European Conference on Speech Communication and Technology*, September 1995.

M. Steedman. On the generality of the nested dependency constraint and the reason for an exception in Dutch. In B. Butterworth, B. Comrie, and O. Dahl, editors, *Explanations for language universals*. Mouton, 1984.

M. Steedman. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 5:403–439, 1987.

M. Steedman. Gapping as constituent coordination. *Linguistics and Philosophy*, 13:207–264, 1990.

M. Steedman. *The syntactic process*. A Bradford Book, The MIT Press, 2000.

S. Stevenson. Competition and recency in a hybrid network model of syntactic disambiguation. *Journal of Psycholinguistic Research*, 23(4):295–322, 1994.

S. Stevenson. Parsing as incremental restructuring. In Fodor and Ferreira (1998), pages 327–363.

S. Stevenson and P. Smolensky. Optimality in sentence processing. In P. Smolensky and G. Legendre, editors, *The harmonic mind: From neural computation to optimality-theoretic grammar*, chapter 19. MIT Press, 2005.

A. Stolcke. *Bayesian learning of probabilistic language models*. PhD thesis, Dept. of Electrical Engineering and Computer Science, University of California at Berkeley, 1994.

P. Sturt, F. Costa, V. Lombardo, and P. Frasconi. Learning first-pass structural attachment preferences with dynamic grammars and recursive neural networks. *Cognition*, 88(2):133–169, 2003.

P. Sturt and M. W. Crocker. Generalized monotonicity for reanalysis models. In Fodor and Ferreira (1998), pages 365–400.

P. Tapanainen. *Parsing in two frameworks: Finite-state and functional dependency grammar*. PhD thesis, University of Helsinki, Language Technology, Department of General Linguistics, Faculty of Arts, 1999.

B. Tesar. *Computational Optimality Theory*. PhD thesis, University of Colorado, 1995. Rutgers Optimality Archive 90-0000.

L. Tesnière. *Esquisse d'une syntaxe structurale*. Klincksieck, Paris, 1953.

M. Trautwein. The complexity of structure-sharing in unification-based grammars. In T. Andernach, M. Moll, and A. Nijholt, editors, *CLIN V: Papers from the fifth CLIN meeting*, pages 165–180, 1994.

J. C. Trueswell and M. K. Tanenhaus. Toward a lexicalist framework for constraint-based syntactic ambiguity resolution. In Clifton et al. (1994), pages 155–179.

H. Vater. On the possibility of distinguishing between complements and adjuncts. In W. Abraham, editor, *Valence, semantic case and grammatical relations*, pages 21–45. John Benjamins, Amsterdam, 1978.

J. von Neumann and O. Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1944.

T. Vosse and G. Kempen. Syntactic structure assembly in human parsing: A computational model based on competitive inhibition and a lexicalist grammar. *Cognition*, 75:105–143, 2000.

H. T. Wareham. *Systematic parameterized complexity analysis in computational* *phonology*. PhD thesis, University of Victoria, 1998. Rutgers Optimality Archive 318-0599.

B. Webber. DLTAG: Extending lexicalized TAG to discourse. *Cognitive Science*, 28:751–779, 2004.

B. Webber, A. Knott, M. Stone, and A. Joshi. Discourse relations: A structural and presuppositional account using Lexicalised TAG. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, College Park MD*, June 1999.

B. Webber, M. Stone, A. Joshi, and A. Knott. Anaphora and discourse structure. *Computational Linguistics*, 29(4):545–587, 2003.

R. Wiese. Phonological vs. morphological rules: On German umlaut und ablaut. *Journal of Linguistics*, 32:113–35, 1996.

F. Wolf and E. Gibson. The descriptive inadequacy of trees for representing discourse coherence. Unpublished manuscript, 2003.

H. Yamada and Y. Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of 8th International Workshop of Parsing Technologies*, April 2003.

# Index