Decomposing and Regenerating Syntactic Trees

Federico Sangati

Decomposing and Regenerating Syntactic Trees

ILLC Dissertation Series DS-2012-01

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 904
1098 XH Amsterdam
phone: +31-20-525 6051
fax: +31-20-525 5206
e-mail: illc@uva.nl
homepage: http://www.illc.uva.nl/

Decomposing and Regenerating Syntactic Trees

Academisch Proefschrift

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. D. C. van den Boom
ten overstaan van een door het college voor
promoties ingestelde commissie, in het openbaar
te verdedigen in de Agnietenkapel
op donderdag 12 januari 2012, te 10.00 uur

door

Federico Sangati

geboren te Abano Terme, Italië.

Promotiecommissie:

Promotor:
prof. dr. L.W.M. Bod
Co-promotor:
dr. W.H. Zuidema

Overige leden:
prof. dr. P.W. Adriaans
dr. T. Cohn
prof. dr. S. Kahane
prof. dr. R.J.H. Scha
dr. K. Sima'an

Faculteit der Geesteswetenschappen
Universiteit van Amsterdam

The research reported in this thesis was supported through a Vici-grant "Integrating Cognition" (nr. 277.70.006) to Rens Bod by the Netherlands Organization for Scientific Research (NWO).

Printed and bound by Ipskamp Drukkers

ISBN: 978–90–5776–234–5

all'Italia del dopo Berlusconi
perché si possa finalmente risvegliare dal lungo sonno

Contents

Acknowledgments **xi**

1 Introduction **1**

1.1 Learning Language Structures 2

1.1.1 The hidden structure of language 2

1.1.2 Different perspectives on language 3

1.2 Syntactic structures of language 4

1.2.1 Syntactic representation and generative processes 4

1.2.2 Different representations 5

1.2.3 Phrase-Structure 5

1.2.4 Dependency-Structure 6

1.2.5 Relations between PS and DS 7

1.3 Generative models of syntactic structures 7

1.3.1 Context-Free Grammars 8

1.3.2 Generalized models 8

1.3.3 Probabilistic generative models 10

1.4 Computational models of syntax 11

1.5 Thesis overview 14

2 Generalized Tree-Generating Grammars **17**

2.1 Introduction 18

2.1.1 Symbolic and Probabilistic models 18

2.1.2 Tree structures 20

2.2 Symbolic Generative Models for Trees 22

2.2.1 The event space 22

2.2.2 The conditioning context 23

2.2.3 Context-Free Grammar 24

2.2.4 The generative process 25

2.2.5 Extracting a symbolic grammar from a treebank 30

2.2.6 Examples of generative tree grammars 30
2.3 Probabilistic Generative Models for Trees 38
2.3.1 Resolving the syntactic ambiguity 38
2.3.2 The probability of a tree 39
2.3.3 Estimating Probability Distributions 39
2.4 Parsing through Reranking 43
2.5 Discriminative and Generative models 44
2.6 Conclusions 47

3 Recycling Phrase-Structure Constructions 49
3.1 Introduction to Phrase-Structure 50
3.2 Review of existing PS models 51
3.2.1 Head-driven models 51
3.2.2 State-Splitting Models 53
3.3 Data-Oriented Parsing 54
3.3.1 Introduction 55
3.4 The symbolic backbone 56
3.4.1 Explicit vs. Implicit Grammars 56
3.5 Finding Recurring Fragments 58
3.5.1 The search algorithm 59
3.5.2 A case study on the Penn WSJ 60
3.6 The probability model 66
3.6.1 Parsing 67
3.6.2 Inducing probability distributions 69
3.6.3 Maximizing Objectives 70
3.7 Implementation 73
3.8 Annotated Resources 75
3.9 Evaluation Metrics 76
3.10 Results 76
3.11 Conclusions 83
3.11.1 Future Directions 83
3.11.2 Next steps 84

4 Learning Dependency-Structures 85
4.1 Introduction 86
4.2 Dependency-Structure 87
4.3 Comparing PS with DS 88
4.3.1 Structural relations between PS and DS 88
4.3.2 Relations between PS and DS grammars 90
4.4 Other related syntactic theories 92
4.5 Models for parsing DS 95
4.5.1 Probabilistic Generative models 95
4.5.2 Discriminative models 99

4.6 Reranking generative models 100
4.6.1 Experiment Setup 102
4.6.2 Comparing the Eisner models 102
4.6.3 A new generative model 103
4.6.4 Results 105
4.7 Conclusions 107
4.8 Future Directions 108

5 Tesnière Dependency-Structure 109
5.1 Introduction 110
5.2 Dependency-Structures à la Tesnière 111
5.2.1 The dependency relation 111
5.2.2 Words, blocks and categories 111
5.2.3 Junction 112
5.2.4 Transference 113
5.3 Comparing TDS with DS 114
5.3.1 Choosing the correct heads 116
5.3.2 Categories and Blocks 116
5.3.3 Coordination 117
5.4 Converting the Penn WSJ in TDS notation 118
5.4.1 Elements of a TDS 118
5.4.2 The conversion procedure 119
5.5 A probabilistic Model for TDS 121
5.5.1 Model description 121
5.5.2 Experimental Setup 123
5.5.3 Evaluation Metrics for TDS 124
5.5.4 Results 125
5.6 Other representations of the WSJ Treebank 126
5.6.1 Prague English dependency treebank 128
5.6.2 Stanford Typed Dependency Representation 128
5.6.3 Bubble Trees 131
5.6.4 The CCG-bank 131
5.7 Assessment of the converted treebank 133
5.8 Conclusion 134
5.9 Future directions 134

6 Conclusions 137

A Phrase-Structure Models 141
A.1 Models parameters 141
A.2 Evaluation procedure 142
A.3 Comparing Double-DOP and Berkeley parser 144

B Dependency-Structure Models **145**

B.1 DS to PS 145

B.2 Smoothing details 147

C TDS model **149**

C.1 Head annotation 149

C.2 Coordination 151

C.3 Smoothing in the TDS model 155

C.4 Examples of TDS trees 155

Bibliography **160**

Index **183**

Samenvatting **185**

Abstract **187**

Acknowledgments

I am very grateful to Rens Bod and Jelle Zuidema for their supervision. Their complementary roles were fundamental for the development of my research. Since the beginning of my PhD, Rens has shown lot of trust and confidence in my capabilities and has let me total freedom in exploring various research paths, yet providing me solid guidance in moments of need. This has allowed me to learn how to work independently and understand my real research interests. Jelle's presence was also indispensable throughout the whole project: he has great patience as a listener, and a remarkable ability to quickly understand a problem and formulate brilliant solutions. But above all his intellectual honesty, and his ability to relativize things within and outside academia made him one of the most relevant guiding figures in all these years.

I am thankful to a number of people who have accompanied me during the last four years. In particular the PhD members of the LaCo group: Gideon Borensztajn with whom I've shared the whole PhD adventure, including many moments of intense discussions, big struggles, exciting ping-pong challenges, and other fun activities; Markos Mylonakis great hiking companion as well as irreplaceable machine learning advisor; Gideon Maillette de Buy Wenniger for the nice discussions and his big courage in taking over and extending my GUI code; and Barend Beekhuizen for his great passion and for his invaluable help in annotating hundreds of sentences.

I am in debt with Yoav Seginer one of the most significant teachers who have inspired me during the beginning of my PhD, and accepted to go through the pre-final draft of my thesis (long after having left academia) providing extremely valuable feedbacks. Many thanks to Chiara Mazza for the fruitful collaboration started during the summer 2009 which has lead to the joint work on Tesnière Dependency Structures; and to Andreas van Cranenburgh, for the intense interaction towards the end of my project, and for the effort of proof-reading the thesis and translating the abstract into Dutch.

Special thanks to Pieter Adriaans, Remko Scha, Khalil Sima'an, and Henk

Zeevat from whom I've learned a lot since I came to Amsterdam for the MSc. In the last few years they have continued supporting my work and frequently provided me with valuable feedbacks in dedicated meetings and dry runs. Many thanks also to the external members of the PhD committee Trevor Cohn and Sylvain Kahane for their numerous comments and suggestions on the thesis.

I am grateful to other colleagues with whom I had the pleasure to build fruitful discussions in Amsterdam and abroad: Tejaswini Deoskar, Yoav Goldberg, Gerold Schneider, Djamé Seddah, and Reut Tsarfaty.

Next, I would like to express profound gratitude to my paranymphs Inés Crespo and Umberto Grandi for their assistance in completing all the procedures surrounding the end of my PhD, but above all for having represented solid figures of support in times of struggle, and great companions in several activities we have gone through the last few years.

A collective thank you to all the other colleagues at the ILLC, with whom I have shared interesting conversations, coffees, and many laughs: Stéphane Airiau, Sophie Arnoult, Cédric Dégremont, Ulle Endriss, María Esteban Garcia, Vanessa Ferdinand, Stefan Frank, Pietro Galliani, Nina Gierasimczuk, Davide Grossi, Aline Honingh, Tikitu de Jager, Yurii Khomskii, Lena Kurzen, Daniele Porello, Michael Repplinger, Raul Leal Rodriguez, Raquel Fernández Rovira, Sanchit Saraf, Mehrnoosh Sadrzadeh, Maria Spychalska, Jakub Szymanik, Joel and Sara Uckelman, Fernando Raymundo Velazquez-Quesada, and Jacob Vosmaer.

I would like to also thank the ILLC administrators for their indispensable support: Karin Gigengack, Tanja Kassenaar, Ingrid van Loon, Peter van Ormondt, and Marco Vervoort.

A big thank you to the Nuts Ultimate Frisbee teammates, for having shared lot of fun practices and games throughout the last two years.

Finally I intend to express my affection and gratitude to all those people who have continuously supported me during all these years and contributed to enrich my life: my parents, Andrea and Marco Sangati, Irene Bertazzo, Paipin Cheng, Winnie & Renée, Chiara Brachini, Daniil Umanski, Lisa Kollwelter, Margot Colinet, Pablo Seban, Giulia Soravia, Martina Deana, Ermanno Miotto, Sara Sambin, Tino Ginestri, and the Pablo Neruda group.

It is extremely difficult to give full acknowledgments to all the people who have directly or indirectly contributed to the completion of my PhD, so I apologize if someone has been accidentally omitted or hasn't been given the appropriate relevance.

Edinburgh
November, 2011.

Federico Sangati

Chapter 1

Introduction

The word of man is the most durable of all material.

Arthur Schopenhauer

1.1 Learning Language Structures

During the last decades, research in Natural Language Processing (NLP) seems to have increasingly lost contact with linguistic theory, as Mark Steedman has stated:

> "[...] while from the 1950s to the 1980s, the information theoreticians and statistical modelers among us used to make common cause with the linguists, we have subsequently drifted apart." (Steedman, 2008, p. 139)

The current thesis can be seen as a reaction to this observation: it aims at building computational models of syntax based on a number of basic linguistic theories showing that, contrary to common wisdom, their notions and concepts can be highly beneficial to NLP. This work is not meant to provide any final assessment for the validity of the theories under consideration, but should rather be seen as an attempt to formalize and understand them better. A second goal of this thesis is to contribute to the development of computer systems which aim at solving linguistic tasks, for which syntax plays an important role.

In this introductory chapter we will provide some background to the ongoing quest of discovering the hidden structures of language, and the role that computational models can play in accomplishing this goal.

1.1.1 The hidden structure of language

Language is one of the most acknowledged traits of human beings: it pervades in everyday life, and it has existed across all cultures for thousands of generations. Nevertheless, language remains one of the most controversial subjects of inquiry. Language is in fact a vague concept, hard to map to a precise entity which can be scientifically investigated. It is dynamic, as it changes in time and across linguistic communities, and there is no way to isolate it as a whole: even if we could sample all human utterances for the next 100 years we would only cover a very small fraction of all possible language productions.

Moreover, language exists in different modalities, i.e., oral, gestural, and written. Within each of these modalities, an external observer has access only to the surface manifestation of language, i.e., sound, gestures, and text. But surface information cannot fully account for what we describe as language: as in other cognitive abilities, external manifestations are only the tip of the iceberg. The structures[1] underlying language productions remain well hidden,[2] and although

[1] A language structure, in general, describes how the parts of a language production are related and built into a whole.

[2] Although there is also some dispute about the actual existence of underlying structures in language, we decided not to enter into this debate. It should suffice to say that language as other complex dynamical systems is constantly shaped by many forces, and the regularity it

regular patterns in the surface layer may provide useful clues for building hypotheses on the hidden ones, there is no precise way to verify if the linguistic analysis matches the representation used by the speaker. This observation was already made by Ferdinand de Saussure, who wrote about word categories (nouns, adjectives, etc.):

> "All these things exist in language, but as *abstract entities*; their study is difficult because we never know exactly whether or not the awareness of speakers goes as far as the analyses of the grammarian." (de Saussure, 1915, p. 190)

1.1.2 Different perspectives on language

Language can be studied using a wide variety of methodologies. In theoretical linguistics one of the most dominant modus operandi is the *introspective approach* (Tesnière 1959, p. 37; Chomsky 1984, p. 44). Under this perspective, any person who attempts to investigate language is not only seen as an external observer, but also as an active language user. He/she is therefore able to construct hypotheses on language structures based on internal intuitions, and assess how well they generalize, relying on internal judgements.

In contrast to this approach, the investigation of language has included much experimental research, whose objective is to validate hypotheses about the structure of language, based on experimental data obtained from language users performance in specific tasks, such as sentence processing (Hale, 2006; Levy, 2007; Frank and Bod, 2011), or acquisition of phonology and morphology (Boersma and Hayes, 2001; Goldwater et al., 2007), as well as from brain activations on linguistic stimuli (Bachrach, 2008).

But ultimately, both introspective and experimental approaches could provide only a partial description of the investigated phenomenon. Language, in fact, is a means of communication, and as such it is not confined to any specific organ or individual. It is a *dynamic* system whose behavior can be explained only when considering the system as a whole, including the speaking community, the communicative interactions established between its members, and the external environment they share. Fortunately, there are other approaches to language which try to give accounts for its dynamic aspects. These include fields such as language evolution (Lieberman, 1975; Zuidema, 2005), sociolinguistics (Labov, 1972; Wardhaugh, 2006), historical linguistics, and language change (Lass, 1997).

The abundance of perspectives on the study of language reflects the enormous complexity of this phenomenon. As all of these theories are small pieces of the same puzzle, it is important to develop methodologies which attempt to integrate them in order to build a basis for a unified theory. Unfortunately, in the current

exhibits is a strong evidence for the existence of underlying structures.

state of affairs, there is a tendency for each perspective to develop independently from the others.

The current work is not entirely excluded from this critique, as the models which will be presented are all based on *static* perspectives on language and they only focus on syntax. However, one of the primary goals of this work is to provide a bridge between computational models and traditional syntactic theories of language, two approaches which are increasingly diverging from each other. We are strongly in favor of a more shared agenda between syntacticians and computational linguists and in §1.4 we illustrate a possible way to achieve this.

1.2 Syntactic structures of language

In this thesis we will focus on a number of syntactic models of language. Syntax is the study of the rules governing the construction of phrases and sentences in natural languages. Most existing syntactic theories analyze language at an intermediate level: they assume words as the elementary units of production, and sentences as the largest elements under investigation. This is a strong simplifying assumption, as there are processes both below word level and above sentence level, which cannot be considered entirely independent from syntax. This separation is justified as the first step for isolating the phenomenon under study, i.e., the rules for describing how words are put together to form sentences. Ultimately, any syntactic theory should still define bridges to other levels of analysis, such as phonetics, phonology and morphology (below word level), as well as pragmatics, prosody and discourse-processing (above word and sentence level).

Semantics is another linguistic field which studies the *meaning* of language productions. Similarly to syntax, semantics typically analyzes language at the intermediate level between word and sentence boundaries. As there is a great amount of overlapping concepts between the two fields, their separation is somehow artificial, and varies upon the definition of the linguistic theories within the two domains.

1.2.1 Syntactic representation and generative processes

Although the tradition of investigating language syntax can be dated back to Panini's work (circa 5th century BC), the discussion about which theory we should use is still open. Any syntactic theory presupposes a certain type of structure beyond the directly observable utterances, and attempts to explain how to map the surface form, i.e., the sequence of words in the sentence, into the hidden representation. It is therefore important to distinguish between the *syntactic representation* of a theory, i.e., the type of syntactic structures it presupposes, and its *generative model*, i.e., the description of the way to construct them. Certain theories may remain incomplete in this respect, typically focusing only on the representation (e.g., Tesnière, 1959). A complete syntactic theory should aim at describing both aspects as formally as possible, in order to be unambiguous. We will start by introducing the representational part of syntax, and further on (in §1.3) proceed to describe its generative account.

1.2.2 Different representations

In this thesis we will adopt two main classes of syntactic theories, characterized by two different representations: phrase-structure (PS, also known as constituency structures), and dependency-structure (DS). Historically, these two theories have emerged around the same time during the 1950s: PS in the U.S. with Noam Chomsky (1957), and DS in continental Europe with the somewhat lesser known Lucien Tesnière (1959). Neither Chomsky nor Tesnière formulated their respective theories from scratch, but owed a lot to previous work: Chomsky inherited the notion of (Immediate) Constituency from Wundt (1900), Bloomfield (1933), Wells (1947), and Harris (1951), while Tesnière borrowed several concepts and methods from de Saussure (1915).

In parallel and after the foundational work of Chomsky and Tesnière, a vast number of other syntactic theories have been developed, considered more deep than either PS or DS, such as Categorial Grammars (Ajdukiewicz, 1935; Bar-Hillel, 1953), LFG (Bresnan, 2000; Dalrymple, 2001), HPSG (Pollard et al., 1994), TAG (Joshi, 1985), Word Grammars (Sugayama and Hudson, 2005), Meaning Text Theory (Mel'čuk, 1988) and many others (we will review some of them in §4.4 and §5.6). Our choice to focus on DS and PS is justified by the need to compromise between the possibility of defining *data-driven* parsing algorithms on the one hand, and using linguistically adequate representations on the other. Although there is continuous effort to parse with deep linguistic analyses (e.g., Riezler et al., 2002; Bod and Kaplan, 2003), it is not easy to translate these formalisms into data-driven parsing models, both for their complexity, and for the shortage of corpora directly annotated with these representations.

However, in our approach, rather than committing to a single syntactic analysis, we are interested in taking several approaches in parallel. We are in fact quite agnostic about what is the "correct representation", and we therefore advocate for the integration of different perspectives as a way to obtain a more complete syntactic description. In the remaining part of this section we will introduce and compare the PS and DS representations.

1.2.3 Phrase-Structure

In a phrase-structure (PS) representation, the words of a sentence are *grouped* in hierarchical *constituents* (or *phrases*): a sequence of words, functioning as a single unit, is grouped into a basic constituent; adjacent constituents are grouped into higher phrases forming a hierarchical structure, whose highest level spans all the words in the sentence. For instance, the sentence “*My old friend sang this nice song*” can be mapped into the PS reported in figure 1.1. A more typical (yet isomorphic) *tree representation* for this structure is shown in figure 1.2, where every non-terminal node uniquely maps to a box in the representation of figure 1.1. The non-terminal nodes in a PS tree are usually assigned categorial labels, such as NP (noun phrase), and VP (verb phrase). A version of the same tree with such labels is illustrated in figure 1.4 (left), and will become more relevant when we will introduce a generative account for PS.

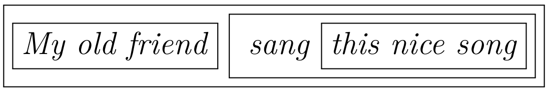


Figure 1.1: Structure of the sentence “*My old friend sang this nice song*”, according to a phrase-structure (PS) representation.

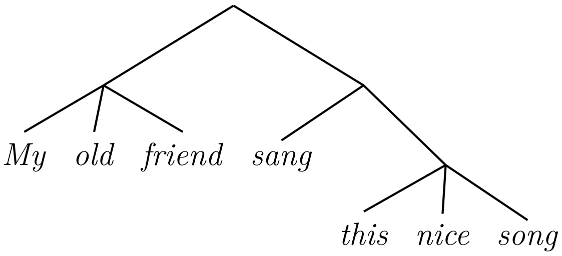


Figure 1.2: Example of the PS in figure 1.1 in an equivalent tree representation: each box in the former representation corresponds to a non-terminal node of this tree.

1.2.4 Dependency-Structure

In a dependency-structure (DS) representation, words of a sentence are *related* to one another (instead of being grouped together as in PS). For every two words A and B in the sentence, there can be a dependency relation. If this relation exists, we say that one of the two words, say B, is a *dependent* or *modifier* of A, while A is the *governor* or *head* of B. Roughly speaking, B is a dependent of A if its presence is only justified by the presence of A, and only if B modifies the meaning of A. All words in a sentence should be connected directly or indirectly by dependency relations forming a *dependency tree*, having a single word as the root of the structure (usually the main verb of the sentence), which governs directly or indirectly all other words.

The same example sentence introduced in figure 1.1 can be assigned the DS in figure 1.3. The highest element of the sentence is the *verb* (sang), which has two direct *dependents*: the actor of the singing (friend) and what has been sung (song). Moreover, the noun ‘friend’ is modified by two dependents (my, old), which specify further qualities of the noun. Analogously ‘song’ is modified by two other dependents (this, nice). It is important to remark that in this simplified DS representation the order of the words is not preserved, but in the DS trees we will employ, word order will be specified (see §2.1.2 and chapter 4).

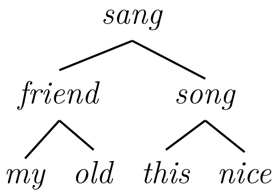


Figure 1.3: Dependency-Structure of the sentence “*My old friend sang this nice song*”, according to Tesnière notation (Tesnière, 1959, p. 14).

1.2.5 Relations between PS and DS

PS and DS are based on different types of structure. PS assumes the notion of hierarchical phrases as abstract entities at intermediate levels of the tree structure. No such grouping is postulated in DS, as words are placed in all the nodes of the tree, and relations between words are the only assumed abstract entities. But we argue that there is no reason to claim the exclusive validity of PS or DS, since each notation focuses on a specific aspect of syntax, viz. grouping vs. relations.

There are, however, more similarities between PS and DS than apparent from a first look. In fact, as will become more clear in later chapters, the two systems are not incompatible with one another, as it is possible to define specific transformations for converting one representation into the other (see §4.3.1), and even define syntactic structures which include both notions of constituents and dependencies (see chapter 5).

1.3 Generative models of syntactic structures

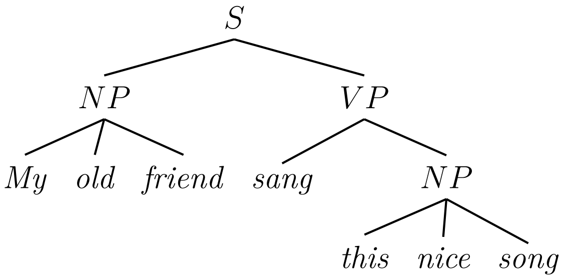
After defining the structural representation of sentences, a syntactic theory should provide a rigorous account for how sentence structures are constructed. The seminal work of Chomsky (1956, 1957) has represented a major turning point in modern linguistics in this sense, as it was the first successful attempt of deriving a *formal theory of syntax*, characterized by the introduction of *generative models*,[3] which can be described as *algebraic machineries* for deriving sentence structures.

[3] The work of Harris (1951) includes the first description of generative grammars for syntax, but Chomsky’s formulation is more complete and formal.

1.3.1 Context-Free Grammars

Chomsky (1957, ch.4) assumes *labeled* phrase-structures as the underlying representation of language syntax, such as the one illustrated in the tree of figure 1.4 (left), and describes a system for generating them, also known as Context-Free Grammar (CFG). A CFG is defined[4] as a finite set of rewriting rules, such as the ones illustrated in figure 1.4 (right), each characterized by a single non-terminal on the left-hand side of the arrow, which *rewrites* to any number of non-terminals and words on the right-hand side of the arrow. Each CFG has a unique starting non-terminal symbol (typically S) which constitutes the root category of all PS trees the grammar can generate.

A *generative model* based on a CFG gives an account for how to generate all sentence structures which are compatible with the grammar. The generative process starts with the starting symbol S in the grammar, and chooses a rule r_S with S as the left-hand side. This rule will constitute the starting branching at the root (top) of the tree. Afterwards, for any non-terminal symbol X at the frontier of the partially constructed tree, the model chooses a rule r_X for extending it. This last step is iterated until all the nodes at the bottom of the tree are words (also called terminals, as they cannot rewrite to anything else).



$$\begin{array}{lll} S & \longrightarrow & NP\ VP \\ NP & \longrightarrow & \textit{My old friend} \\ VP & \longrightarrow & \textit{sang}\ NP \\ NP & \longrightarrow & \textit{this nice song} \end{array}$$

Figure 1.4: Left: the labeled version of the PS tree in figure 1.2. Right: the Context-Free Grammar for generating the PS tree.

1.3.2 Generalized models

So far, we have described generative models only as unidirectional processes: given a grammar, the model can produce all structures compatible with it. But the process can be easily *reversed*: given a set of sentence structures (i.e., a *treebank*), it is possible to extract a grammar which can generate all observed trees, and, if general enough, other unobserved ones.

[4] For a more formal definition of CFGs see §2.2.2.

In the central chapters of this thesis (3, 4, 5), we will take this reverse perspective and make use of hand-annotated treebanks for extracting several generative grammars.

Models for PS Besides Context-Free Grammars, there is an infinity of other generative models for PS that we could take into account. In particular, there are several limitations implicit in a CFG we would like to solve. A CFG is in fact subject to an *under-generation problem*: the nodes in the right-hand side of a rule are inseparable from each other, as they are all attached to the derived tree at the same time, once the rule is applied; a grammar can therefore not generalize over its rules. But the CFG rules needed to describe natural sentences can get arbitrarily long,[5] and it is impossible to define them all as the number of possible combinations is infinite. At the same time, the derivation process makes a strong *independence assumption* when combining the rules, as every choice is only determined by a single node, i.e., the left-hand side of the rule.[6] This leads to an *over-generation problem*: a CFG usually produces, for a given sentence, many syntactic structures which are not acceptable according to human judgement.

In chapter 2 we will explore a range of different generative models which try to solve such limitations: for instance we will consider models which generate a PS tree one node at a time (instead of attaching all symbols in the right-hand side of a CFG rule at once) and *conditioning* every decision on more than a single node present in the partially derived tree, as initially proposed by Magerman (1995), Collins (1997), and Charniak (1997). For each different model, we will need to define specific elementary units (*fragments*), and specific *operations* to combine them into trees. According to the *reversed* perspective, for each combinatory operation there must be a corresponding *deconstruction* operation. Given a treebank we can therefore *decompose* all the trees into a set of fragments in order to derive our grammar.

In particular, in chapter 3 we will focus on one specific generative grammar based on the Data-Oriented Parsing framework (Bod et al., 2003), in which the elementary units are subtrees of unrestricted size extracted from a treebank.

Models for DS More generally, we can also come up with generative models for dependency-structures introduced in §1.2.4. The work of Tesnière (1959), is neither formal nor generative, since it does not provide any algebraic machinery for describing how to combine words into a sentence structure. This does not mean that it is not possible to derive a formal-generative model based on this theory. Like PS, in fact, DS can be described as well-formed *trees*, and this

[5] There is in principle no upper-bound on the number of nodes in the right-hand side. For instance a coordination structure can have an unlimited number of elements which are coordinated.

[6] The severity of this independence assumption can be reduced by including contextual information into the non-terminal labels of the grammar, as discussed in §3.2.2.

can allow us to define specific generative models for this representation. In the last two decades, as the DS representation has become more widely studied in computational linguistics, several generative models have been proposed (e.g., Eisner, 1996a,b). In chapter 4 we will review some of these and describe a novel model for parsing DS.

However, in order to build a supervised model for dependency-structure, we need to have access to a collection of consistently annotated DS trees. As there is no significant manually annotated treebank for the DS representation, we will make use of standard methodology for automatically converting PS treebanks into DS notation (see §4.3.1).

However, the resulting DS representation misses several of the fundamental features which were proposed in the original work of Tesnière (1959), for instance, it does not have a proper way to represent *coordination* constructions (e.g., "*John and Mary*"). The main contribution of chapter 5 is to propose a more elaborated version of dependency-structure which we believe to be more complete with respect to Tesnière's work, and therefore named *Tesnière Dependency-Structure* (TDS). In particular we will define a conversion procedure for transforming a PS tree into a TDS, and propose a generative model for this representation.

1.3.3 Probabilistic generative models

So far we have described generative processes which are purely *symbolic*. A symbolic grammar assigns equal degree of grammaticality[7] to a set of sentences, i.e., the ones it can generate. This is in line with Chomsky's *competence* paradigm (Chomsky, 1965), according to which native speakers have the *internal* ability to decide whether a sentence is grammatical or not. This binary perspective has raised much debate in the last few decades, leading to a *performance* approach which targets the *external* nature of language communication (Levelt, 1974; Scha, 1990; Abney, 1996): language users produce all sort of utterances including those which are not judged entirely sound, but nonetheless constitute real language productions. According to this perspective the grammaticality of a sentence should range on a *continuum* rather than *discretely*.[8]

There are several possible ways to provide a generative model with a notion of grammatical continuity over the generated sentences (and sentence structures). The most commonly adopted strategy, which is followed in this thesis, is to augment the model with a *probabilistic* component: at each step of the derivation process, all available alternative decisions admitted by the grammar are placed

[7] The notion of grammaticality of a sentence is here tightly related to the notion of acceptability.

[8] This graded notion of grammaticality should account for all the factors which make certain sentences more plausible than others. It is in fact common that between two equally sound sentences differing in length, the shorter one is regarded as more acceptable (grammatical) than the other.

in a probabilistic distribution. This gives the means to favor certain choices over others (if the distribution is not uniform). Constructing and validating a probabilistic generative model is nevertheless a difficult task which requires a careful analysis for the delicate decisions belonging to both symbolic and statistical domains as Klavans and Resnik have stated:

> "[...] combining symbolic and statistical approaches to language is a kind of balancing act in which the symbolic and the statistical are properly thought of as parts, both essential, of a unified whole." (Klavans and Resnik, 1996, p. x)

One of the major benefits of adopting probabilistic models on top of symbolic ones is that they allow for solving grammatical ambiguities. In fact, the grammars which will be extracted from large treebanks easily become extremely productive: they generate many novel trees, and many different structures yielding the same sentence (see §2.1.1). A probabilistic model implicitly defines a probability distribution over sentence structures it can generate (obtained from the probability of each single decision in the derivation process), and hence it can place the various alternative structures for a certain sentence in a continuous scale of grammaticality. This can enable us to select the most plausible structure according to the model as the most grammatical one.

The process of disambiguating between possible valid structures of the same sentence is essential for two main reasons. First of all we want to be able to evaluate the syntactic theory under investigation, and we can do this only if we have a single correct analysis (or a restricted set of analyses) for a given sentence. Second, syntactic disambiguation is considered one of the most important tasks for developing natural language processing applications.

1.4 Computational models of syntax

As this thesis aims at building computational models of syntax, it is worth reflecting what we mean by them and what is their relevance in the study of language structures.

Given any formal theory about a real-world phenomenon, we can build a computational model (CM) for implementing and testing it. The theory needs to be formally defined in order to be integrated into the CM, viz. it needs to describe precisely how the CM should map any set of partial information to some informative counterpart. For each experiment, we provide the CM with partial information of the observed system, and ask the CM to return novel information about the system. Finally, we can quantify in how far the predicted outcome differs from the observation.

During the last few decades computer models have been adopted in all scientific fields: from physics, to astronomy, chemistry and biology, computational approaches are currently used to validate a full range of scientific theories. One of the historical examples of computational models in chemistry is DENDRAL (Lindsay et al., 1980), a computer system which aims at determining the molecular structure of an organic chemical sample, given its spectroscopic data. The system has strong background knowledge about chemistry laws, i.e., how atoms combine with each other. For instance, it knows that carbon atoms have *valence* four, nitrogen valence three or five, oxygen valence two, and so on. According to these chemistry laws, the system could combine, for example, six carbon atoms, thirteen hydrogen, one nitrogen, and two oxygen atoms into over 10,000 different ($C_6H_{13}NO_2$) structural descriptions (Buchanan, 1982, p.135). All the CM has to do is to acquire the surface information of the chemical sample (i.e., the spectroscopic data), and derive what is the most likely chemical structure according to the theory.

In our case a computational models needs to implement some syntactic theory, both in its representational and generative aspect, and be able to derive the most likely structure of an input sentence given its surface form. There is, however, a striking analogy with the DENDRAL project illustrated before: where in chemistry a model attempts to predict how atoms connect with one another, a syntactic model does the same with words as elementary blocks. It is no coincidence that terms like *valence* have been adopted in linguistic theories, as the combinatorial nature of words strongly resembles that of chemical elements.[9] But there is also a major difference between the two approaches: while in chemistry there is a wide consensus about the molecular description of organic material and the methodology to determine it, in syntax there is no agreement on underlying structures, and no ultimate way to verify them.

What is then the role of computational models of syntax? We believe that a CM of syntax (and more generally of language) can provide a major contribution to linguistic theories. The possibility of implementing a number of syntactic theories into a CM gives us the means to effectively predict the “behavior” of those theories, and although it will not provide any final judgement for their validity, it would enable us to create a common ground for comparing and evaluating them.

Building a computational model of syntax involves interdependent efforts between *syntacticians* and *computational linguists*. The role of the syntacticians is, as we see it, to define the linguistic theory under enquiry. This includes i) the formulation of the guidelines for annotating a big set of sentences into the assumed representation (the treebank), ii) the description of the generative process for constructing such structures according to the theory (implicitly defining a way to extract a grammar from the treebank) and iii) the definition of the *evaluation*

[9] The analogy between language syntax and chemistry is not new: it has been mentioned by several linguists including Jespersen (1937, p. 3), Tesnière (1959, p. 238), who imported the notion of valence (see p. 109 and §5.2.2), and Chomsky (1957, p. 44).

criteria.

The role of a computational linguist is to implement the linguistic theory into a computational model. This includes i) the definition of a consistent data structure for representing syntactic constructions, ii) the implementation of an algorithm for extracting the fragments underlying the grammar from the training treebank, iii) the implementation of a statistical *parser* (or any alternative disambiguation machinery) for obtaining the most likely structure of novel sentences according to the (probabilistic) model, and iv) the automatization of the evaluation procedure.

In practice, such division of tasks does not need to be sharply defined, and it is even desirable that all points from either side are discussed from both perspectives (and, of course, an individual researcher can be both a linguist and a computational linguist). It is unfortunately the case, however, that there is relatively little collaboration between people working in the two fields (see Klavans and Resnik, 1996), as linguists do not typically rely on quantitative methods for evaluating their hypotheses, and computational linguists are usually more attracted by the performance of a model rather than by its linguistic implications. This description is of course rather simplistic and in many respects imprecise, as there are several exceptions to this view (see for instance Baldwin and Kordoni, 2009), but it is nonetheless a widely recognized tendency.

We believe that CL is currently facing the challenge to bridge the gap between theoretical and computational research on language. Regarding parsing, the need to integrate the two perspectives is well illustrated by Mark Johnson:

> "[...]statistical parsers define the probability of a parse in terms of its (statistical) features or properties, and a parser designer needs to choose which features their parser will use, and many of these features reflect at least an intuitive understanding of linguistic dependencies." (Johnson, 2009)

We hope that the current thesis could help at least in small part to meet those challenges: in particular we hope that our effort to formulate theory-independent (probabilistic) generative models (chapter 2) and our attempt to explore different syntactic representations would encourage more discussion, especially with syntacticians. They could in fact greatly contribute to improving computational models by becoming principal actors in the definition of the syntactic models and formulating more sound evaluation criteria.

1.5 Thesis overview

In the following, we present a short overview of the remaining chapters of this thesis.

Chapter 2 In this chapter we illustrate a general paradigm for the formal definition of generative models based on generic tree structure representations. This chapter is rather technical but is intended to present the general methodology which is adopted in the specific models proposed in the rest of the thesis. All the specific models which are presented in later chapters (3, 4, 5), can be in fact seen as instantiations of this general methodology. It is however possible for the reader to skip this chapter, as its content is not indispensable for understanding the rest of the thesis. The chapter is divided in two parts: the first part focuses on the definition of symbolic tree-generating models, and presents several grammar examples, including some which will be used in the rest of the thesis. The second part explains how to extend a symbolic model with a probabilistic component, and it introduces a general reranking technique for simulating the behavior of a parser based on a given probabilistic tree-generating grammar.

Chapter 3 In the third chapter we focus on the PS representation and present a probabilistic generative model based on the Data-Oriented Parsing framework (Bod et al., 2003). We first define a novel way for extracting a large set of representative fragments from the training corpus, which will constitute the symbolic grammatical backbone of the model. We then show how to define several probabilistic instantiations of such a symbolic grammar. We test the system on different treebanks (for several languages) using a standard CYK parser, via a specific grammar transformation. The content of this chapter is partially extracted from the following publications:

Sangati et al. (2010) : Federico Sangati, Willem Zuidema, and Rens Bod. Efficiently extract recurring tree fragments from large treebanks. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC)*, Valletta, Malta, May 2010.

Sangati and Zuidema (2011) : Federico Sangati and Willem Zuidema. Accurate Parsing with Compact Tree-Substitution Grammars: Double-DOP. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 84–95, Edinburgh, July 2011.

Chapter 4 In this chapter we focus on the DS representation. In the introductory part we present the main differences and commonalities between PS and DS. In the rest of the chapter we explain in depth how to use a reranking technique for testing a number of probabilistic models of DSs, based on bi-lexical grammars (Eisner, 1996a,b). We finally test how the proposed models perform on the standard dependency parsing task for the English WSJ treebank (Marcus et al., 1999). The content of this chapter is partially extracted from the following publication:

Sangati et al. (2009) : Federico Sangati, Willem Zuidema, and Rens Bod. A generative reranking model for dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT)*, pages 238–241, Paris, France, October 2009.

Chapter 5 In this chapter, we introduce a novel syntactic representation, i.e., the Tesnière Dependency-Structure (TDS). This representation is the result of a formalization effort of the dependency-structure scheme proposed by Tesnière (1959). In order to obtain a large corpus of TDS trees, we define an automatic procedure for converting the English Penn WSJ treebank into this novel representation. We introduce a generative model for parsing TDS trees, and evaluate it using a reranking methodology on three new proposed metrics. Finally we discuss the main advantages of the TDS scheme with respect to the original PS format, to the standardly adopted DS representation, and other proposed treebanks which have resulted from manual or automatic conversion of the same treebank. The content of this chapter is partially extracted from the following publications:

Sangati and Mazza (2009) : Federico Sangati and Chiara Mazza. An English Dependency Treebank à la Tesnière. In *The 8th International Workshop on Treebanks and Linguistic Theories (TLT)*, pages 173–184, Milan, Italy, 2009.

Sangati (2010) : Federico Sangati. A probabilistic generative model for an intermediate constituency-dependency representation. In *Proceedings of the ACL Student Research Workshop*, pages 19–24, Uppsala, Sweden, July 2010.

Chapter 6 The concluding chapter is dedicated to the final remarks about this thesis, and summarizes its main contributions.

Chapter 2
Generalized Tree-Generating Grammars

Velvet imperative – “Name the sentence
Parts in ‘He did give us fish to eat.’ ”
Echoes, seeking in syntax of synapse, sense.
They sit before me in the present, tense,
Except for those who, vaulting to the feat,
Sit convicted (subject, “He” – and complete;
“Us” the object, indirectly; recompense
Of fish, the direct object; “fish to eat”
Shows the infinitive can modify.)
Despair for him who cannot comprehend?
Who cannot in the pattern codify
For wonder that objective case can bend
To subject? We who know our truth react,
And never see the substance of the fact.

Bernard Tanner, 1963

2.1 Introduction

This chapter is intended to provide the reader with a general description of probabilistic models for learning syntactic tree structures. The *learning* methods adopted in this thesis are purely *supervised*, meaning that each system under consideration is initially presented with a large number of syntactically annotated natural language sentences (the training treebank), and the task is to learn how to produce novel syntactic tree structures for unobserved sentences. This methodology is complementary to *unsupervised* approaches which aim at deriving syntactic structures from unannotated sentences (e.g., Klein, 2005; Bod, 2006; Seginer, 2007; Blunsom and Cohn, 2010).

As in this thesis we will be dealing with a number of different syntactic tree structures, and each representation can be instantiated in several *generative models*,[1] we are interested in presenting a general methodology which is applicable to them all. The possibility of working with a general paradigm introduces a number of advantages: i) it allows for comparing more easily the various models as they can be presented with a unified notation; ii) it facilitates the process of implementing novel generative models by reducing the effort required for the actual definition of the model, as the representation of the event space is unique within the whole framework; iii) together with the introduction of a reranking methodology, all the models can share a single evaluation procedure.

The main contributions of this chapter are: the introduction of symbolic tree-generating grammars (§2.2), their probabilistic extension (§2.3), and the description of a reranking methodology for parsing (§2.4). This general perspective on parsing tree structures is reminiscent of other formalisms such as the Simulate Annealing framework (Sampson et al., 1989), the Probabilistic Feature Grammars (Goodman, 1998, p.185), and the Polarized Unification Grammars (Kahane, 2006).

2.1.1 Symbolic and Probabilistic models

The process of defining a computational model of syntax can be divided into two steps which can be treated in large measure separately. In the first phase we have the extraction of a *symbolic grammar* (§2.2) from the treebank, and in the second one its *stochastic instantiation* (§2.3).

A symbolic grammar refers to the algebraic machinery used to derive a sentence tree structure by combining elementary syntactic units. It is generally

[1] The terms *generative model* and *generative grammar* will be often used in this chapter. The two terms are often interchangeable, although there is a subtle difference: while a model refers to an abstract machinery to generate syntactic structures, a grammar is a more specific instantiation. For instance we could have two separate grammars extracted from different treebanks, instantiating the same model.

composed of two primitives: a set of atomic *fragments*[2] (the lexico-syntactic units defined over a set of symbols), and a set of *recombining operations* over the fragments. The system uses these two primitives to generate the observed tree structures, and, if general enough, novel tree structures for unobserved sentences.

It is often the case that, when a grammar succeeds in covering a big set of sentences, it also increases in *ambiguity*, so that it generates many different structures for a given sentence. A certain degree of ambiguity is in general necessary, since there are plenty of cases where the same sentence allows for different interpretations which map to separate syntactic analyses. Frazier (1979) gives the following example (2.1) with two interpretations (2.2, 2.3).

(2.1) They told the girl that Bill liked the story.

(2.2) They told the girl [*that Bill liked the story*].

(2.3) They told [*the girl that Bill liked*] the story.

A more problematic type of ambiguity is encountered when the chosen symbolic grammar tends to over-generalize, and allows for a variety of analyses which are rejected with high confidence by human judgment. A typical example is presented in example 2.4 (Martin et al., 1987). In this example, even when imposing how to group the words in the sentence into the correct *chunks*[3] and assigning the exact categories to these chunks (as in example 2.5), there is a combinatorial explosion of (very unlikely) syntactic analyses that are licensed by commonly used symbolic grammars. Figure 2.1 shows some of the ambiguous relations typically licensed by such grammars.

(2.4) List the sales of products produced in 1973 with the products produced in 1972.

(2.5) [List]$_{\mathrm{V}}$ [the sales]$_{\mathrm{NP}}$ [of products]$_{\mathrm{PP}}$ [produced]$_{\mathrm{V}}$ [in 1973]$_{\mathrm{PP}}$ [with the products]$_{\mathrm{PP}}$ [produced]$_{\mathrm{V}}$ [in 1972]$_{\mathrm{PP}}$.[4]

In order to resolve this type of ambiguity, a stochastic component is introduced in the second phase of the definition of our models. A stochastic model, in fact, defines a probability distribution over the possible structures yielding a specific sentence, and allows us to select the most probable one as the one that has highest chance to be correct according to the model. In §2.3 we will illustrate possible

[2] We will use the general term ‘fragment’ to indicate a lexico-syntactic unit of a grammar. Depending on the specific model, a fragment can refer to an abstract grammatical rule or a production including lexical items.

[3] A *chunk* includes a content word and any number of functional words. See also §5.2.2.

[4] V stands for *verb*, NP for *noun phrase*, and PP for *prepositional phrase*.

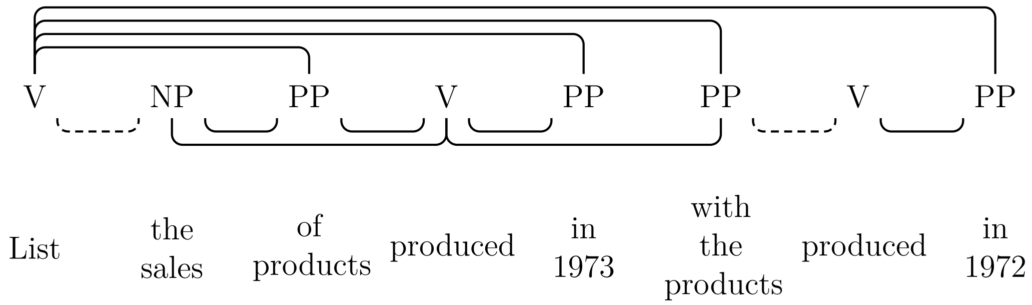


Figure 2.1: Example of an ambiguous sentence. Each edge indicates a possible syntactic relation between two chunks. For instance ‘in 1973’ could refer to ‘produced’ (as in *something produced in 1973*) or to ‘List’ (as in *List something in 1973*). Dashed lines indicate unambiguous relations. The combinatorial explosion of syntactic analysis derives from the presence of four prepositional phrases (PP), each being a possible argument of any preceding verb.

ways of estimating the probability distribution of sentence structures given an underlying symbolic grammar.

A probabilistic model can be implemented by a *parser* which can be used to obtain the most likely syntactic structure of a given sentence according to the model. However, a parser is usually tied to a specific model and a specific syntactic representation. As the aim of this chapter is to describe a general methodology, we will propose a *reranking* framework (cf. §2.4) which can allow us to evaluate different probabilistic models across various syntactic tree representations.

This generalization will become extremely useful in the later chapters where we will study how to model different syntactic schemes (DS in chapter 4, and TDS in chapter 5). Although each representation imposes its idiosyncratic constraints on the implementation of a specific learning model, we will show how it will be possible to instantiate the general reranking paradigm to each of these cases with relatively little effort. Regarding PS, in chapter 3 we will describe how to implement a full parser model to implement the specific probabilistic grammar under investigation.

2.1.2 Tree structures

In the current chapter, in line with the goal of having a general treatment of models of syntax, we will choose to generalize from any syntactic tree representation. A *tree structure* is defined as a connected acyclic graph, with a single vertex as a root, and a defined ordering among the children of each node.

Using a general notion of tree structure allows us to abstract over the details of the syntactic tree representations which we are going to describe in the following chapters. There is only one important structural difference between PS trees and (T)DS trees: because in (T)DS trees all internal nodes are also words, all children of a node are also ordered with respect to the parent node (according to the linear order of the words in the sentence).[5,6] Two examples of generic tree structures with and without ordering between each node and its parent are shown in figures 2.2 and 2.3. In these examples we have explicitly chosen to use abstract node labels and avoid any difference between terminals $\{B, F, G, H, I, K, L\}$ and non-terminals $\{A, C, D, J\}$. In the current chapter the non-ordered tree version (figure 2.2) will be mainly adopted.

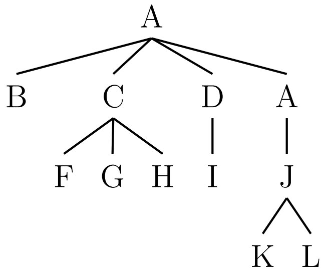


Figure 2.2: A generic tree structures without ordering between a node and its parent. This structure defines a partial order between the nodes.

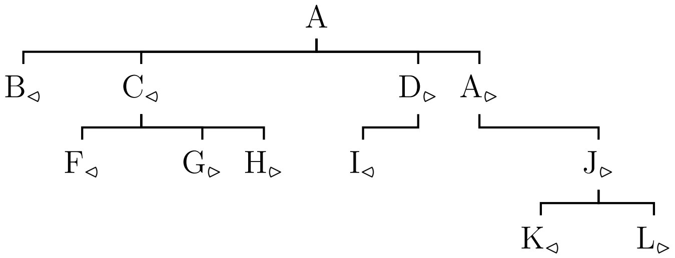


Figure 2.3: A generic tree structures with ordering between a node and its parent. The order between any node (excluding the root) and its parent is marked by means of ◁ (preceding relation) or ▷ (following relation). This structure defines a linear order (total order) between all the nodes.

[5] In the original formulation of dependency-structures (Tesnière, 1959) the linear order of the words was not taken into consideration. See also §5.2.1.

[6] TDS trees are also a bit different structurally from DS, since each node can be expanded to multiple tree structures by means of coordination (see §5.2.3). Although this will add a certain degree of complexity in the structure, the overall learning scheme will remain the same as in PS and DS.

2.2 Symbolic Generative Models for Trees

A symbolic tree-generating grammar can be defined as follows:

2.2.1. Definition. A symbolic tree-generating grammar $\mathcal{G}$ is a tuple:

$$\mathcal{G} = \langle \mathcal{N}, \mathcal{A}, \odot, \oplus, \oslash, \circledast, m, F_1, \ldots, F_m, C_1, \ldots, C_m, O_1, \ldots, O_m \rangle$$

where $\mathcal{N}$ is a finite set of symbols (or nodes), $\mathcal{A} \subset \mathcal{N}$ a set of artificial symbols, $\odot \in \mathcal{A}$ the start symbol, $\oplus \in \mathcal{A}$ the stop symbol, $\oslash \in \mathcal{A}$ the null symbol, $\circledast \in \mathcal{A}$ the wild-card symbol, $m \in \mathbb{N}_{\geq 1}$ the number of operations allowed in the grammar, F_i ($i \in \{1, \ldots m\}$) a finite list (or class[7]) of elementary fragments, C_i a finite list (or class) of conditioning contexts such that $|F_i| = |C_i|$, and O_i a compositional operation that can apply only to fragments in F_i and conditioning contexts in C_i.

A generative model implementing a symbolic grammar is able to derive a tree structure through a series of *generative events*. Each generative event modifies a partially derived structure by means of a specific compositional operation introducing a new elementary fragment. In order for the operation to apply, some specific part of the generated structure, i.e., the conditioning context, must uniquely specify the site where the fragment is introduced. Given a model, each fragment $f_{i,j} \in F_i$ is uniquely mapped to a specific conditioning context $c_{i,j} \in C_i$.

2.2.1 The event space

An *elementary fragment* represents a new piece of the tree introduced by a generative event. It is defined in the general case as a multiset of nodes; if empty, the corresponding operation is a *transformation*[8] of the current tree: no novel nodes are introduced in the structure. Figure 2.4 presents 5 examples of generally valid fragments.

Every class of fragments F_i must characterize the *topology* of each of its members $f_{i,j}$, i.e., the pairwise relations between the nodes in $f_{i,j}$. When the nodes form a tree structure, the edges implicitly define these relations. In other cases the relations need to be clearly specified. For instance the fragment (b) in figure 2.4 represents a sequence of nodes which do not form a tree structure.[9] Such a list of nodes is often used (without dotted edges) in later examples to define a sequence of adjacent siblings, which will share the same parent node already present in the partially derived tree. It is important to specify that such relations

[7] We will generally use the term *list* to refer to an enumeration of elements (duplicates allowed), while *class* will be used to refer to the set of the elements in the list. We have chosen this convention to allow a unique mapping between each $f_{i,j} \in F_i$ and $c_{i,j} \in C_i$ specified by the index j.

[8] For an example of a transformation see example 2.2.8 at page 36.

[9] In order to be a tree structure the four daughters would need a parent node.

should hold right after the operation is applied, but not necessarily after the tree structure is completed. In fact other generative events might break these relations by introducing novel nodes in the structure. The class F_i can characterize its members imposing a set of properties. For instance a fragment class might specify that no more than 2 nodes are allowed, or that all its fragments are trees.

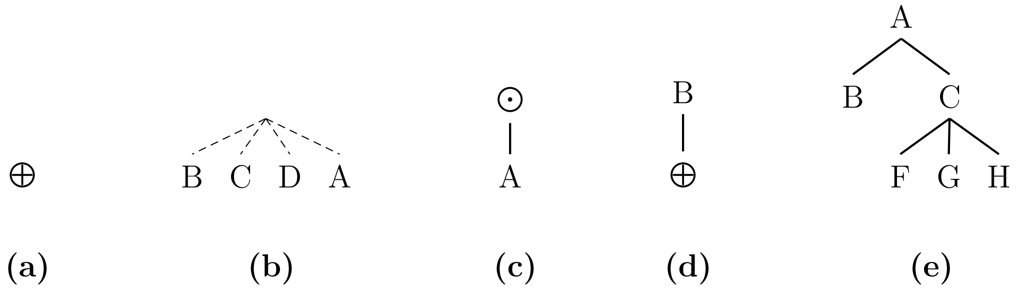


Figure 2.4: Examples of 5 elementary fragments.

2.2.2 The conditioning context

A conditioning context (in short *context*) describes a part of the structure which has been previously generated (therefore also sometimes referred to as a *history*).

Differently from fragments, each context is defined as a multiset containing one or more nodes (no empty contexts are allowed) connected in a tree structure. As for F_i, a model may define for a class C_i of contexts possible constraints on the structure, while the topology (relations between the nodes) is always implicitly defined by the tree structure.

Figure 2.5 shows 6 possible conditioning contexts. When defining a context, a model can introduce an arbitrary number of *check conditions*, which are shown in the contexts by means of two artificial nodes: the null symbol $\oslash$ which represent the absence of a node, and the wild-card node $\circledast$ with represent the presence of an unspecified node. For instance one could enforce that in a certain context, a certain node A does not have any daughters (figure 2.5-b); that nodes B and D are daughters of an unspecified parent node[10] (figure 2.5-c); that A is A's rightmost daughter (figure 2.5-d); that C and D are adjacent daughters of A, without any node in between (figure 2.5-e); and finally that F and H are daughters of A with one unspecified daughter in between (figure 2.5-f).

[10] It is important to understand that in this example the context specifies that B and D are siblings, with B preceding D, but not necessarily immediately (see difference with figure 2.5-e).

In a model it is possible that two context tokens are equivalent, i.e., $c_{i,j} = c_{i,q}$ with $j \neq q$. In this case $c_{i,j}$ and $c_{i,q}$ are the same *context type* since they represent the same structure, but different *context tokens* as they map to different fragments. It is in fact assumed that grammars are not redundant, so that $c_{i,j} = c_{i,q} \rightarrow f_{i,j} \neq f_{i,q}$ with $j \neq q$.

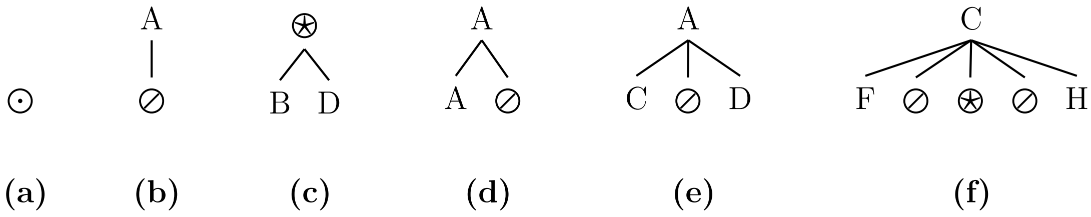


Figure 2.5: Examples of 6 conditioning contexts (or histories).

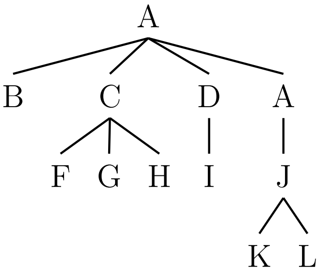
2.2.3 Context-Free Grammar

In order to clarify the notation introduced so far we will now describe an example of Context-Free Grammar (Chomsky, 1956).

2.2.2. Example. [Context Free Grammar] According to definition 2.2.1, we have $m = 1$ (one single fragment and context class and a single operation), each $f_{1,j} \in F_1$ is a list of adjacent daughters (the right hand side of each production rule), each $c_{1,j} \in C_1$ a single node (the corresponding left hand side of the same production) such that $c_{1,j}$ is the parent node of the nodes in $f_{1,j}$, and O_1 the operation of attaching the daughters in $f_{1,j}$, to $c_{1,j}$ (substitution operation). As a check condition the node in $c_{1,j}$ should have no daughter nodes, i.e., it must be a frontier node in the tree structure before the operation is applied. Figure 2.6 (right) shows the contexts and the elementary fragments for the CFG extracted from the tree in figure 2.2, also reported in the left-side of the same figure for convenience.

When using this grammar to generate a structure T, we begin with the start symbol $\odot$ identifying T_0 in all the models. At this point only the first conditioning context in the grammar ($\odot - \oslash$) can apply, and therefore T_1 is obtained by attaching A as the unique daughter node of the initial symbol. At this point there are 2 identical conditioning contexts which are applicable ($A - \oslash$, at indices $j = 2, 3$), and there are therefore two possible ways of continuing the generation of a structure.[11]

[11] See also the section on multiple derivations in §2.2.4



j	C_1	F_1
1	$\odot$–$\oslash$	A
2	A–$\oslash$	B C D A
3	A–$\oslash$	J
4	C–$\oslash$	F G H
5	D–$\oslash$	I
6	J–$\oslash$	K L
7	B–$\oslash$	$\oplus$
8	F–$\oslash$	$\oplus$
9	G–$\oslash$	$\oplus$
10	H–$\oslash$	$\oplus$
11	I–$\oslash$	$\oplus$
12	K–$\oslash$	$\oplus$
13	L–$\oslash$	$\oplus$

Figure 2.6: Left: the PS tree from figure 2.2. Right: the CFG extracted from the left tree. C_1 identifies the left-hand side of the CFG rules, F_1 the right-hand side.

The first 4 steps of a possible derivation of this grammar are illustrated in figure 2.7. At every step the left-most non-terminal node at the frontier of the intermediate structure is the context for the following generative step.[12] This partial derivation can be completed to return the original structure in figure 2.2. The remaining steps are shown in figure 2.8(a) using indices in nodes to refer to the stages of the derivation process in which the nodes are introduced. Figure 2.8(b) reports an alternative derivation licensed by the same grammar. According to this grammar, a tree is complete when all the nodes at the frontier are the stop symbol $\oplus$.

2.2.4 The generative process

The role of a context $c_{i,j}$ is essential in a generative process. If its corresponding fragment $f_{i,j}$ is empty, it specifies where the transformation takes place (specified by the operation O_i). Otherwise, it locates where the corresponding fragment needs to be placed within the current incomplete structure via the associated compositional operation O_i. In this case it is necessary to define the relation between each class of conditioning contexts C_i and the corresponding class of

[12] See also the section on locating a single context at a time in §2.2.4

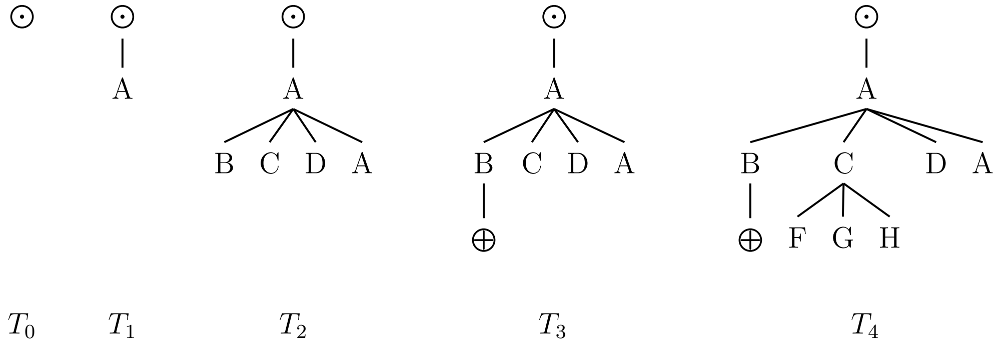


Figure 2.7: The first 5 steps in deriving a tree structure with the CFG in figure 2.6.

fragments F_i. For this it is sufficient to define the relation between a specific node in $f_{i,j}$, and a specific node in $c_{i,j}$, since all the other relations can be derived from it.[13] In the example 2.2.2 just illustrated, the node in $c_{1,j}$ is the parent node of the list of nodes in $f_{1,j}$.

Given an intermediate tree structure T_t obtained after t generative events, a context $c_{i,j}$ might be present in T_t iff its structure (including the check conditions specified by the artificial nodes) are matched in T_t. If this is the case, the corresponding operation O_i is eligible to apply, introducing the fragment $f_{i,j}$. After the operation is performed, T_{t+1} is produced. The presence of a certain context in an intermediate tree must be verified before the operation is applied, and need not hold in later stages of the generation process. In case no more conditioning contexts defined by the model are present in the tree structure, the generation process has reached termination, and the generated tree is a complete structure derived from the grammar.

Ranking the operations and the context classes

Every generative model must define at least one operation, i.e., O_1. If $m > 1$ multiple operations can apply in the same model (e.g., the substitution and adjunction operations in TAG). In such a case it might happen that different operations can apply on the same intermediate structure T_t, i.e., there are at least two contexts $c_{i,j}$ and $c_{p,q}$ (with $i \neq p$) which are present in T_t. When defining a generative model it is sometimes convenient to define a ranking over the operations (and consequently over the corresponding context classes): $rank : m \to \mathbb{N}_{\geq 1}$, where 1 is

[13] Not all the nodes of the fragment need to be introduced in the tree, as there can be possible overlaps between nodes in the fragment and those in the corresponding context.

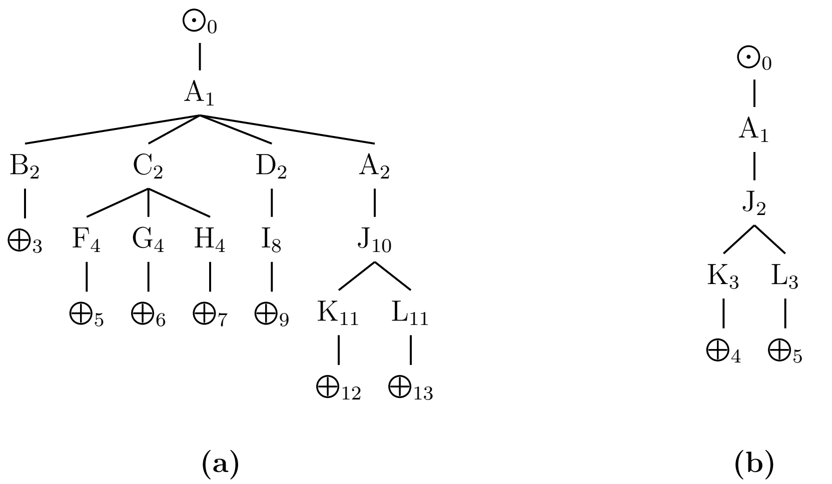


Figure 2.8: Two complete trees which can be derived from the CFG in figure 2.6.

the highest rank.[14] If not specified otherwise, *rank* is the identity function: the first operation (conditioning context class) has priority over the second, which has priority over the third, and so on. At each stage we retain only the context tokens present in the current tree with highest rank. If no ranking is adopted in a specific model, all the operations can apply at a given stage.

Locate a single context at a time

It can happen that at a given stage t of the generation process, a context type[15] is present in T_t at different locations, or that two different context types are present in T_t each at a different location. For instance in the intermediate tree T_2 in figure 2.7 all four nodes B,C,D,A at the frontier of the intermediate structure are possible contexts in which O_1 can apply. In general this can be problematic, as the same sequence of operations applied in different locations might result in different emerging structures. Therefore at every stage of the derivation process we want to ensure that every model defines a way to deterministically locate in T_t a single context type on which to apply the corresponding compositional operation.

[14] If no rank is defined, multiple operations might apply at the same time. In the probabilistic extensions of symbolic tree-generating grammars (§2.3), we will always assume the definition of a ranking function.

[15] Remember the distinction between context type and context token specified at the end of §2.2.2.

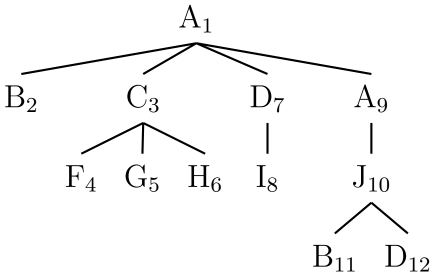


Figure 2.9: A depth-first ordering of the nodes in a tree.

For CFGs the solution is to impose the left-most substitution to apply (as shown in the indices of figure 2.8). In the general case, we define a *location function* $\mathscr{L}(c, T_t)$ returning the location of a context type c in T_t. If $n = |c|$ is the number of nodes in c, we define $\mathscr{L}(c,T_t)$=$\{\ell_1(c,T_t)$, $\ell_2(c,T_t)$, ..., $\ell_r(c,T_t)\}$ with $r \geq 1$ being the number of times c is present in T_t, and $\ell_i(c,T_t) \in \mathbb{N}^n$ the i^{th} location of c in T_t, i.e., a set of indices identifying the positions of c's nodes in T_t. The indices of the nodes in T_t are assigned according to a pre-establish ordering, conventionally a depth-first ordering as shown in figure 2.9. To give an example let us consider context in figure 2.5-c, and assume that the tree in figure 2.9 is our T_t. We then have $\mathscr{L}(c,T_t) = \{\ell_1(c,T_t) = \{1,2,7\}, \ell_2(c,T_t) = \{10,11,12\}\}$.

Every model must ensure that for every two context types $c \neq c'$ that can apply[16] in T_t there exist no $i, j \in \mathbb{N}_{\geq 1}$ such that $\ell_i(c,T_t) = \ell_j(c',T_t)$. In other words, different contexts should be always mutually exclusive, if one applies at a certain location in T_t the other should not be present or its location should differ, and vice versa. This is to ensure that in every model a given sequence of generative events produces a unique structure. We can therefore define an ordering of the locations of all contexts present in T_t, in order to localize a unique context at a time. If $A = \ell_i(a,T_t)$ and $B = \ell_j(b,T_t)$, with $A \neq B$, we have $A < B \Leftrightarrow A \subset B \ \vee \ min(A \smallsetminus B) < min(B \smallsetminus A)$. For example, the contexts c, d, e, f in figure 2.5 are present in tree T_t of figure 2.9 at locations: $\ell_1(c,T_t) = \{1,2,7\} < \ell_1(e,T_t) = \{1,3,7\} < \ell_1(d,T_t) = \{1,9\} < \ell_1(f,T_t) = \{3,4,5,6\} < \ell_2(c,T_t) = \{10,11,12\}$. Context c would therefore apply.

Multiple derivations

Given that the model has successfully selected a single context type c in the intermediate tree T_t at location $\ell_i(c,T_t)$, for most non-trivial grammars there

[16] Remember that in order for c and c' to apply, they must have the same rank.

might be multiple context tokens instantiating c, e.g., $c_{i,j} = c_{p,q} = c$, with $i \neq p \lor j \neq q \land rank(i) = rank(p)$. Trivially all context tokens apply at the same location $\ell_i(c, T_t)$.

When this circumstance arises, multiple distinct fragments are associated with the same type of context, and are therefore eligible to apply. For instance in example 2.2.2 when A is the selected context both rules A → B C D E and A → J can apply.

In this case the model must allow for all corresponding fragments to apply, but in parallel: each fragment is applied on an identical copy of the current partial tree T_t. In other words the current derivation *splits* in multiple derivations, one for every distinct context tokens which applies. This generates novel derivations of the grammar that will potentially differ in all following steps, producing different complete trees. But it can also happen that some of these derivations eventually produce the same complete structure. When this occurs, the grammar is said to have spurious ambiguities.[17] Context-free grammars do not show this type of ambiguity, but TSG grammars do (see example 2.2.3 and chapter 3).

Artificial symbols

The *artificial symbols* thus far introduced are the symbols in the set $\mathcal{A} = \{\odot, \oplus, \oslash, \circledast\}$. In addition, a model can introduce an arbitrary number of artificial symbols, which usually serve as placeholders to represent specific choices which are made along the way in the generative process. All the artificial symbols need to be removed after the termination of the generative process in order to obtained a cleaned complete structure.

The *start symbol* $\odot$ represents the only symbol which is present in the tree structure T_0, before any generation event takes place. In general, the start symbol may be present in fragment $f_{i,j}$, but only as a reference node as no operation O_i can insert a second start symbol into the tree.

The *stop symbol* $\oplus$ represents a node in the tree which signals the termination of a specific generation sub-process, but a derivation does not necessarily need to employ $\oplus$ in order to terminate.[18]

The *null symbol* $\oslash$ represents the absence of a certain node. It can only be used within contexts, typically to define specific check conditions that need to be present in the current incomplete structure for a certain context to apply (see figure 2.5).

Finally, the *wild-card symbol* $\circledast$, like $\oslash$, can be only used for contexts. It matches any possible non-artificial node ($\mathcal{N} \setminus \mathcal{A}$).

[17] This should not be confused with the standard notion of ambiguity of the grammar, for which there exist multiple structures associated to the same sentence.

[18] As explained in §2.2.4, it is sufficient that no conditioning context is present in the derived structure for the generative process to terminate.

2.2.5 Extracting a symbolic grammar from a treebank

Symbolic grammars can be written manually or they can be extracted from a collection of annotated parse trees, the training treebank $\mathscr{T}$. We are mainly interested in pursuing the second option, but in either case we have to choose what the model behind the grammar looks like, i.e., what its composition operations are, and how each fragment and context class is defined. After describing the primitives of the model, we take each structure one by one in the training treebank $\mathscr{T}$ and simulate how it can be derived according to the model under consideration. This procedure can be seen as a *decomposition* of the structure into elementary fragments, each linked to a conditioning context. During this decomposition, we equip the grammar with the fragments and conditioning contexts pairs which are employed. After iterating the same procedure for all the trees in $\mathscr{T}$, we have derived a symbolic grammar. This in turn will be capable of deriving all the tree structures in $\mathscr{T}$, and possibly many other structures which are a generalization of the observed ones in the treebank.

2.2.6 Examples of generative tree grammars

To demonstrate the generality of the notions just described, we will now describe 6 examples of generative grammars. All the grammars are extracted from the PS in figure 2.2, except for the grammar in example 2.2.5 which is extracted from the DS in figure 2.3.

2.2.3. Example. [Tree-Substitution Grammar] In this example we present a TSG grammar which employs subtrees of arbitrarily large size as fragments for the generative operations. This formalism was first implemented in Bod (1992), and will be used in chapter 3 for phrase-structure parsing. For this grammar we choose $m = 1$, $f_{1,j} \in F_1$ is a tree structure, $c_{1,j} \in C_1$ the root node of $f_{1,j}$ exactly as in CFG (example 2.2.2), and O_1 the operation of introducing the full fragment $f_{1,j}$ in the tree by substituting $c_{1,j}$ with the root of $f_{1,j}$. Figure 2.10 (a) shows a TSG grammar which is able to derive the same tree structure as in the CFG example (§2.2.3) with two different derivations as shown in figure 2.10 (a,b). Moreover, this grammar can derive an infinite number of tree structures, and exactly those produced by the previous CFG.

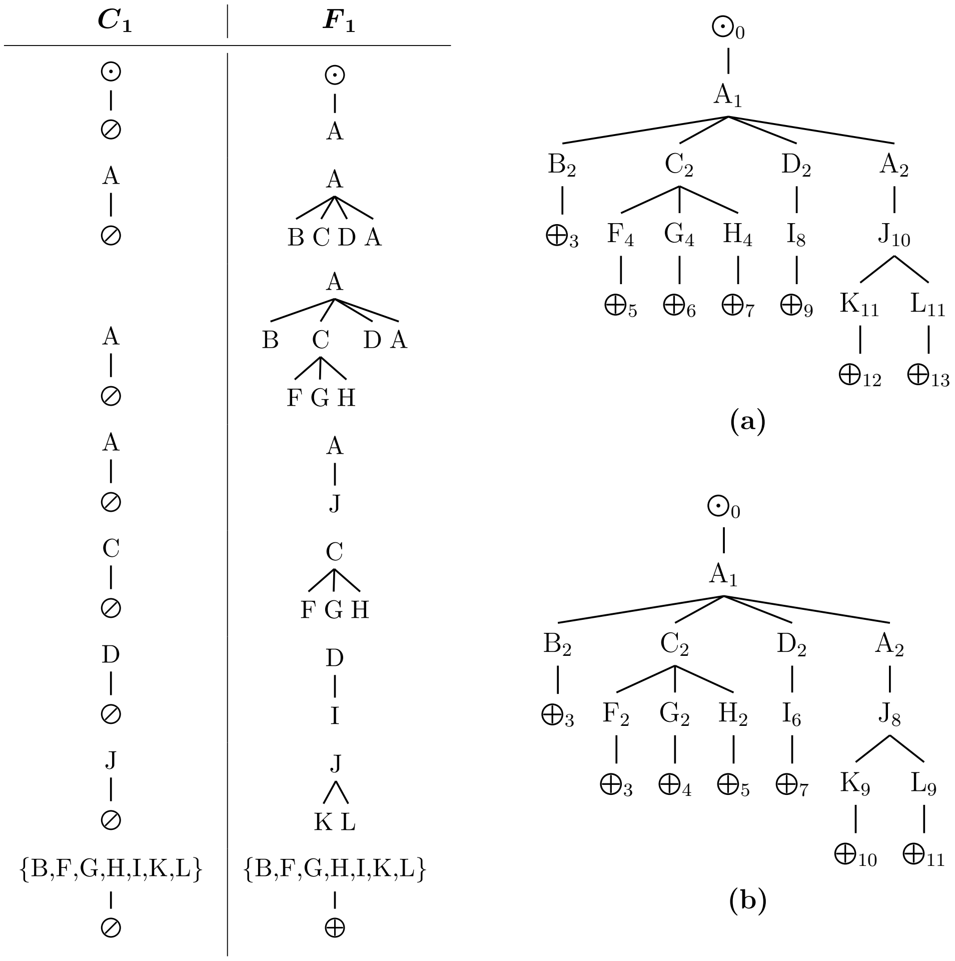


Figure 2.10: An instantiation of a tree-substitution grammar (left), generating the same tree structure with 2 different derivations (a,b). The last line in the grammar, is a short notation to represent 7 different contexts and corresponding fragments.

2.2.4. Example. [Right Sister Insertion Grammar (PS)] This grammar is a simplification of previous work (Magerman, 1995; Collins, 1996), and demonstrates that our formalisms is general enough to model rule markovization without relying on a special-purpose grammar representation.

The basic operation behind this grammar is to introduce a single daughter to a node in the tree, conditioned on its parent node and its left sister. We choose $m = 1$, $f_{1,j}$ is a single node (the new daughter), $c_{1,j}$ contains the root node of $f_{1,j}$ and its right-most daughter (⊘ if no daughters are present), O_1 the operation of attaching $f_{1,j}$ as the new right-most daughter of the parent node in $c_{1,j}$. Figure 2.11 shows this example in more detail.

$\boldsymbol{C_1}$	$\boldsymbol{F_1}$
⊙ ǀ ⊘	A
⊙ ⋀ A ⊘	⊕
A ǀ ⊘	B
A ⋀ B ⊘	C
A ⋀ C ⊘	D
A ⋀ D ⊘	A
A ⋀ A ⊘	⊕
B ǀ ⊘	⊕
C ǀ ⊘	F
C ⋀ F ⊘	G
F ⋀ G ⊘	H
G ⋀ H ⊘	⊕
D ǀ ⊘	I
D ⋀ I ⊘	⊕
A ǀ ⊘	J
F ǀ ⊘	⊕
G ǀ ⊘	⊕
H ǀ ⊘	⊕
I ǀ ⊘	⊕
J ǀ ⊘	⊕
J ⋀ K ⊘	K
J ⋀ L ⊘	L
J ǀ ⊘	⊕
K ǀ ⊘	⊕
L ǀ ⊘	⊕

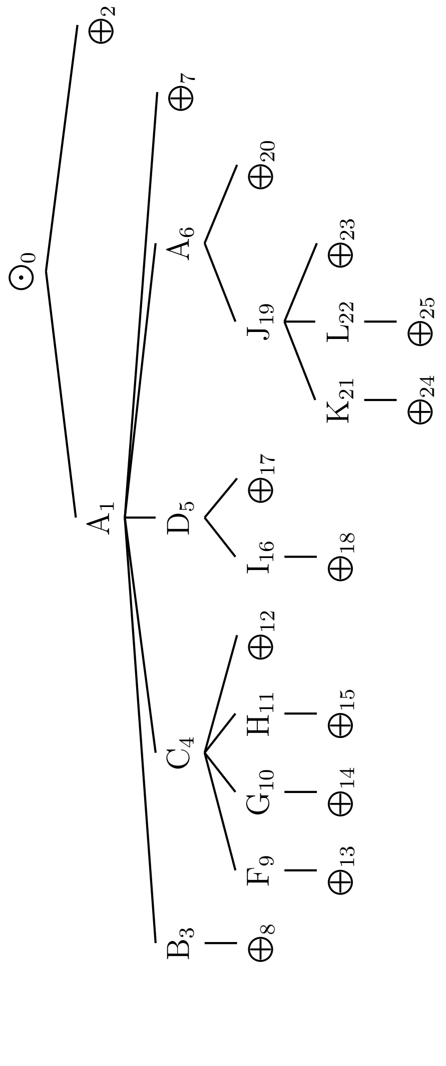


Figure 2.11: A Right Sister Insertion Grammar (left) for PS, generating a tree structure (right) by introducing a new rightmost daughter of a node at every step.

2.2.5. Example. [Right Sister Insertion Grammar (DS)] This grammar is similar to the previous one except that it is adapted to the DS representation (Eisner, 1996b). As before a single node is introduced at each step, conditioned on its parent node and its left sister. We set $m = 1$, $f_{1,j}$ is a fragment partially overlapping with the respective context, $c_{1,j}$ contains the root node of $f_{1,j}$ and its right-most daughter ($\oslash$ if no daughters are present), O_1 the operation of attaching the rightmost daughter in $f_{1,j}$ as the new right-most daughter of the parent node in $c_{1,j}$. Left and right daughters of every node are filled in as two separate processes. Figure 2.12 shows this example in more detail.

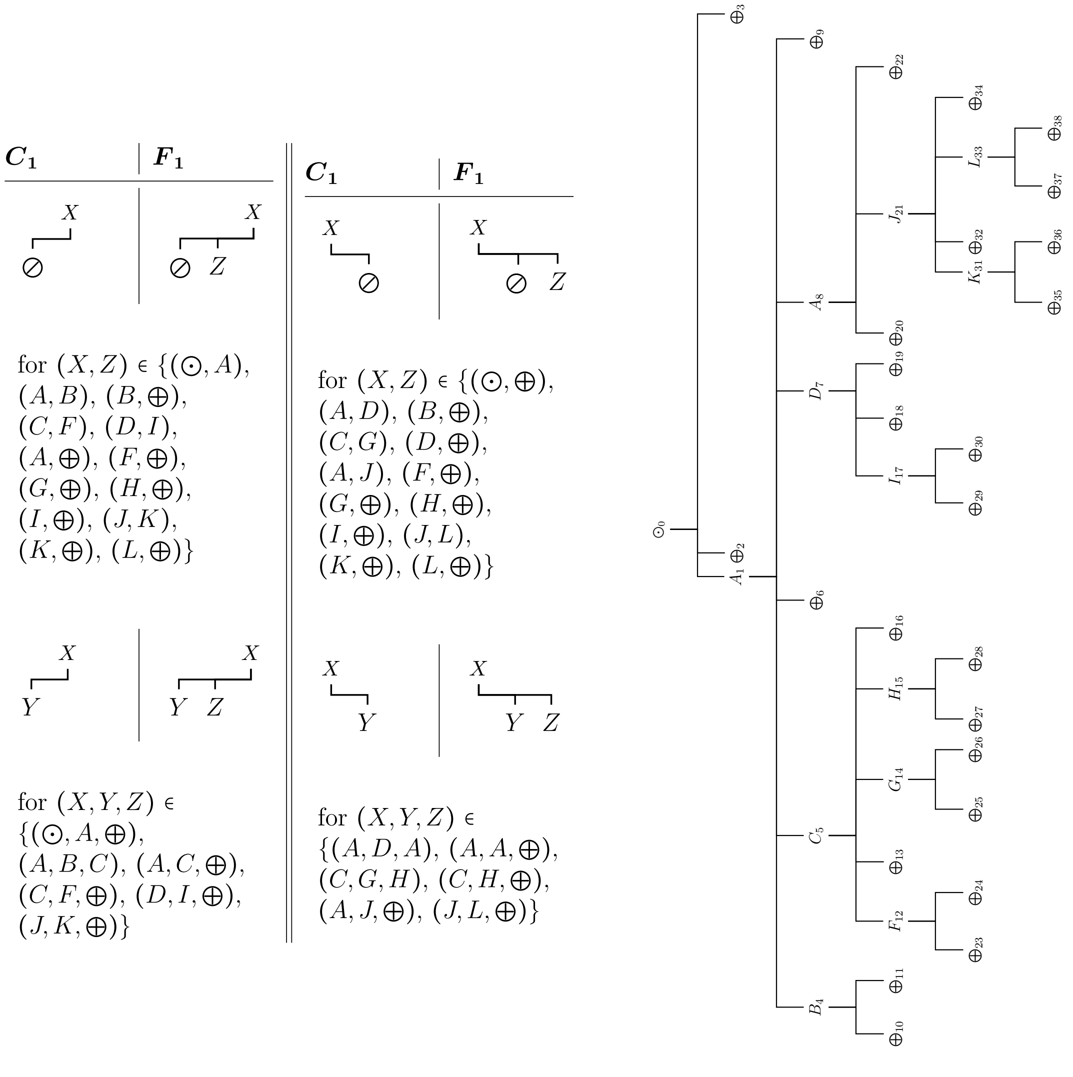

C1
F1
X
⊘
X
⊘ Z
for (X,Z) ∈ {(⊙,A), (A,B), (B,⊕), (C,F), (D,I), (A,⊕), (F,⊕), (G,⊕), (H,⊕), (I,⊕), (J,K), (K,⊕), (L,⊕)}
X
Y
X
Y Z
for (X,Y,Z) ∈ {(⊙,A,⊕), (A,B,C), (A,C,⊕), (C,F,⊕), (D,I,⊕), (J,K,⊕)}
C1
F1
X
⊘
X
⊘ Z
for (X,Z) ∈ {(⊙,⊕), (A,D), (B,⊕), (C,G), (D,⊕), (A,J), (F,⊕), (G,⊕), (H,⊕), (I,⊕), (J,L), (K,⊕), (L,⊕)}
X
Y
X
Y Z
for (X,Y,Z) ∈ {(A,D,A), (A,A,⊕), (C,G,H), (C,H,⊕), (A,J,⊕), (J,L,⊕)}
⊙0
⊕3
A1 ⊕2
⊕9
A8
D7
⊕6
C5
B4
⊕22
J21
⊕19 ⊕20
⊕18
I17
⊕16
H15
G14
⊕13
F12
⊕11
⊕10
⊕34
L33
⊕32
K31
⊕30
⊕29
⊕28
⊕26 ⊕27
⊕25
⊕24
⊕23
⊕38
⊕37
⊕36
⊕35

Figure 2.12: A Right Sister Insertion Grammar (left), generating the DS tree structure in figure 2.3 (right).

2.2.6. Example. [Sandwich Insertion Grammar] This example presents another insertion grammar on PS which allows two different operations: the first is the one encountered in the CFG (example 2.2.2), while the second is a "sandwich" insertion operation. In this case we have $m = 2$. F_1, C_1, and O_1 are as in the CFG. On the other side, $c_{2,j}$ is a tree structure with 3 nodes: a parent node P (with at least two daughters), and two of its adjacent daughters D_1 and D_2, while $f_{2,j}$ is a single node; finally O_2 is the operation of inserting $f_{2,j}$ as a daughter of P in between D_1 and D_2. This grammar could be suitable at modeling a generative process where the arguments of a node are generated at once (O_1), while the adjuncts are inserted one at a time in more restricted contexts. Figure 2.13 shows an instantiation of this grammar in more detail.

$\boldsymbol{C_1}$	$\boldsymbol{F_1}$
⊙–⊘	A
A–⊘	B D A
C–⊘	F G H
D–⊘	I
A–⊘	J
J–⊘	K L
B–⊘	⊕
F–⊘	⊕
G–⊘	⊕
H–⊘	⊕
I–⊘	⊕
K–⊘	⊕
L–⊘	⊕

$\boldsymbol{C_2}$	$\boldsymbol{F_2}$
A B ⊘ D	C
A B ⊘ C	⊕
A C ⊘ D	⊕
A D ⊘ A	⊕
C F ⊘ G	⊕
C G ⊘ H	⊕
J K ⊘ L	⊕

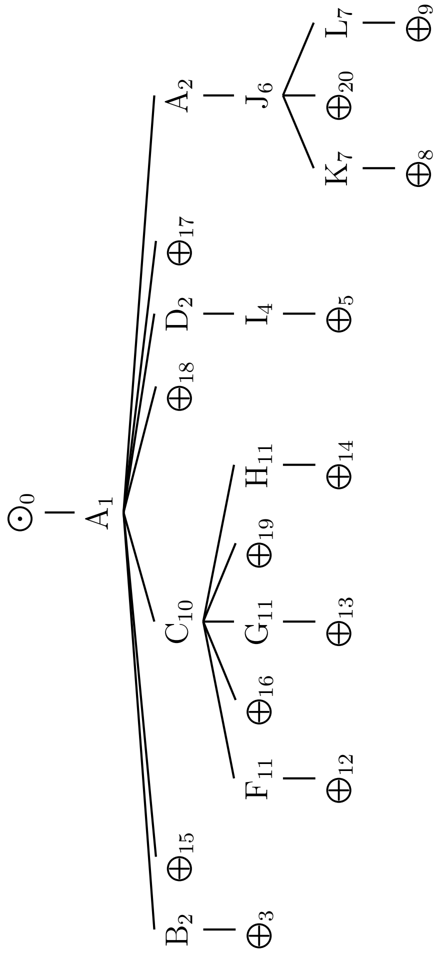


Figure 2.13: An instantiation of a sandwich insertion grammar (left) deriving a tree structure (right) by means of standard CFG productions (as in figure 2.6), and in addition an insertion operation.

2.2.7. Example. [Tree-Adjoining Grammar] In this example we describe a grammar which is based on previous work on TAGs (Joshi, 1985; Joshi and Schabes, 1991). In this case we have $m = 2$. F_1, C_1, and O_1 are as in the CFG described before. O_2 is an operation which allows to *adjoin* a full subtree $f_{2,j}$ at a specific site of the current tree according to the location of $c_{2,j}$ (a single node having at least one daughter node). Each fragment in F_2 is such that one of its frontier nodes should have the same label as its root node X; since multiple X can be present at the frontier, we will mark one as X^* (the *foot* node). The operation O_2 excises the subtree rooted in $c_{2,j}$ from the tree, and puts in its place $f_{2,j}$. Moreover the excised subtree is substituted to the foot node. Both the root node X and the foot X^* cannot be used as locations for other adjunction operations. To ensure this, the two node are marked as $\bar{\mathrm{X}}$ (this could be achieved also by adding special purpose artificial nodes, but we have chosen a shorter notation for simplicity). Figure 2.14 presents an instance of this grammar.

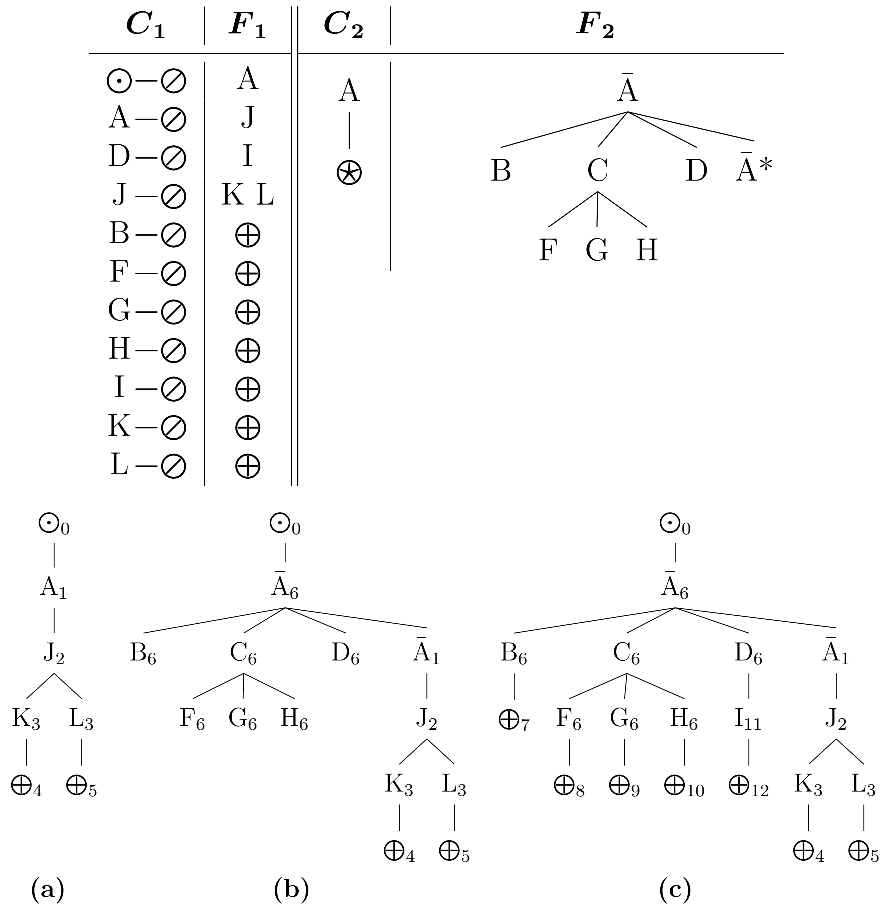


Figure 2.14: An example of a tree-adjoining grammar (above), and one of its derivations (below): after 5 steps (a), after 6 steps (b), after completion (c).

2.2.8. Example. [Bottom-Up Grammar] The last example is inspired by the work of Ratnaparkhi (1997). Similar ideas are described in shift-reduced approaches such as in Aho and Johnson (1974), and Shieber (1983). This example demonstrates that our formalism is general enough to model a wide range of generative models including those based on a bottom-up generation process.

According to this generative model, structures are generated starting from the leaf-nodes, and going upwards until reaching the root of the tree. One way to do this is to introduce internal nodes, level after level.

A bottom-up grammar is described as follows: $m = 3$, O_1 is the operation which adds each leaf node of the tree structure (represented by $f_{1,j}$), as the last daughter of the starting node given its current rightmost leaf ($c_{1,j}$). The second (O_2) and third operation (O_3) have the same context which have equal priorities: $c_{2,j} = c_{3,j}$ is found in the current tree when the right-most leaf Y of the starting node, immediately following X, is completed (it contains the stop symbol $\oplus$ as its rightmost daughter).

If context $c_{2,j}$ is present in the current tree, and O_2 is applied, the operation 'opens' a new constituent, i.e., it inserts a new node ($Z = f_2$) as the parent of Y and the right-sister of X. This operation is explicitly marked with a new artificial node $\nearrow$, inserted in between Z and Y. In addition $\nearrow$ is followed by a new artificial symbol $\rightarrow$ to indicate that the newly open constituent Z needs at least one other daughter, or alternatively by $\oplus$ to indicate that it is completed.

If context $c_{3,j}$ is present and O_3 is applied, a *transformation* takes place: Y is attached as the new rightmost daughter of X. This operation is explicitly marked with a new artificial node $\uparrow$, inserted in between X and Y. In addition $\uparrow$ is followed by a new artificial symbol $\rightarrow$ to indicate that X needs at least one other daughter, or alternatively by $\oplus$ to indicate that it is completed.

Figure 2.15 instantiates a bottom-up grammar which can derive the tree in figure 2.2. In the derivation process, O_1 is used in steps 1-8; in the remaining cases, O_2 is used when $\nearrow$ is inserted (in steps 9,10,14,16,18), and O_3 is used when $\uparrow$ is inserted (in steps 11,12,13,15,17,19).

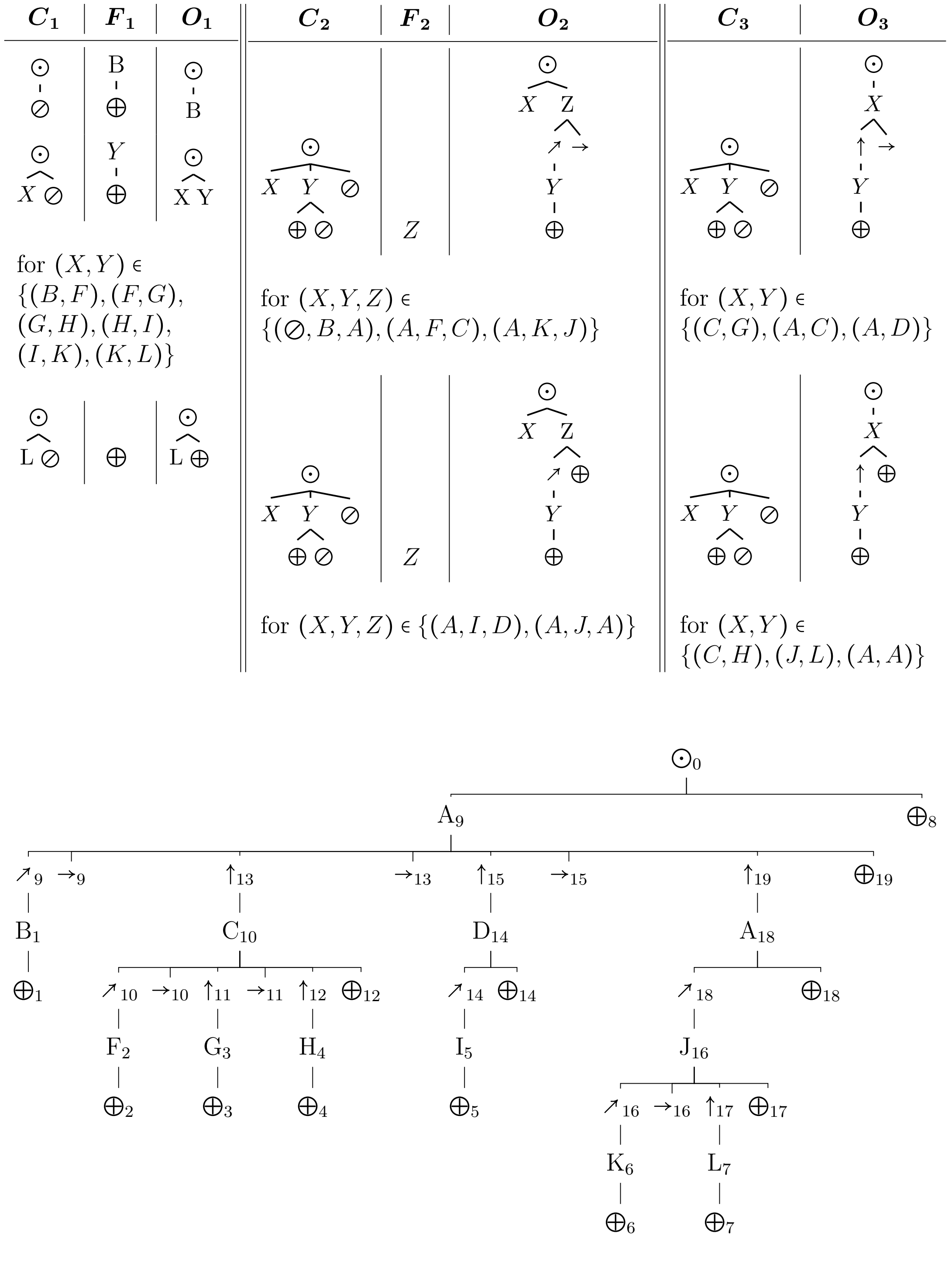

C₁ | F₁ | O₁
C₂ | F₂ | O₂
C₃ | O₃
for (X,Y) ∈ {(B,F),(F,G),(G,H),(H,I),(I,K),(K,L)}
for (X,Y,Z) ∈ {(⊘,B,A),(A,F,C),(A,K,J)}
for (X,Y) ∈ {(C,G),(A,C),(A,D)}
for (X,Y,Z) ∈ {(A,I,D),(A,J,A)}
for (X,Y) ∈ {(C,H),(J,L),(A,A)}
⊙0
A9
⊕8
↗9 →9 ↑13 →13 ↑15 →15 ↑19 ⊕19
B1 C10 D14 A18
⊕1
↗10 →10 ↑11 →11 ↑12 ⊕12 ↗14 ⊕14 ↗18 ⊕18
F2 G3 H4 I5 J16
⊕2 ⊕3 ⊕4 ⊕5
↗16 →16 ↑17 ⊕17
K6 L7
⊕6 ⊕7

Figure 2.15: A bottom-up grammar (left) and one of its derivation. All the nodes in the structure are introduced one level at a time.

2.3 Probabilistic Generative Models for Trees

The symbolic formalism developed so far allows for a straightforward probabilistic extension. For simplicity we assume that the underlying symbolic model always defines a ranking function over the operations (when multiple are present). This ensures that there is a unique operation which applies at a given stage of the derivation process (see 2.2.4). A probabilistic tree-generating grammar is defined as follows:

2.3.1. Definition. A probabilistic tree-generating grammar $\mathcal{G}_p$ extends the definition 2.2.1 of a symbolic tree-generating grammar with a function $p(f_{i,j}|c_{i,j})$, which returns the probability of a generative event, i.e., employing a fragment $f_{i,j}$ in an intermediate tree where context $c_{i,j}$ applies, by means of operation O_i. The two indices i, j are such that $1 \leq i \leq m$ and $1 \leq j \leq |F_i| = |C_i|$. The index i will be always fixed as it specifies the operation which is deterministically selected by the rank according to the symbolic model. We maintain this index in order to be consistent with the previous notation of the underlying symbolic grammar (§2.2).

The function p must correspond to a proper probability distribution over the fragments sharing the same conditioning context type. This means that for every i, j the following equation should be satisfied:

$$\sum_{\substack{q \ s.t. \\ c_{i,j}=c_{i,q}}} p(f_{i,q}|c_{i,j}) = 1 \tag{2.6}$$

2.3.1 Resolving the syntactic ambiguity

As mentioned in §2.1.1 a symbolic generative grammar can be extended with a stochastic component to resolve cases of syntactic ambiguities. In fact, for any non-trivial symbolic grammar, there are many possible tree structures yielding a given sentence s. We therefore need to define a way to induce probabilities over the trees generated from the grammar whose yield is s, and select the one with maximum probability. This is equivalent to say that we want to obtain the parse tree $\hat{T}$ such that:

$$\begin{aligned} \hat{T} &= \arg\max_T p(T|s) & (2.7) \\ &= \arg\max_T \frac{p(T,s)}{P(s)} & (2.8) \\ &= \arg\max_T p(T,s) & (2.9) \end{aligned}$$

2.3.2 The probability of a tree

Given a probabilistic tree-generating grammar $\mathcal{G}_p$, and a tree structure T it can generate, we want to obtain $P(T|\mathcal{G}_p)$: the probability of T according to $\mathcal{G}_p$. In the following explanation we assume an underlying probabilistic grammar $\mathcal{G}_p$ and we therefore use the simpler notation $P(T)$.

We start by decomposing T into a sequence of independent generative events, forming the first derivation of T: $\delta_1(T) = (e_1, e_2, \ldots, e_n)$ producing a sequence of intermediate structures $(T_1, T_2, \ldots, T_{n-1})$ and terminates with the complete tree $T_n = T$. Each event $e = \langle f_{i,j}, c_{i,j} \rangle$ is characterized by an elementary fragment $f_{i,j}$ and a corresponding conditioning context $c_{i,j}$ which are present in $\mathcal{G}_p$. The probability of each event $p(e)$ is given by the function $p(f_{i,j}|c_{i,j})$ introduced before (in definition 2.3.1).

Since the generative events are defined to be independent, we can compute the probability of a derivation by multiplying the probabilities of its events.

$$P(\delta_i(T)) = \prod_{e \in \delta_i(T)} p(e) = \prod_{e \in \delta_i(T)} p(e) \tag{2.10}$$

If the model generates m different derivations for T, we obtain them all, i.e., $\mathcal{D}(T) = \{\delta_1(T), \delta_2(T), \ldots, \delta_m(T)\}$. The probability of T is then obtained by summing the probabilities of all possible derivations in $\mathcal{D}(T)$:

$$P(T) = \sum_{\delta_i \in \mathcal{D}(T)} \delta_i(T) = \sum_{\delta_i \in \mathcal{D}(T)} \prod_{e \in \delta_i(T)} p(e) \tag{2.11}$$

2.3.3 Estimating Probability Distributions

The function p in definition 2.3.1 is the only function needed to characterize a probabilistic tree-generating grammar. The possible choices for p (satisfying equation 2.6) are unlimited, and there is not a single one which is "correct" in any principled way. Nevertheless, choosing a probability distribution at random is also not a recommended strategy. More viable solutions are obtained by defining specific heuristics to guide the choice. Alternatively it is possible to define an independent *objective function* and find the distribution $\hat{p}$ which maximizes it. Again there is not a single heuristic or objective function which is more correct. A standard way to compare these strategies is by analyzing their differences based on empirical testing.

Following different guiding strategies, we will now introduce two well studied *probability estimates* to infer p: the Relative Frequency Estimate, and the Maximum Likelihood Estimate. Finally, we also discuss an alternative general methodology to induce a probabilistic grammar $\mathcal{G}_p$: Bayesian Inference.

Relative Frequency Estimate

The Relative Frequency Estimate (RFE) is the simplest heuristic to derive a probability distribution p_{RFE} over the fragments. It is widely used in statistical models, as for many formalisms there are theoretical and empirical proves of its stability. RFE makes use of two count functions $\langle count_f, count_c \rangle$, which correspond to the frequencies with which fragments and contexts occur in deriving the tree structures in the training treebank $\mathscr{T}$:

$count_f(f_{i,j})$: returns the frequency of fragment token $f_{i,j}$ in $\mathscr{T}$, when occurring in context $c_{i,j}$.

$count_c(c_{i,j})$: returns the frequency of context types $c_{i,j}$ in $\mathscr{T}$.

While $count_f$ is defined over fragment tokens, $count_c$ is defined over context types. This implies that for any j and q such that $c_{i,j} = c_{i,q}$ the following equation should hold:

$$count_c(c_{i,j}) = count_c(c_{i,q}) \tag{2.12}$$

Moreover, the sum of the counts of all fragments sharing the same context type must equal the count of that context:

$$\sum_{\substack{q \ s.t. \\ c_{i,j}=c_{i,q}}} count_f(f_{i,q}) = count_c(c_{i,j}) \tag{2.13}$$

The probability p_{RFE} is defined following the intuition that a fragment $f_{i,j}$ must be used in a certain context $c_{i,j}$ proportionally to the fraction of times it has been extracted in that context:

$$p_{RFE}(f_{i,j}|c_{i,j}) = \frac{count_f(f_{i,j})}{count_c(c_{i,j})} \tag{2.14}$$

This estimate infers a proper probability distribution. In fact for every j equation 2.6 is satisfied:

$$\begin{aligned}
\sum_{\substack{q \ s.t. \\ c_{i,q}=c_{i,j}}} p(f_{i,q}|c_{i,q}) &= \sum_{\substack{q \ s.t. \\ c_{i,q}=c_{i,j}}} \frac{count_f(f_{i,q})}{count_c(c_{i,q})} && (2.15)\\
&= \frac{1}{count_c(c_{i,j})} \cdot \sum_{\substack{q \ s.t. \\ c_{i,q}=c_{i,j}}} count_f(f_{i,q}) && (2.16)\\
&= \frac{count_c(c_{i,j})}{count_c(c_{i,j})} && (2.17)\\
&= 1 && (2.18)
\end{aligned}$$

where equation 2.16 is derived from equation 2.12, and 2.16 from 2.13.

Maximum Likelihood Estimate

The Maximum Likelihood Estimate (MLE) uses an objective function to guide the selection of the probability distribution p of the probabilistic grammar $\mathcal{G}_p$. This objective function is the *likelihood* of the training treebank $\mathscr{T}$ according to $\mathcal{G}_p$, in notation $P(\mathscr{T}|\mathcal{G}_p)$, which is computed as the product of the probability of each tree[19] in $\mathscr{T}$ according to $\mathcal{G}_p$.

$$\begin{aligned} P(\mathscr{T}|\mathcal{G}_p) &= \prod_{T\in\mathscr{T}} P(T|\mathcal{G}_p) && (2.19) \\ &= \prod_{T\in\mathscr{T}} \sum_{\delta_i\in\mathcal{D}(T)} \prod_{e\in\delta_i(T)} p(e) && (2.20) \end{aligned}$$

where equation 2.20 is obtained from equation 2.11. According to this criterion we want to find the probability distribution p_{MLE} which maximizes the likelihood of $\mathscr{T}$:

$$p_{MLE} = \arg\max_p (P(\mathscr{T}|\mathcal{G}_p)) \tag{2.21}$$

Although for many grammar formalisms (e.g., CFG) RFE induces a probability distribution over the tree structures in the observed treebank which maximizes its likelihood, this is not true in the general case (e.g., it is not the case for TSG and other formalisms presenting spurious ambiguities, as will be discussed in §3.6.2). In fact, as equation 2.20 is in the general case a sum of products,[20] it is not possible to maximize the likelihood analytically, i.e., there is no algorithm which is guaranteed to derive such distribution p_{MLE} for a given model. Nevertheless, there exist algorithms that given an initial distribution p_0 are able to re-estimate a sequence of new probability distributions $p_1, p_2, \ldots, p_m$ for which it is guaranteed that the likelihood of the corpus monotonically increases:

$$P(\mathscr{T}|\mathcal{G}_{p_0}) \leq P(\mathscr{T}|\mathcal{G}_{p_1}) \leq P(\mathscr{T}|\mathcal{G}_{p_2}) \leq \ldots \leq P(\mathscr{T}|\mathcal{G}_{p_m}) \tag{2.22}$$

The best known techniques to re-estimate this sequence of probability distributions are EM, i.e., the Expectation-Maximization algorithm (Wu, 1983), and IO, i.e., the Inside-Outside algorithm (Lari and Young, 1990; Pereira and Schabes, 1992; Prescher, 2003). Prescher (2004) gives formal proof that for *string rewriting* PCFG, IO is a dynamic-programming variant of EM. These techniques are commonly studied on *string rewriting*, meaning that the training material refers to a multiset of flat sentences, which are not trees as in our *tree rewriting*

[19] In fact the model is assumed to generate all the sentences in the training corpus independently of each other. This is of course another approximation, since in reality a sentence is often related to the previous one.

[20] It is a simple product of terms when the model uses a single derivation for every tree, such as in the case of PCFG.

case.[21] In string rewriting PCFG, MLE aims at maximizing the likelihood of the training sentences when varying the probabilities of the CFG rules. There is a striking parallelism between the two approaches; in fact, in MLE for string rewriting PCFG, there are multiple CFG derivations for every training sentence, while in our general case there are multiple derivations for every tree in $\mathscr{T}$. We will discuss in more detail one instance of MLE for TSG in §3.6.2.

Bayesian Inference

Bayesian Inference is a framework that derives its methodology from Bayes' rule.

$$\begin{aligned} P(\mathcal{G}_p|\mathscr{T}) &= \frac{P(\mathcal{G}_p, \mathscr{T})}{P(\mathscr{T})} & (2.23) \\ &= \frac{P(\mathscr{T}|\mathcal{G}_p)P(\mathcal{G}_p)}{P(\mathscr{T})} & (2.24) \\ &= \frac{P(\mathscr{T}|\mathcal{G}_p)P(\mathcal{G}_p)}{\sum_{\mathcal{G}'_p} P(\mathscr{T}|\mathcal{G}'_p)P(\mathcal{G}'_p)} & (2.25) \end{aligned}$$

In the last two equations, $P(\mathscr{T}|\mathcal{G}_p)$ is the likelihood, the one defined in the previous section, while $P(\mathcal{G}_p)$ is called the *prior* and specifies how good a chosen probabilistic grammar is, before having observed any data.

In Bayesian inference we are often interested in obtaining the maximum a posteriori distribution $\mathcal{G}_p^{MAP}$ which is defined as:

$$\begin{aligned} \mathcal{G}_p^{MAP} &= \arg\max_{\mathcal{G}_p} P(\mathcal{G}_p|\mathscr{T}) & (2.26) \\ &= \arg\max_{\mathcal{G}_p} \frac{P(\mathscr{T}|\mathcal{G}_p)P(\mathcal{G}_p)}{\sum_{\mathcal{G}'_p} P(\mathscr{T}|\mathcal{G}'_p)P(\mathcal{G}'_p)} & (2.27) \\ &= \arg\max_{\mathcal{G}_p} P(\mathscr{T}|\mathcal{G}_p)P(\mathcal{G}_p) & (2.28) \end{aligned}$$

Goldwater et al. (2009) claim that unconstrained maximum-likelihood estimation is a poor way to choose among probabilistic grammars, and that Bayesian inference provides a more principled approach. In Bayesian modeling, the effect of the likelihood is counter-balanced by choosing a prior distribution that favors for instance simpler grammars. In this case the posterior probability can be seen as a compromise between the likelihood and the simplicity of the grammar, producing a probabilistic grammar which will adequately fit the training data while avoiding over-fitting; in addition the chosen hypothesis will tend to generalize more successfully to novel data.

[21] Similarly to our approach, treebank-based MLE is used in Prescher (2005a) and Petrov (2009).

Previous work describes how to approximate $\mathcal{G}_p^{MAP}$ for different formalisms (see Johnson et al., 2007a; Post and Gildea, 2009; O'Donnell et al., 2009; Cohn et al., 2009, 2010). The difference between our approach and these models is that in our approach we always build a probabilistic instantiation on top of a symbolic grammar, while (the mentioned) Bayesian models do not separate the two processes. This can be in general considered a point of strength of the methodology, since the two processes are bound together using a single objective function. On the other hand, like EM, this methodology uses approximate techniques to find a sequence of grammars that increase in a posteriori probability. These approximations often rely on sampling over the symbolic space. For this reason, we believe that for models having a large symbolic event space (for instance TSG) these sampling techniques will have difficulties reaching an optimal grammars.

2.4 Parsing through Reranking

Given a probabilistic tree-generating grammar $\mathcal{G}_p$, a naive way to find the most probable candidate structure for a given sentence (which maximizes the function in equation 2.9) is to implement the following three steps:

1. Enumerate all possible trees yielding s according to $\mathcal{G}_p$.

2. For every tree T yielding s compute its probability according to $\mathcal{G}_p$.

3. Select the tree with maximum probability.

The naiveness of this approach is located in step 1. In fact, in commonly used generative grammars, the number of trees yielding a certain sentence s grows exponentially with the number of words in s, so it is impossible to list all possible parses for long sentences. The most elegant solution to this problem is to rely on an efficient parser, which uses a compact representation of the possible trees in the grammar (the parse forest) and efficiently select the most probable one. The main drawback of this approach is that the parser needs to be specific to the model. Moreover for certain models it is not guaranteed that the most probable tree can be derived efficiently, i.e., in polynomial time (see also Sima'an, 1996, and §3.6.3).

Alternatively it is possible to restrict the search space by considering only a subset of trees yielding s, and to select the one with maximum probability. This is the approach followed in the reranking methodology that will be now introduced.

Given a probabilistic grammar, the reranking methodology aims at calculating the probability the grammar assigns to a given subset of k different tree structures that yield the same sentence. Such a methodology is able to approximate what a parser implementing the exact same probabilistic grammar would output when presented with the same test sentence. In fact, the chosen structure will tend to match the best analysis obtained with the parser, as the number k of alternatives increases. For this reason we can consider a reranking system a *parser simulator*.

The main shortcoming of a reranking methodology is that it relies on a separate parser to obtain k-best candidates for every test sentence. We will refer to this step as the first phase of the reranking procedure. The parser does not need to be extremely accurate. Ideally, one should let the parser draw structures of a given sentence from a uniform distribution over the space of all possible structures yielding the sentence. This requirement will be relaxed for practical reason of keeping k reasonably small.

One important thing to emphasize is the two-way relation that exists between *decomposing* a given structure into elementary fragments and recomposing them to *generate* the same structure. This fact is essential for understanding that a reranking framework is in fact conceptually identical to the process of extracting fragments and conditioning contexts from the training treebank. In other words, in the reranking framework we compute the probability of an unobserved structure by decomposing it into *elementary fragments* and simulating the reversed process of recomposing them into the original structure.

2.5 Discriminative and Generative models

Generative models of language have characterized earlier approaches in computational linguistics. In more recent times, another class of models, namely *discriminative models*, has been widely employed in the field. The difference between discriminative and generative models has been exhaustively discussed on both theoretical and empirical grounds (Vapnik, 1998; Ng and Jordan, 2001; Xue and Titterington, 2008).

In parsing, the distinction lies in the way the model generalizes from the observed annotated sentences in order to obtain the best structures for a novel sentence. A discriminative model conceives each possible structure of a given sentence as a multitude of features. A feature is generally defined as a binary-valued function, signaling its absence or presence in a given structure. A discriminative model is able to learn from a training treebank, by means of a *machine learning classifier*. This refers to a family of powerful statistical tools that learn the importance of every feature, or feature combination present in a candidate structure, to judge its level of correctness. A trained classifier is then able to assign a score to every candidate according to the features it contains. In the majority of the implemented discriminative parsers, a machine learning classifier can be employed in a full *chart based* implementation, where for every test sentence the entire set of possible tree structures is inspected, and allows the one with highest score to be efficiently selected. For instance Collins and Roark (2004), make use of the perceptron algorithm (Rosenblatt, 1958) on the full phrase-structure parsing forest, and McDonald (2006) employs a Support Vector Machine Classifier (Boser et al., 1992) for dependency parsing.[22]

A generative approach, in contrast, attempts to model the process of generating the structure underlying a sentence, starting from a given set of building elements and compositional operations (as described in this chapter). In this thesis we will only focus on generative models as we are mainly interested in simulating the full process behind the emergence of a sentence together with its underlying structure according to a given model. This process offers the opportunity to better investigate linguistic hypotheses on the generation of tree structures.

The use of reranking methodology in combination with a machine learning classifier is often referred to as *discriminative reranking.* As in our generative reranking approach, a discriminative reranker makes use of another parser in the first phase to produce the k-best candidates for each sentence to parse. This type of approach is common in computational linguistics, although it has been mostly explored for PS parsing (Collins, 2000; Collins and Duffy, 2002; Shen et al., 2003; Daumé III and Marcu, 2004; Collins and Koo, 2005; Charniak and Johnson, 2005; McClosky et al., 2006; Huang, 2008; Fraser et al., 2009; Johnson and Ural, 2010). There are few other studies adopting reranking on other representations (Hall et al., 2007; White and Rajkumar, 2009) and other CL tasks (Shen et al., 2004; Dinarelli et al., 2009).

All these systems differ substantially from the reranking methodology introduced in §2.4: besides being discriminative models, they differ in the training phase: for each training sentence, they are presented with the gold structure together with the set of k-best candidates the support parser provides for the same sentence (including other relevant information serving as extra-features, e.g., position in the k-best list, probability). In this way discriminative parsers are able to learn how to “draw a line” between correct and incorrect decisions of the first-phase parser, keeping the good choices of the parser, while repairing its mistakes. In our approach, instead, reranking is seen as a way to mimick the generative process behind the model, and therefore in the training phase only the gold structures are provided as input to the system.

To summarize the different approaches illustrated so far, and to place in context the contribution of this thesis, Table 2.1 shows how the main parsing models in the CL literature are situated with respect to the learning approach (discriminative, generative), the syntactic representation (PS, DS, TDS), and the search space (chart based, reranking).

[22] The description of machine learning classifiers and discriminative models in general is beyond the scope of this thesis. A more complete discussion on this topic is presented in Smith (2011).

	Discriminative		Generative		
	PS	DS	PS	DS	TDS
Chart Based	Collins and Roark (2004)	Yamada and Matsumoto (2003); Nivre (2003); McDonald (2006); Hall et al. (2006)	Bod (1993); Collins (1996, 1997); Charniak (1996); Matsuzaki et al. (2005); Petrov et al. (2006); **this work (ch. 3)**	Eisner (1996a,b)	
Reranking	Collins (2000); Collins and Duffy (2002); Charniak and Johnson (2005)	Hall et al. (2007)	Bod (2003)†; Borensztajn and Zuidema (2011)	**this work (ch. 4)**	**this work (ch. 5)**

Table 2.1: Comparison of the main parsing models in the CL literature in relation to the current work, with respect to the learning approach (discriminative, generative), the syntactic representation (PS, DS, TDS), and the search space (chart based, reranking).

† Bod makes use of reranking when selecting the shortest derivation from the k most probable parses.

2.6 Conclusions

In this chapter we have presented a unified formalism for describing probabilistic generative models of syntactic tree structures. This approach has allowed us to illustrate a number of known models for parsing syntactic structures within both the phrase-structure and dependency-structure representation, as well as easily defining new ones (e.g., the "Sandwich Insertion Grammar" in example 2.2.6).

This general methodology will allow us to experiment with a number of various generative models of dependency-structure in chapter 4 which are evaluated through the reranking procedure (illustrated in §2.4) as well as defining a completely new generative model for a new tree structure representation (TDS) in chapter 5.

In the following chapter we introduce a novel Data-Oriented Parsing model for phrase-structure, an instantiation of the TSG example 2.2.3 (p. 30).

Chapter 3
Recycling Phrase-Structure Constructions

Though a tree grows so high,
the falling leaves return to the root.

Malay proverb

3.1 Introduction to Phrase-Structure

In this chapter we will deal with phrase-structure (PS) trees. An example of a PS tree is shown in figure 3.1. A PS tree can be formally defined as the repeated partitioning of adjacent words in a sentence into equivalence classes (Hays 1960, p.259; see also figure 1.1). Edges in the PS trees represent *dominance relations* or equivalently *part-whole relations*, in line with modern theories of syntax, first formalized in the work of Chomsky (1956, 1957) (see also §1.2.3). In this example tree, the sentence node S dominates nodes NP (noun phrase) and VP (verb phrase), meaning that NP and VP are *part of* S (sentence), and equivalently that the whole sentence is partitioned in two parts (NP and VP). We will say that NP and VP are *child nodes* of the *parent node* S, with NP preceding VP. As previously illustrated (see the difference between figure 2.2 and 2.3) PS trees do not define precedence relations between parent and child nodes.

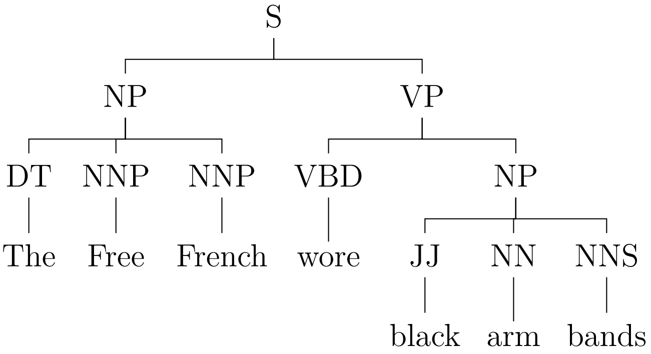


Figure 3.1: Example of a phrase-structure (PS) tree.

One of the main quests in syntax is the identification of the building blocks in natural language sentences (Fillmore et al., 1988; Goldberg, 1995; Kay and Fillmore, 1997). Although standard CFG rewriting rules (see §1.3.1) are often used as a standard choice for practical purposes, it is important to realize that there is a number of ways to decompose a PS tree into its constituent parts. The various alternative range from models generating a CFG rule in several steps, as in head-driven lexicalized models (Magerman, 1995; Collins, 1999), to others utilizing arbitrary large constructions (Bod et al., 2003). In this chapter we will address this last hypothesis, and propose a way to automatically detect the most linguistically relevant syntactic constructions in PS trees. Our methodology follows the basic principle of accepting a syntactic construction as linguistically relevant if there is evidence about its reusability in a representative corpus of annotated trees. We can therefore see our work as a study on construction-recycling, viz. *formulating an hypothesis on how syntactic constructions are built using 'bits and pieces' borrowed from observed sentence structures.*

In the following section we will review previously proposed PS models which are related to our work. Afterwards we will describe our main contributions: first we will present the methodology to efficiently extract reusable constructions from large PS treebanks, and provide qualitative analysis of their features. Secondly, we will present a Data-Oriented Parsing model (Bod et al., 2003) using the extracted constructions for parsing.

3.2 Review of existing PS models

In §1.3.1 and §2.2.3 we have introduced the Probabilistic Context-Free Grammar formalism (PCFG), which represent one of the most basic models for constructing PS trees. As explained in §1.3.2, such grammars impose heavy limitations, as they suffer from both under-generation and over-generation problems. In the past decades novel probabilistic extenstions of CFGs were proposed to overcome such limitations. In the following sections we will review some of the most successful models.

3.2.1 Head-driven models

One of the main innovation in PS parsing was the employment of *head* information to impose lexicalized conditioning on parsing decisions. The head of a phrasal constituent is a central concept in most current grammatical theories (Zwicky, 1985; Hudson, 1987; Corbett et al., 2006) and many syntax-based NLP techniques (e.g., Magerman, 1995; Collins, 1999). The term is used to mark, for any nonterminal node in a PS tree, the specific daughter node that fulfills a special role. Figure 3.2 shows the tree in figure 3.1 with extra head labels (-H).

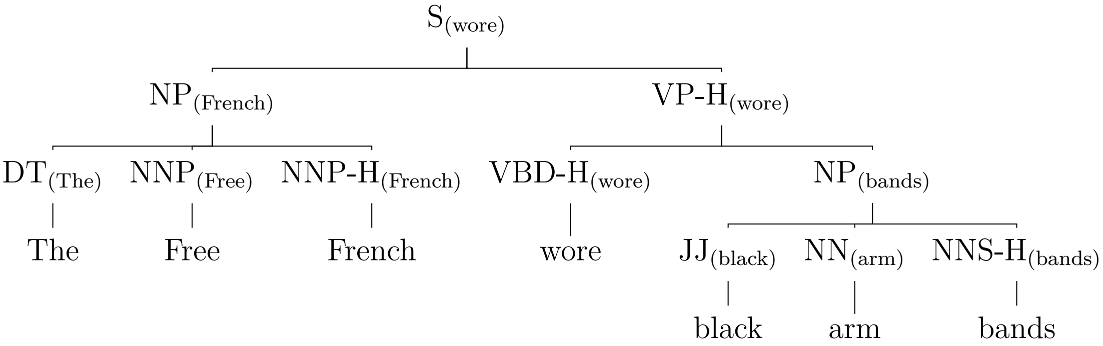


Figure 3.2: Head enrichment (-H) of the tree structure in figure 3.1, and percolation of lexical information (in parenthesis).

As shown in the same figure, internal nodes of the PS tree can be enriched with lexical information which percolates up from the leaves following head annotations. Head-enrichment is usually done through compact heuristic tables as initially proposed by Magerman (1994, 1995). Similar heuristics are used in state of the art head-driven parses (Collins, 1997, 1999; Charniak, 1997).

Collins' models

Collins (1997, 1999) developed three generative parsing models, based on head lexicalization techniques initially proposed by Magerman (1995). Head lexicalization, as introduced above, consists of enriching the internal nodes of a tree with lexical information based on head annotation. This serves to constrain the lexical content on which the probability of a production should depend on, and is therefore fundamental to preventing the over-generation problems of standard PCFG models. The success of Collins' models can be ascribed to the combination of two further innovative techniques: rule factorization, and back-off probability smoothing techniques.

Rule factorization refers to the technique of generating a parse tree attaching one daughter of a node at a time (*horizontal markovization*). More specifically, the head daughter is generated first, conditioned on the parent node; afterwards left and right siblings are produced conditioned on the head sibling, the parent node, and a certain number of previously generated daughters.[1] This is similar to the model illustrated in example 2.2.4, where each CFG production rule is factorized in several steps according to an n^{th} order markovian assumption. So for instance if we want to compute the probability to generate the top rule $\text{S}_{(\text{wore})} \rightarrow \text{NP}_{(\text{French})}\ \text{VP}_{(\text{wore})}$ of the tree in figure 3.2, with $n = 1$ we will have:

$$\begin{array}{lll}
\text{head daughter} & P_h(VP_{(\text{wore})}|S, wore) & \times \\
\text{first left sibling} & P_l(NP_{(\text{French})}|S, VP, wore, \oslash) & \times \\
\text{no more left siblings} & P_l(\oplus|S, VP, wore, NP_{(\text{French})}) & \times \\
\text{no more right siblings} & P_r(\oplus|S, VP, wore, \oslash) &
\end{array}$$

where $\oslash$ is the null symbol and $\oplus$ the stop symbol as defined in 2.2. Rule factorization is very effective to solve the under-generation problems of standard PCFG models, as the new model is capable of producing CFG rules not observed in the training set.

Back-off probability smoothing techniques are used in order to reduce complex probabilistic events estimations involving a large number of elements in the conditioning context (as above), to a sequence of more simple and general formulas. This is done in order to handle a great number of cases in which the complex events were not observed in the training corpus. For instance if we

[1] Collins introduces other elements in the conditioning context, such as a notion of distance between a node and the head sibling (model 1), whether a node is an argument or an adjunction (model 2), gaps for expression involving wh-movement (model 3). These advanced features are not illustrated here for simplicity reasons.

have a probabilistic model estimating event e given conditioning context c as $P(e|c) = P(A|BCDE)$ we could simplify it imposing 3 backing-off levels (l_1, l_2, l_3) in which we ignore an increasingly big number of elements in the conditioning context, e.g., we might choose to ignore B and D first, and C in a second step:

$$\begin{aligned} l_1 &= P(A|BCDE) \\ l_2 &= P(A|CE) \\ l_3 &= P(A|E) \end{aligned}$$

The three distributions are estimated via relative frequency from the training corpus and then interpolated to obtain a smoothed probability estimation: $P(e|c) = \lambda_1 l_1 + (1-\lambda_1)(\lambda_2 l_2 + (1-\lambda_2) l_3)$, with λ_i chosen empirically to maximize accuracy, imposing $0 \leq \lambda_i \leq 1$. This smoothing technique is also refer to as *deleted interpolation* (for more details see also Eisner 1996a; Bikel 2004a and Appendix B).

Charniak's models

The model in Charniak (1997) is similar to Collins (1997, 1999) except that it uses slightly different generative models. In particular the generation of a constituent is decomposed in two parts: initially the lexical head of the constituent is determined based on the dependency relations with its parent node; as a second step, non-lexicalized part of a rule is predicted (the category of the constituent together with its CFG expansion) conditioned on the parent node and its head-word. Moreover, the implementation of deleted estimation makes use of statistics over clustering of words instead of single words. A more advanced parser is presented in Charniak (1999). The model is still generative, and conceptually similar to the previous model. The main difference is in the definition of a much larger set of features in the history of each generative event, and the use of a model inspired by the maximum-entropy framework (Berger et al., 1996; Ratnaparkhi, 1999), which makes it more robust and easier to train.

3.2.2 State-Splitting Models

With state-splitting approaches we refer to recent techniques (Matsuzaki et al., 2005; Prescher, 2005b; Petrov et al., 2006; Petrov, 2009) which automatically refine standard PCFG by introducing new artificial rules obtained by splitting each internal category A to a certain number of artificial subcategories $A_1, A_2, \ldots, A_n$, and estimating a new probability distribution over trees. In a broad sense, other node-enrichment techniques (Goodman, 1997; Johnson, 1998; Klein and Manning, 2003) can be also considered as instances of state splitting models, because each node in every tree is split into more refined categories based on contextual features (e.g., head, parent information). Nevertheless we would like here to draw some separation between the two approaches. In fact, recent state-splitting models strongly rely on automatic methods of enrichment rather than using manual annotations or pre-defined contextual features. The other important difference is that while in manual or feature-based enrichment models each parse tree is mapped into a specific enriched structure, in state-splitting models, such as Petrov (2009), each parse tree is mapped to multiple enriched structures. As a result, the probability of a given parse tree for a test sentence is the sum of the probabilities of all the refined structures allowed by the grammar for that parse tree. As we will see it in the later section, this characteristic is shared with Data-Oriented Parsing Models.

Although there exist a number of automatically induced state-splitting models, we will only present the work of Petrov (2009) as it is methodologically simpler (with respect to e.g., Matsuzaki et al., 2005) and the one achieving better results.

Berkeley parser

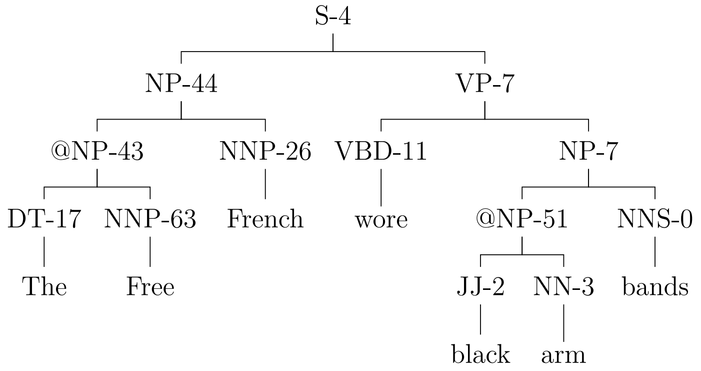
Petrov (2009) developed a fast and accurate parser based on state-splitting. The learning of the refined grammar is done automatically and incrementally, alternating splitting and merging phases. In a split phase every category is forked in two subcategories and a re-estimation of the probabilities of rules is performed in order to maximize the probability of the training corpus using the Expectation-Maximization algorithm. In the merge phase, the model checks to which extent each splitting contributes to increase the likelihood of the data; a certain splitting is undone if its contribution is found to be negligible. Figure 3.3 shows the tree in figure 3.1 with the refined categories after the 6th iteration of the Berkeley model.

The use of refined categories effectively helps the model in solving the over-generation problem of the underlying PCFG, as it imposes stronger constraints on the possible ways of extending each internal node of a parse tree. The under-generation problem is solved by using artificial binarization of the treebank (marked in the tree in figure 3.3 with '@'), which is a way to implicitly encode *horizontal markovization* in the model as in Collins (1997, 1999) (see also §3.7).

3.3 Data-Oriented Parsing

In this section we present a novel approach to Data-Oriented Parsing (DOP). As in previous DOP models (Bod, 2001b; Bansal and Klein, 2010), our parser utilizes syntactic fragments of arbitrary size from a treebank to analyze new sentences. As the number of fragments which can be extracted from a large treebank is extremely large, previous approaches have resorted to explicitly extracting a random sample of fragments (e.g., Bod, 2001b), or implicitly representing them all in a compact grammar (e.g., Bansal and Klein, 2010).

The main contribution of our work is to propose a more principled-based approach for explicitly extracting a relatively small but representative set of



Category	Words
DT-17	The, A
NNP-26	Commission, Committee, Association, Department, Museum, Revolution, Office, ...
VBD-11	sent, exercised, received, retained, completed, followed, made, ...
JJ-2	ceramic, young, daily, imperial, full, ...

Figure 3.3: Example of a phrase-structure (PS) tree according to Berkeley model, including binarization nodes marked with '@' (left), and a sample of the set of words yielded by some of the refined categories used in the tree.

fragments from a treebank, i.e., those which are encountered at least twice in the treebank, for which there is evidence about their reusability. The extracted fragment-grammar can be employed as the symbolic backbone of several probabilistic generative models.

3.3.1 Introduction

Data-Oriented Parsing (DOP) is an approach to wide-coverage parsing based on assigning structures to new sentences using fragments of variable size from a treebank (see example 2.2.3). It was first proposed by Remko Scha in 1990 and formalized by Bod (1992), and preceded many developments in statistical parsing (e.g., the "treebank grammars" of Charniak 1997). It is related to the linguistic theory of construction-grammars (Fillmore et al., 1988), in which constructions are not limited to single CFG productions but may span over several nested syntactic phrases.

A rich literature on DOP has emerged since, yielding state-of-the-art results on the Penn treebank benchmark test (Bod, 2001b; Bansal and Klein, 2010) and inspiring developments in related frameworks including tree kernels (Collins and Duffy, 2001, 2002), reranking (Charniak and Johnson, 2005) and Bayesian adaptor and fragment grammars (e.g., Johnson et al., 2007b; O'Donnell et al., 2009; Cohn et al., 2010).

By formalizing the idea of using large fragments of earlier language experience to analyze new sentences, DOP captures an important property of language cognition that has shaped natural language (Lieven et al., 2003; Arnon, 2009; Arnon and Snider, 2010). It therefore complements approaches that have focused on properties like lexicalization or incrementality, and might bring supplementary strengths into other NLP tasks.

In this section we present a novel DOP model (Double-DOP) in which we extract a restricted yet representative subset of fragments: those recurring at least twice in the treebank. The explicit representation of the fragments allows us to derive simple ways of estimating probabilistic models on top of the symbolic grammar. This and other implementation choices aim at making the methodology transparent and easily replicable. The accuracy of Double-DOP is well within the range of state-of-the-art parsers currently used in other NLP-tasks, while offering the additional benefits of a simple generative probability model and an explicit representation of grammatical constructions.

We present a number of technical contributions: (i) a way to restrict the set of fragments to only those that occur multiple times in the train set, (ii) a transform-backtransform approach that allows us to use off-the-shelf PCFG parsing techniques, and (iii) a way to integrate DOP with recent state-splitting approaches (Petrov et al., 2006), yielding an even more accurate parser and a better understanding of the relation between DOP and state-splitting.

In line with the symbolic/probabilistic separation conducted in the previous chapter we will first introduce the symbolic backbone of our DOP model (§3.4) and later on describe its stochastic instantiation (§3.6).

3.4 The symbolic backbone

The basic idea behind DOP is to allow arbitrarily large fragments from a treebank to be the elementary units of production of the grammar.

A fragment can be formally described as a subtree of a PS tree, where for each node in the fragment either all or none of its child nodes are present in the subtree. Fragments can be combined through the *substitution operation* to obtain the PS tree of a new sentence. Figure 3.4 shows an example of how to obtain the complete syntactic tree in figure 3.1 by combining three elementary fragments. As in previous work, two fragments f_i and f_j can be combined ($f_i \circ f_j$) only if the leftmost substitution site $X\downarrow$ in f_i has the same label as the root node of f_j; in this case the resulting tree will correspond to f_i with f_j replacing X. The DOP formalism is discussed in detail in e.g., Bod et al. (2003).

3.4.1 Explicit vs. Implicit Grammars

The first step to build a DOP model is to define its symbolic grammar, i.e., the set of elementary fragments in the model. Early versions of DOP (e.g., Bod et al., 2003) aimed at extracting all subtrees of all trees in the treebank. The total number of subtrees, however, is prohibitively large for non-trivial treebanks: it grows exponentially with the length of the sentences, yielding the astronomically

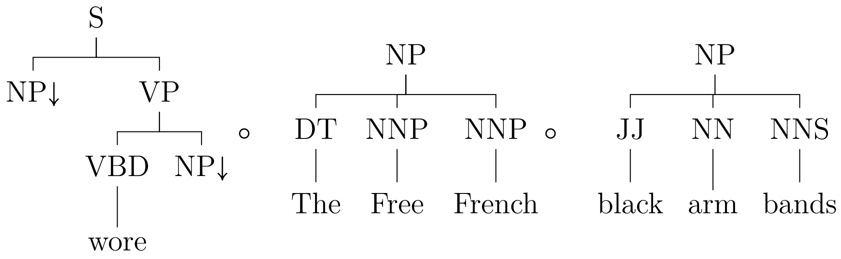


Figure 3.4: An example of a derivation of the syntactic structure in figure 3.1 obtained combining three elementary fragments by means of the substitution operation ∘. Substitution sites are marked with the symbol ↓.

large number of approximately 10^{46} for section 2-21 of the Penn WSJ treebank (see §3.5.2). DOP models that work with an explicit extraction of subtrees (Bod, 1992, 2001b; Zuidema, 2007), thus resorted to random sampling or to specific heuristics to filter fragments based on their features (e.g., depth, number of lexical or nonterminal elements in frontier). But, as we will show in section 3.5.2, the majority of the constructions extracted by most of these sampling techniques occur only once in the training corpus. For instance the chance that a randomly extracted large fragment is reusable in a different structure is, for all practical purposes, 0. At the same time, any sampling technique will most likely fail to extract many relevant syntactic productions.

Goodman transformation Later DOP models have used the Goodman transformation (Goodman, 1996, 2003) to obtain a compact representation of all fragments in the treebank (Bod, 2003; Bansal and Klein, 2010). The transformation was defined for some versions of DOP to an equivalent PCFG-based model, with the number of rules extracted from each parse tree being linear in the size of the trees.

This is possible by means of enriching the treebank with a unique index i for every node N (becoming N_i) in the trees. Assuming that the treebank is binarized, for every PCFG rule $A_j \rightarrow B_k C_l$, 8 rules are extracted:

$$\begin{array}{lcl@{\qquad\qquad}lcl}
A_j & \rightarrow & BC & A & \rightarrow & BC \\
A_j & \rightarrow & B_k C & A & \rightarrow & B_k C \\
A_j & \rightarrow & B C_l & A & \rightarrow & B C_l \\
A_j & \rightarrow & B_k C_l & A & \rightarrow & B_k C_l
\end{array}$$

Here we have reported only the symbolic part of the conversion. Goodman defines also specific weights for each transformed rule, so that the model generates subderivations with the same probabilities as various probabilistic DOP models. This transform, is used in most recent DOP parsers (e.g., Bod, 2003; Bansal and Klein, 2010). The grammar represents larger fragments only *implicitly*, by means of the unique indices which function as locks, constraining every indexed node N_i to the CFG production where it occurs in the training corpus.

Bod has argued for the Goodman transform as the solution to the computational challenges of DOP (e.g., Bod, 2003); it is important to realize, however, that the resulting grammars are still very large: WSJ sections 2-21 yield about 7.8×10^6 rules in the basic version of Goodman's transform.[2] Moreover, the transformed grammars differ from untransformed DOP grammars in that larger fragments are no longer explicitly represented. Rather, information about their frequency is distributed over many CFG-rules: if a construction occurs n times and contains m context-free productions, Goodman's transform uses the weights of $7nm + m$ rules to encode this fact. Thus, the information that the idiomatic fragment *(PP (IN "out") (PP (IN "of") (NP (NN "town")))))* occurs 3 times in WSJ sections 2-21, is distributed over 132 rules. This way, an attractive feature of DOP, viz. the explicit representation of the 'productive units' of language, is lost.[3]

3.5 Finding Recurring Fragments

In the current work we return to the first approach, and explicitly extract a subset of fragments from the training treebank. We believe that an explicit representation of fragments in a grammar could provide great advantages for better understanding the model, since it allows us to reproduce the process by which syntactic pieces are combined to form sentential structures. Moreover in several applications it is conceptually easier to deal with explicit grammar productions, as for instance in machine translation systems which need to resolve a mapping between syntactic productions of two distinct languages.

Unfortunately, as explained above, explicit fragment grammars can grow extremely large in size. To limit the fragment set size, we use a simple but heretofore unexplored constraint: we extract only those fragments that occur *two or more times* in the treebank. This intuition follows the natural assumption, common in many current linguistic theories, to consider a construction linguistically relevant if there is empirical evidence about its reusability in a representative corpus. Thus with Double-DOP we hope to overcome some of the limitations of previous probabilistic extensions of CFGs as well as the problem of efficiency and

[2] About 10^6 lexical and 6.8×10^6 internal rules. This has been calculated from the treebank which was binarized as in our experiments.

[3] Bansal and Klein (2010) address this issue for contiguous constructions by extending the Goodman transform with a 'Packed Graph Encoding' for fragments that "bottom out in terminals". However, constructions with variable slots, such as *whether S or not*, are left unchanged.

reproducibility of previous DOP models based on the extraction of an explicit fragment-grammars.

The idea of working with fragments occurring two or more times in the treebank, can be seen as related to the *held-out estimation* procedure used in DOP* (Zollmann and Sima'an, 2005). In this work, fragments are extracted from a subset of the training treebank (the extraction corpus, EC) and their weights are estimated on the remaining part (the held-out corpus, HC) in order to maximize its likelihood. As a result, the fragments which obtain non-zero weights in the final grammar are those occurring both in EC and HC. If HC consists of a single tree (leave-one-out), and by repeating the procedure such that each tree in the training corpus is used once as the held-out data (*K-fold cross-validation*), the final grammar will contain exactly those fragments occurring at least twice in the training corpus.

In the remaining of this section we describe the algorithm for extracting the set of recurring fragments from a treebank, which will constitute our symbolic grammar, and illustrate some properties of the selected constructions. In §3.6 and §3.7 we will propose a number of probabilistic instantiations of the underlying symbolic model.

3.5.1 The search algorithm

Extracting recurring fragments in a large treebank is not a trivial task: a naive approach that filters a complete table of fragments together with their frequencies would fail because that set, in a reasonably sized treebank, is astronomically large. Instead, we use an efficient kernel-based algorithm, which is conceptually similar to previously proposed methods using this technique (Collins and Duffy, 2001, 2002; Moschitti, 2006). The main difference, however, is that, while in previous studies kernels are mainly used to numerically quantify the similarity between two trees, in the current work we are interested in identifying the actual constructions they share, i.e., the common largest (or maximal) fragments.

Our search algorithm[4] iterates over every pair of trees in the treebank and looks for common maximal fragments. More precisely we extract only the largest shared fragments for all pairs of trees in the treebank. All subtrees of these extracted fragments necessarily also occur at least twice, but they are only explicitly represented in our extracted set if they happen to form a largest shared fragment from another pair of trees. Hence, if a large tree occurs twice in the treebank the algorithm will extract from this pair only the full tree as a fragment and not all its (exponentially many) subtrees.

Figure 3.5 shows an example of a pair of trees $\langle \alpha, \beta \rangle$ being compared. All the non-terminal nodes of the two trees are indexed following a depth-first ordering

[4] The implemented software for extracting recurring fragments (`FragmentSeeker`) is available at `http://staff.science.uva.nl/~fsangati/`.

(as in figure 2.9). The algorithm builds a chart $\mathcal{M}$ with one column for every indexed non-terminal node α_i in α, and one row for every indexed non-terminal node β_j in β. Each cell $\mathcal{M}\langle i, j\rangle$ identifies a set of indices corresponding to the largest fragment in common between the two trees starting from α_i and β_j. This set is empty if α_i and β_j differ in their labels, or they do not have the same list of child nodes. Otherwise (if both the labels and the lists of children match) the set is computed recursively as follows:

$$\mathcal{M}\langle i, j\rangle = \{\alpha_i\} \bigcup \left(\bigcup_{c=\{1,2,\ldots,|ch(\alpha)|\}} \mathcal{M}\langle ch(\alpha_i, c), ch(\beta_j, c)\rangle \right) \tag{3.1}$$

where $ch(\alpha)$ returns the indices of α's children, and $ch(\alpha, c)$ the index of its c^{th} child. The procedure to recursively compute the maximal shared fragments between two nodes (N_i, N_j) of a pair of PS trees is described in the algorithm in figure 3.5.

After filling the chart, the algorithm extracts the set of recurring fragments, and stores them in a table to keep track of their counts. This is done by converting back each fragment implicitly defined in every cell-set,[5] and filtering out those that are properly contained in others.[6]

The time complexity of the overall procedure, is $O(n^2 \cdot m^2)$ where n is the size of the treebank and m the number of nodes in the biggest tree of the corpus.[7] In terms of space the number of maximal fragments which are extracted for every pair of trees is in the worst case m^2.

3.5.2 A case study on the Penn WSJ

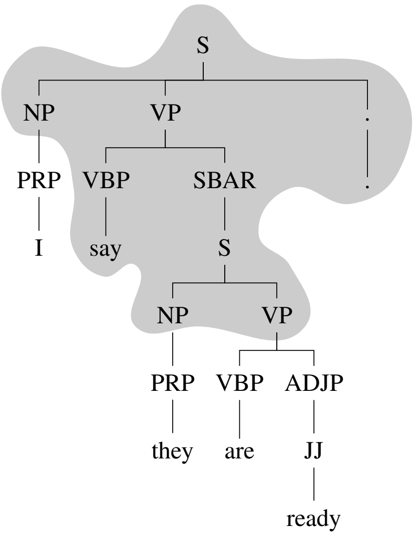
In this section we describe some statistics derived from testing our extraction procedure on the Penn WSJ corpus Marcus et al. (1993). We have restricted the treebank to the 39,832 structures of sections 2-21 after removing null productions and traces. Differently from the preprocessing in later experiments the treebank here is not binarized and functional tags (SBJ, TMP, etc...) are kept.

Figure 3.6 reports some statistics on the set of all maximal fragments which are extracted from the treebank. The total number of extracted fragments types

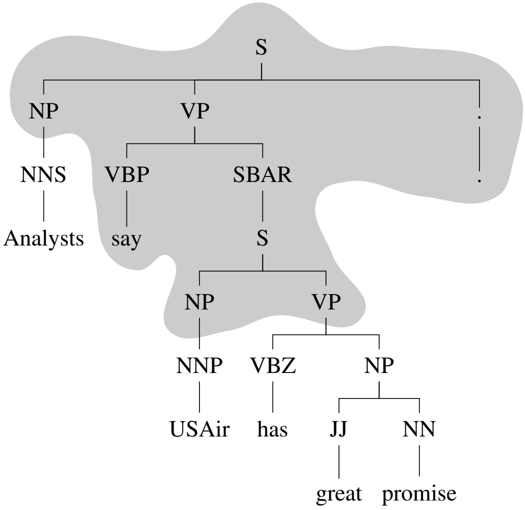
[5] A cell-set containing a single index corresponds to the fragment including the node with that index together with all its children.

[6] In a second pass over the treebank, exact counts can be obtained for each fragment in the extracted set. However, the approximate counts returned by the extraction algorithm are extremely close to the exact values.

[7] In terms of empirical computation time, using a 2.5 GHz processor machine, our program takes about around 50-CPU hours for WSJ (sections 2-21). Although our code could still be optimized further, it does already allow for running the job on N CPUs in parallel, reducing the time required by a factor N (less than 3 hours with 16-CPUs). Computing time of course becomes problematic for very large treebanks, but we are optimistic about the effectiveness of approximate strategies when training sets becomes very large.



α



β

	S	NP	PRP	VP	VBP	SBAR	S	NP	PRP	VP	VBP	ADJP	JJ	.
S	×						×							
NP		×						×						
NNS														
VP				×						×				
VBP					×						×			
SBAR						×								
S	×						×							
NP		×						×						
NNP														
VP				×										
VBZ														
NP														
JJ													×	
NN														
.														×

```
Algorithm: $ExtractMaxFragment(N_i, N_j)$
Input: two PS nodes $(N_i, N_j)$
Output: a set of nodes representing the largest
        fragment rooted in $N_i$ and $N_j$
begin
    if $N_i \neq N_j$ then return $\{\}$;
    $NodeSet \leftarrow \{N_i\}$;
    if $N_i.daughters = N_j.daughters$ then
        for $d \in (1, 2, \ldots, N_i.daughters.size)$ do
            $D_i \leftarrow N_i[d]$;
            $D_j \leftarrow N_j[d]$;
            $NodeSetD \leftarrow$
            $ExtractMaxFragment(D_i, D_j)$;
            $NodeSet.union(NodeSetD)$;
    return $NodeSet$;
```

Figure 3.5: Above: example of two trees sharing a single maximum fragment, highlighted in the two trees. Below: the chart used in the algorithm to extract all maximum fragments shared between the two trees, and the pseudocode of the algorithm used for extracting the largest fragment rooted in two nodes of two PS trees. In the chart, the highlighted cells are the ones which contribute to extracting the shared fragment. The cells marked with a cross are those for which the corresponding nodes in the two tree have equivalent labels.

is 527,217, and their distribution with respect to their depths, reported on the graph of the same figure, shows that fragments of depth 3 and 4 are the most abundant recurring fragments in the corpus. Their frequency distribution follows Zipf's law as shown in figure 3.7.

Figure 3.8 shows the distribution of the total number of fragments tokens which are present in the same treebank, with respect to their depths and maximum branching. The maximum branching of a fragment corresponds to the maximum number of daughters of the most prolific node. From the figure we can see that this variable is the primary factor to affect the number of subtrees present in a tree structure. The total number of fragments without any restriction in depth and branching, is estimated to be $8.7 \cdot 10^{46}$.[8] It follows that the portion of fragment tokens which are recurring in the treebank (the shaded area in the graph) is an extremely small fraction[9] of all possible fragments.

When looking at the extracted fragments we ask if we could have predicted which fragments occur twice or more. Figure 3.10 attempts to tackle this question by reporting some statistics on the extracted fragments. In particular it shows the distribution of the recurrent fragments types according to several features: depth, number of words, and number of substitution sites. Not surprisingly most of the frequent recurring fragments have low values for these features: the majority of fragments are rather small with a limited number of words or substitution sites in the frontier. Nevertheless there is still a significant portion of fragments, in the tail of the distribution, with more than 10 words (or substitution sites). Since the space of all fragments with such characteristics is enormously large, selecting big recurring fragments using random sampling technique is like finding a needle in a haystack. Hence, random sampling processes (like Bod, 2001b), will tend to represent frequent recurring constructions such as *from NP to NP* or *whether S or not*, together with infrequent overspecialized fragments like *from Houston to NP*, while missing large generic constructions such as *everything you always wanted to know about NP but were afraid to ask*. These large constructions are excluded completely by models that only allow elementary trees up to a certain depth (typically 4 or 5) into the symbolic grammar (Zollmann and Sima'an, 2005; Zuidema, 2007; Borensztajn et al., 2009), or only elementary trees with exactly one lexical anchor (Sangati and Zuidema, 2009).

Finally, in figure 3.9 we have reported the most frequently recurring fragments in the WSJ containing the verb "*say*" (when it is a present tense verb). This kind of statistics, can give an insight on the specific template constructions of this particular verb.

[8] This number is calculated by summing the number of subtrees $\omega(N_i)$ rooted in every node N_i of every tree in the treebank. $\omega(N_i)$ is calculated recursively: $\omega(N_i) = \prod_{D \in d(N_i)} (\omega(D) + 1)$, where $d(N_i)$ returns the list of daughter nodes of N_i. If N is a terminal node $\omega(N) = 0$.

[9] In the graph, the shaded area representing this tiny fraction ($7.1 \cdot 10^{-41}$), is visible because of the logarithmic scale in the y-axes.

Depth	Types	Tokens
1	27,893	1,570,869
2	86,512	1,549,523
3	138,709	1,428,777
4	128,927	923,315
5	83,218	455,448
6	40,524	179,548
7	14,849	52,424
8	4,677	14,133
9	1,343	3,692
10	398	951
11	96	213
12	39	95
13	14	28
14	6	16
15	4	8
16	3	8
17	2	5
18	2	4
20	1	2
Total	527,217	6,179,059

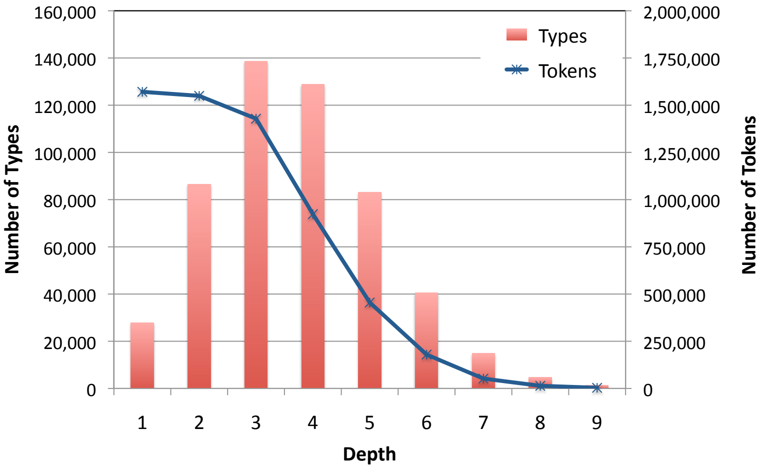


Figure 3.6: Distribution of the types and tokens frequencies of the recurring maximal fragment with respect to their depths. Fragments are extracted from sections 2-21 of the Penn WSJ Treebank.

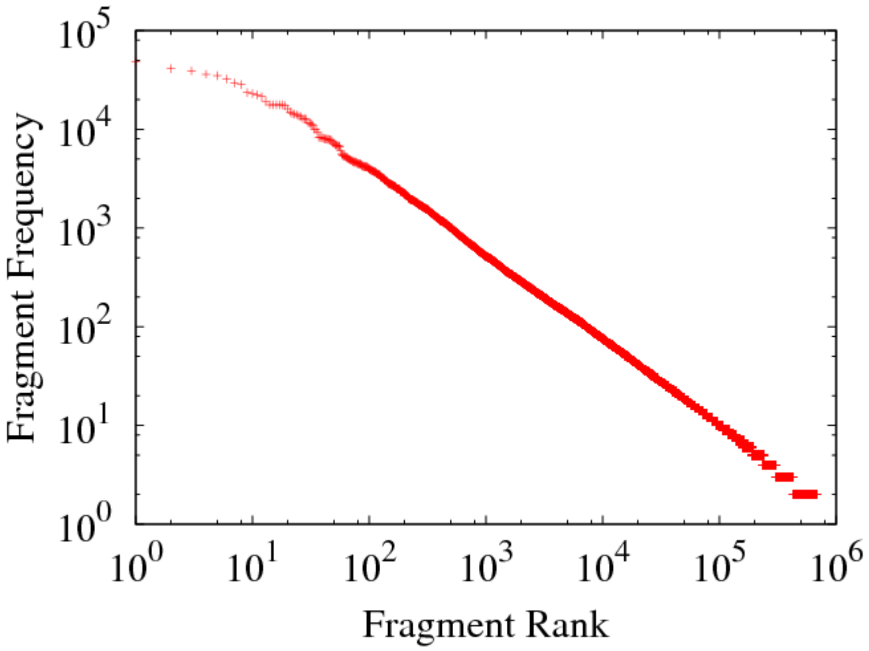


Figure 3.7: Zipf distribution of the recurring fragments extracted from section 02-21 of the Penn WSJ Treebank.

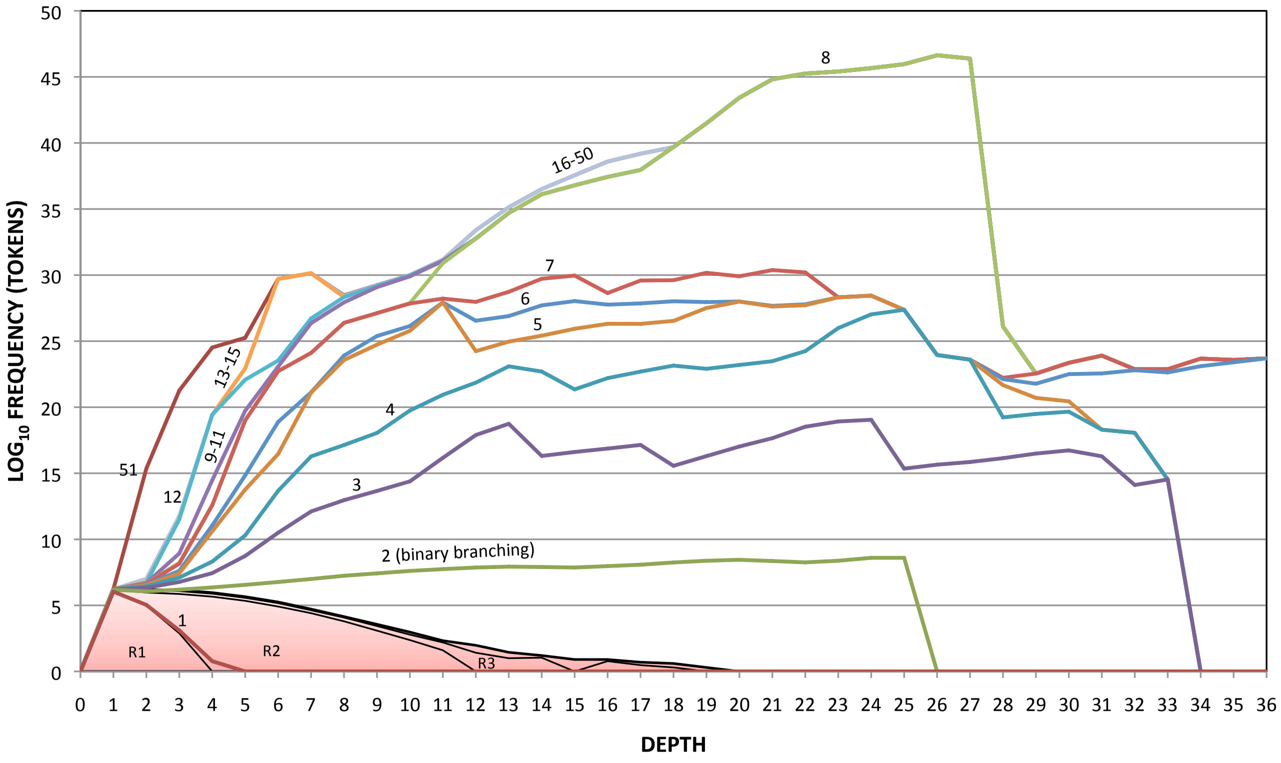


Figure 3.8: Distribution of the total number of fragments tokens which are present in the trees of sections 2-21 of the Penn WSJ Treebank, with respect to the fragment depths (horizontal axes, with depth=1 standing for single tree nodes, and depth=2 for CFG-rules). Every line corresponds to the total number of fragments at different depth values, when limiting the maximum branching of the fragments to a certain constant (the number reported close to the line). The maximum branching of a fragment is defined to be the maximum number of daughters in the most prolific node of the fragment. The shaded area at the bottom of the graph represents the portion of recurring fragments which we extract from the treebank (the maximal fragments shared between at least two trees). R1 is the sub-portion including fragments with only unary branching (almost all are recurring fragments), and similarly R2 and R3 represent sub-portions with fragments with maximum branching 2 and 3.

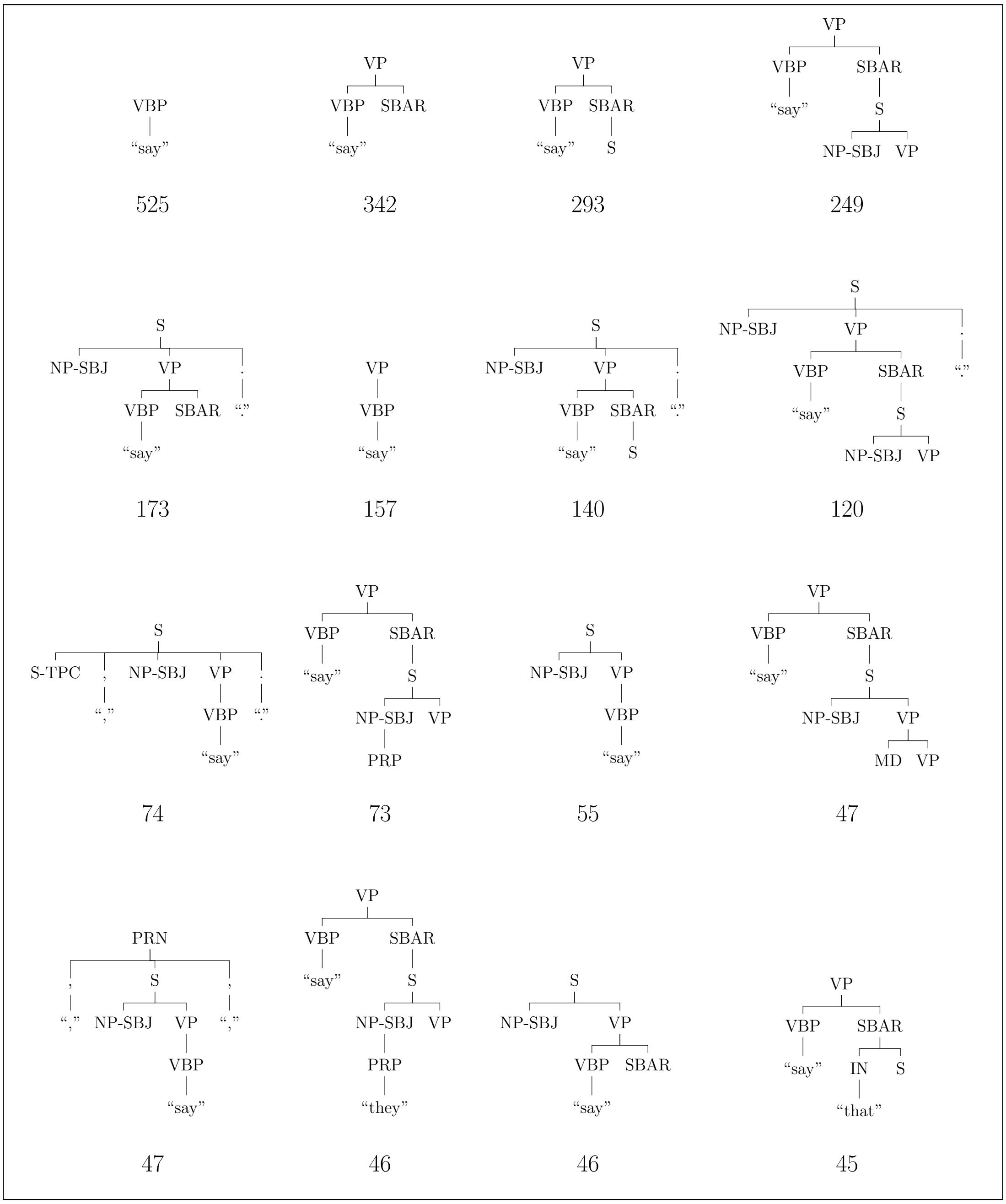

VBP “say”
525
VP VBP SBAR “say”
342
VP VBP SBAR “say” S
293
VP VBP SBAR “say” S NP-SBJ VP
249
S NP-SBJ VP . VBP SBAR “.” “say”
173
VP VBP “say”
157
S NP-SBJ VP . VBP SBAR “.” “say” S
140
S NP-SBJ VP . VBP SBAR “.” “say” S NP-SBJ VP
120
S S-TPC , NP-SBJ VP . “,” VBP “.” “say”
74
VP VBP SBAR “say” S NP-SBJ VP PRP
73
S NP-SBJ VP VBP “say”
55
VP VBP SBAR “say” S NP-SBJ VP MD VP
47
PRN , S , “,” NP-SBJ VP “,” VBP “say”
47
VP VBP SBAR “say” S NP-SBJ VP PRP “they”
46
S NP-SBJ VP VBP SBAR “say”
46
VP VBP SBAR “say” IN S “that”
45

Figure 3.9: The most frequent fragments containing the verb *say*, when it is a present tense verb (VBP). Below each fragment we report the exact frequency with which it occurs in the WSJ sec 02-21. For example, the second fragment at the top of the figure (occurring 342 times) illustrates a specific template construction of the verb, which requires a relative or subordinate clause (SBAR) as first and only argument to its right; this specific construction accounts for 65% of the occurrences of “say”.

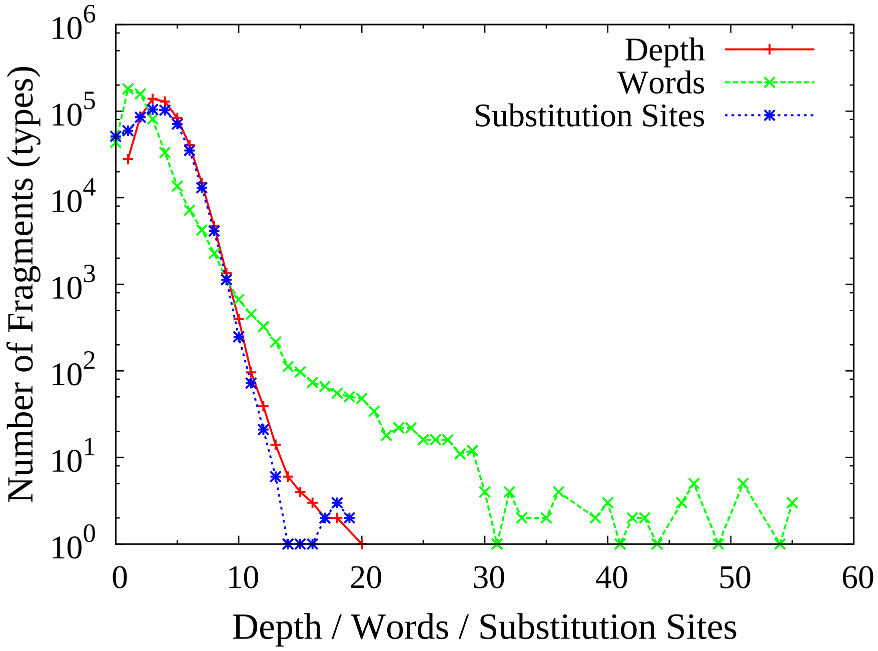


Figure 3.10: Distribution of the recurring fragments types according to several features: depth, number of words, and number of substitution sites. Their corresponding curves peak at 3 (depth), 1 (words), and 3 (substitution sites).

3.6 The probability model

The set of recurring fragments can be used as the symbolic backbone of a probabilistic parser. Like CFGs, our symbolic model produces extremely many parse trees for a given test sentence.[10] We therefore need to disambiguate between the possible parses by means of a probability model that assigns probabilities to fragments, and defines a proper distribution over the set of possible full parse trees.

For every nonterminal X in our grammar we want to have (see also equation 2.6):

$$\sum_{f \in F_X} p(f) = 1 \tag{3.2}$$

where F_X is the set of fragments in our symbolic grammar rooted in X. Adding probabilities turns our grammars into probabilistic tree substitution grammars (PTSGs).

[10] In particular our Double-DOP grammar extracted from a treebank has the same strong generative power of the subsuming CFG: it generate the same tree structures of the CFG extracted from the same treebank.

A derivation $d = f_1, f_2, \ldots, f_n$ of T is a sequence of fragments that through left-most substitution produces T. The probability of a derivation is computed as the product of the probabilities of each of its fragments.

$$P(d) = \prod_{f \in d} p(f) \tag{3.3}$$

In §3.6.2 we describe ways of obtaining different probability distributions over the fragments in our grammar, but we first illustrate how to use standard PCFG parsing techniques given a probabilistic model.

3.6.1 Parsing

It is possible to define a simple transform of our probabilistic fragment grammar, such that off-the-shelf parsers can be used. In order to perform the PTSG/PCFG conversion, every fragment in our grammar must be mapped to a CFG rule which will keep the same probability as the original fragment. The corresponding rule will have as the left hand side the root of the fragment and as the right hand side its yield, i.e., a sequence of terminals and non-terminals (substitution sites).

It might occur that several fragments are mapped to the same CFG rule.[11] These are interesting cases of syntactic ambiguity as shown in figure 3.11. In order to resolve this problem we need to map each ambiguous fragment to two unique CFG rules chained by a unique artificial node, as shown at the bottom of the same figure. To the first CFG rule in the chain we assign the probability of the fragment, while the second will receive probability 1, so the product gives back the original probability. The ambiguous and unambiguous PTSG/PCFG mappings need to be stored in a table, in order to convert back the compressed CFG derivations to the original PTSG model after parsing.

Such a transformed PCFG will generate the same derivations as the original PTSG grammar with identical probabilities. The resulting grammar is more economic than previously proposed transformations: it needs only one PCFG rules per elementary tree, and two for the ambiguous fragment. Previous approaches (e.g., Zuidema, 2007, and others) use a separate rule for each CFG-production inside an elementary tree.

In our experiment we use a standard PCFG parser to produce a list of k-best Viterbi derivations. These, in turn, will be used to maximize possible objectives as described in section 3.6.3.

[11] In our binarized treebank we have 31,465 fragments types that are ambiguous in this sense (about 6% of the total number of extracted fragments). On average 2.05 ambiguous fragments map to the same CFG rule.

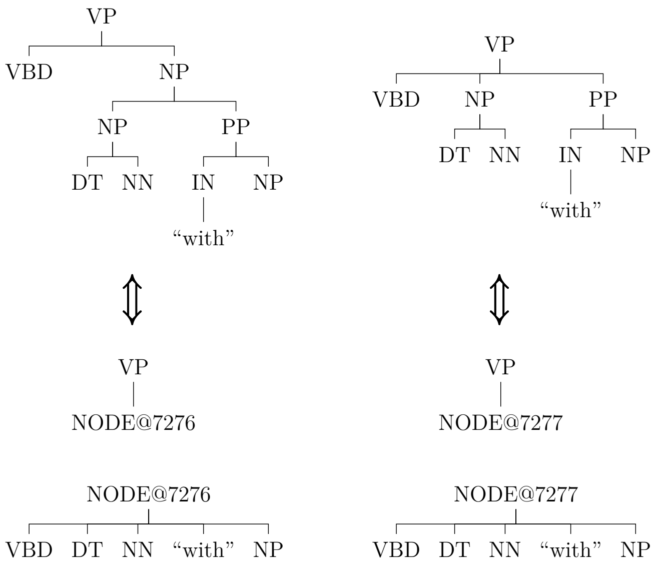


Figure 3.11: Above: example of 2 ambiguous fragments which would map to the same CFG rule VP → VBD DT NN "with" NP. The first fragment occurs 5 times in the training treebank, (e.g., in the sentence *was an executive with a manufacturing concern*) while the second fragment occurs 4 times (e.g., in the sentence *began this campaign with such high hopes*). Below: the two pairs of CFG rules that are used to map the two fragments to separate CFG derivations.

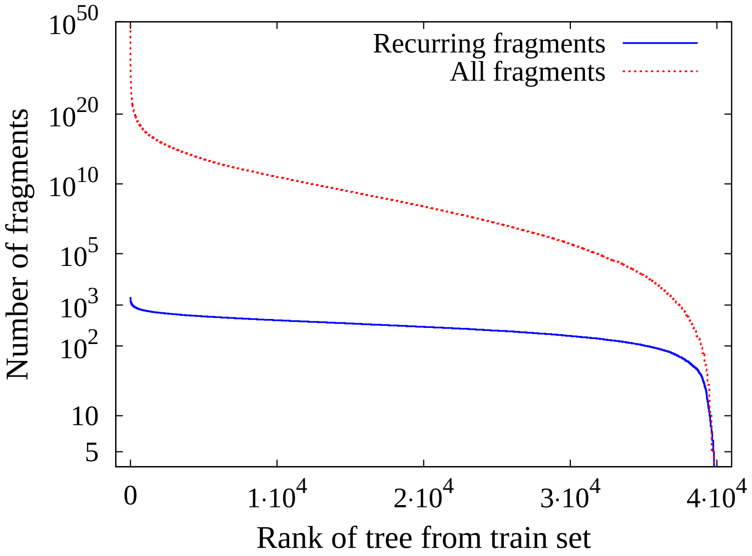


Figure 3.12: Number of fragments extracted from each tree in sections 2-21 of the WSJ treebank, when considering all-fragments (dotted line) and recurring-fragments (solid line). Trees on the x-axis are ranked according to the number of fragments. Note the double logarithmic scale on the y-axis.

3.6.2 Inducing probability distributions

Relative Frequency Estimate (RFE) The simplest way to assign probabilities to fragments is to make them proportional to their counts[12] in the training set. When enforcing equation 3.2, that gives the Relative Frequency Estimate (RFE):

$$p_{\text{RFE}}(f) = \frac{count(f)}{\sum_{f' \in F_{root(f)}} count(f')} \tag{3.4}$$

Unlike RFE for PCFGs, however, the RFE for PTSGs has no clear probabilistic interpretation. In particular, it does not yield the maximum likelihood solution, and when used as an estimate for an all-fragments grammar, it is strongly biased since it assigns the great majority of the probability mass to big fragments (Bonnema et al., 1999): grammars that implicitly encode all fragments found in a treebank are strongly biased to over-represent big fragments found in the largest constructions in the treebank.[13] DOP models relying on Goodman's transform, need therefore to counteract this tendency. Bansal and Klein (2010), for instance, rely on a sophisticated tuning technique to correctly adjust the weights of the rules in the grammar. In our Double-DOP approach, instead, this bias is much weaker as the number of fragments extracted from each tree varies much less (it ranges between 4 and 1,759) as shown in figure 3.12. Although this does not solve all theoretical issues, it makes RFE a reasonable choice.

Equal Weights Estimate (EWE) Various other ways of choosing the weights of a DOP grammar have been worked out. The best empirical results have been reported by Bod (2003) with the EWE proposed by Goodman (2003). Goodman defined it for grammars in the Goodman transform, but for explicit grammars it becomes:

$$w_{\text{EWE}}(f) = \sum_{T \in \mathscr{T}} \frac{count(f,T)}{|\{f' \in T\}|} \tag{3.5}$$

$$p_{\text{EWE}}(f) = \frac{w_{\text{EWE}}(f)}{\sum_{f' \in F_{root(f)}} w_{\text{EWE}}(f')} \tag{3.6}$$

where the first sum is over all parse trees T in the treebank ($\mathscr{T}$), $count(f,T)$ gives the number of times fragment f occurs in T, and $|\{f' \in T\}|$ is the total number of subtrees of T that were included in the symbolic grammar.

Maximum Likelihood Estimate (MLE) As an alternative estimate, we also try to find the probability distribution which maximizes the likelihood of the

[12] We refer to the counts of each fragment as returned by our extraction procedure described in §3.5.1.

[13] In fact, the number of extracted fragments increase exponentially with the size of the tree, and the great majority of the entire set of fragments belongs to the largest tree in the treebank

training treebank. For this we apply the Inside-Outside algorithm (Lari and Young, 1990), an instance of the Expectation-Maximization algorithm (EM, see also Prescher, 2003). The original version of IO is defined over *string rewriting* PCFGs, and maximizes the likelihood of the training set consisting of plain sentences. Reestimation shifts probability mass between alternative parse trees for a sentence. In contrast, our grammars consist of fragments of various sizes, and our training set of parse trees. Reestimation here shifts probability mass between alternative derivations for a parse tree (see also section 2.3.3).

In our EM experiments, we utilize a special-purpose implementation of IO for TSG as illustrate in algorithm 1 (p. 72). We start from an initial probability distribution[14] p_0 over the fragments in F. The expectation step is done in step A and B, which compute the inside and outside probabilities respectively. More specifically, for each parse tree T in the treebank, there are 3 different quantities that we keep track of in the algorithm, which are also illustrated in figure 3.13:

- $InsideNode(n)$: the inside probability of node n in T. It is computed as the sum of all the sub-derivations generating the subtree of T starting from n and ending in the lexical nodes under n.

- $InsideFrag(f, n)$: the inside probability of all sub-derivations starting with fragment f_n and generating the same subtree starting from n and ending in the lexical nodes under n (as above).

- $OutsideNode(n)$: the outside probability of node n in T. It is computed as the sum of all the derivations generating the subtree of T starting on the root node, and including in its yield n as the only substitution site, and all lexical nodes outside n.

In step C of algorithm 1 the maximization step is performed, reestimating the probabilities of the fragments. The algorithm is guaranteed to produce a sequence of reestimated probabilities $p_1, p_2, \ldots, p_n$ for which the likelihood of the training corpus monotonically increases (see also equation 2.22).

3.6.3 Maximizing Objectives

MPD The easiest objective in parsing, is to select the most probable derivation (MPD), obtained by maximizing equation 3.3. This can be done efficiently by computing the best Viterbi derivation.

[14] In our EM experiments we used the RFE from section 3.6 to obtain the initial probability distribution for the IO algorithm.

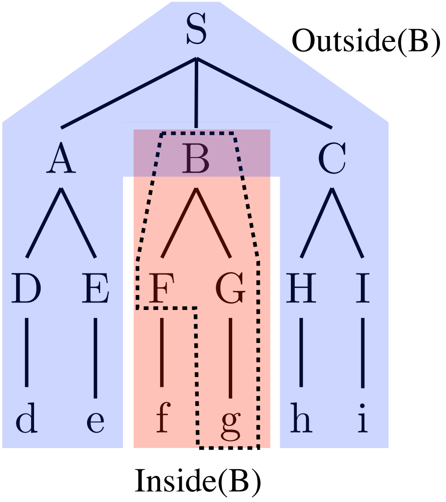


Figure 3.13: Illustration of the inside and outside probabilities of node B in a parse tree, as defined in algorithm 1. The contribution of the dashed fragment f to the inside probability is defined as $InsideFrag(f)$.

MPP A DOP grammar can often generate the same parse tree T through different derivations $D(T) = d_1, d_2, \dots d_m$. The probability of T is therefore obtained by summing the probabilities of all its possible derivations.

$$P(T) = \sum_{d \in D(T)} p(d) = \sum_{d \in D(T)} \prod_{f \in d} p(f) \tag{3.7}$$

An intuitive objective for a parser is to select, for a given sentence, the parse tree with highest probability according to equation 3.7, i.e., the most probable parse (MPP): unfortunately, identifying the MPP is computationally intractable (Sima'an, 1996). However, we can approximate the MPP by deriving a list of k-best derivations, summing up the probabilities of those resulting in the same parse tree, and select the tree with maximum probability.

MCP, MRS Following Goodman (1998), Sima'an (1999, 2003), and others, we also consider other objectives, in particular, the max constituent parse (MCP), and the max rule sum (MRS), which are all instances of the minimum Bayes risk decoding (see Smith 2011, p. 163).

MCP maximizes a weighted average of the expected labeled recall L/N_C and (approximated) labeled precision L/N_G under the given posterior distribution, where L is the number of correctly labeled constituents, N_C the number of constituents in the correct tree, and N_G the number of constituents in the guessed

```
Algorithm: $IO_{TSG}(\mathscr{T}, F, p_0, \varepsilon)$
Input: $\mathscr{T}$:treebank; $F$: fragments; $p_0$: initial prob. distr. over $F$; $\varepsilon$: stop threshold
Output: $\vec{p} = \{p_1, p_2, \ldots, p_n\}$ : reestimated probabilities over $F$
begin
    $PreviousLikelihood \longleftarrow 0$;
    $CurrentLikelihood \longleftarrow 1$;
    $iter \longleftarrow 0$;
    while $true$ do
        for $fragment\ f \in F$ do $count(f) \longleftarrow 0$;
        for $parsetree\ T \in \mathscr{T}$ do
            for $node\ n \in t$ do
                $\{OutsideNode(n), InsideNode(n)\} \longleftarrow 0$;
                for $fragment\ f \in F,\ f \in T,\ f\ rooted\ in\ n$ do
                    $InsideFragment(f, n) \longleftarrow 0$;
            // A) Compute Inside Probabilities
            for $non\text{-}lexical\ node\ n \in T\ (bottom\text{-}up)$ do
                for $fragment\ f \in F,\ f \in T,\ f\ rooted\ in\ n$ do
                    $InsideFrag(f, n) \longleftarrow p_{iter}(f)$;
                    for $node\ s \in subSites(f, T)$ do
                        $InsideFrag(f, n) *= InsideNode(s)$;
                    $InsideNode(n) += InsideFrag(f, n)$;
            // B) Compute Outside Probabilities
            $OutsideNode(root(T)) \longleftarrow 1$;
            for $non\text{-}lexical\ node\ n \in T\ (top\text{-}down)$ do
                for $fragment\ f \in F,\ f \in T,\ f\ rooted\ in\ n$ do
                    for $node\ s \in subSites(f, T)$ do
                        $OutsideNode(s) += \frac{OutsideNode(n) * InsideFrag(f, n)}{InsideNode(s)}$;
            // C) Reestimate Fragments Probabilities
            $ProbTree \longleftarrow InsideNode(root(T))$;
            for $non\text{-}lexical\ node\ n \in T$ do
                for $fragment\ f \in F,\ f \in T,\ f\ rooted\ in\ n$ do
                    $count(f) += \frac{OutsideNode(n) * InsideFrag(f, n)}{ProbTree}$;
            $CurrentLikelihood *= ProbTree$;
        for $fragment\ f \in F$ do $p_{iter+1}(f) \longleftarrow \frac{count(f)}{\sum_{f' \in F,\ root(f') = root(f)} count(f')}$;
        if $CurrentLikelihood - PreviousLikelihood < \varepsilon$ then return $\vec{p}$ ;
        $PreviousLikelihood \longleftarrow CurrentLikelihood$;
        $iter \longleftarrow iter + 1$;
```

Algorithm 1: Pseudocode for the Inside-Outside algorithm reestimating the probability distributions of the TSG fragments, in order to maximize the probability of the training treebank.

tree.[15] Recall is easy to maximize since the estimated N_C is constant. L/N_C can be in fact maximized in:

$$\hat{T} = \arg\max_{T} \sum_{lc \in T} P(lc) \tag{3.8}$$

where lc ranges over all labeled constituents in T and $P(lc)$ is the marginalized probability of all the derivation trees in the grammar yielding the sentence under consideration which contains lc.

Precision, instead, is harder because the denominator N_G depends on the chosen guessed tree. Goodman (1998) proposes to look at another metric which is strongly correlated with precision, which is the mistake rate $(N_G - L)/N_C$ that we want to minimize. We combine recall with mistake rate through linear interpolation:

$$\begin{aligned} \hat{T} &= \arg\max_{T} \mathcal{E}(\frac{L}{N_C} - \lambda \frac{N_G - L}{N_C}) && (3.9) \\ &= \arg\max_{T} \sum_{lc \in T} P(lc) - \lambda(1 - P(lc)) && (3.10) \end{aligned}$$

where 3.10 is obtained from 3.9 assuming N_C constant, and the optimal level for λ has to be evaluated empirically.

Unlike MPP, the MCP can be calculated efficiently using dynamic programming techniques over the parse forest. However, in line with the aims of this chapter to produce an easily reproducible implementation of DOP, we developed an accurate approximation of the MCP using a list of k-best derivations, such as those that can be obtained with an off-the-shelf PCFG parser.

We do so by building a standard CYK chart, where every cell corresponds to a specific span in the test sentence. We store in each cell the approximated probability of seeing every label in the grammar yielding the corresponding span, by marginalizing the probabilities of all the parse trees in the obtained k-best derivations that contains that label covering the same span. We then compute the Viterbi-best parse maximizing equation 3.10.

We implement max rule sum (MRS) in a similar way, but do not only keep track of labels in every cell, but of each CFG rule that span the specific yield (see also Sima'an, 1999, 2003). We have not implemented the max rule product (MRP) where marginal posterior probabilities are multiplied instead of added (Petrov and Klein, 2007; Bansal and Klein, 2010).

3.7 Implementation

In order to build and test our Double-DOP model, we employ the Penn WSJ Treebank (Marcus et al., 1993), and other treebanks of various languages (see

[15] For a definition of recall and precision see also §3.9.

for more details §3.8). The software produced for running our model is publicly available at `http://staff.science.uva.nl/~fsangati`.

Despite the fact that very many different DOP-parsers have been described in the literature, only one such parser has been made available for use by other researchers: the `dopdis` parser[16] (Sima'an, 1995). Zuidema (2007) describes a transform-backtransform approach that allows the use of a standard PCFG parser, but for efficiency issues he needs to restrict his experiments on Penn WSJ to sentences up to length 20. The only state-of-the-art published DOP results on Penn WSJ have been obtained with special purpose parsers that use the Goodman transform and have not yet been publicly released (Bod, 2001b; Bansal and Klein, 2010). Also for iterative re-estimation techniques such as EM, no standard software has been used. Magerman (1993, unpublished) is cited by Bod (2001a) and others as working out an EM algorithm for DOP, but the original report is not publicly available.

Treebank binarization

We start with some preprocessing of the treebank, following standard practice in WSJ parsing. We remove traces and functional tags. We apply a left binarization of the training treebank as in Matsuzaki et al. (2005) and Klein and Manning (2003), setting the horizontal history H=1 and the parent labeling P=1.[17] This means that when a node has more than 2 children, the generation of the i^{th} child (for $i \geq 3$) is conditioned on child $i - 1$. Moreover the labels of all non-lexical nodes are enriched with the labels of their parent node. Figure 3.14 shows the binarized version of the tree structure in figure 3.1. We have tried several different binarization variations, including one based on head enrichment as done in Klein and Manning (2003), and found that this is the one which works best on the WSJ development set (section 22).

Unknown words

We replace words appearing less than 5 times by one of 50 unknown word categories based on the presence of lexical features as implemented by Petrov (2009). In some of the experiments we also perform a smoothing over the lexical elements assigning low counts ($\epsilon = 0.01$) to open-class $\langle$ word, PoS-tag $\rangle$ pairs not encountered in the training corpus. A PoS-tag is an open class if it rewrites to at least 50 different words in the training corpus. A word is an open class word if it has been seen only with open-class PoS-tags. For more details on the used parameters for the various languages see also Appendix A.

[16] The parser is available at `http://staff.science.uva.nl/~simaan/dopdis/`.

[17] For a more thorough discussion on horizontal Markovization (a more general technique than binarization) see Sima'an (2000).

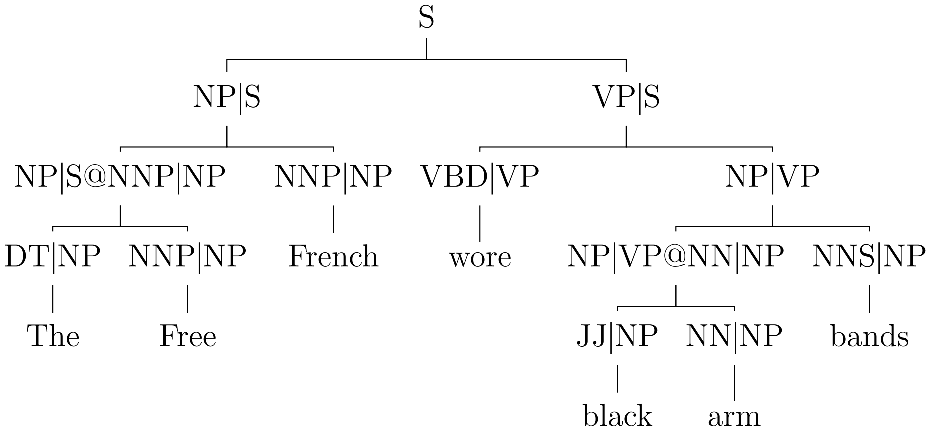


Figure 3.14: The binarized version of the tree in figure 3.4, with H=1 and P=1.

Fragment extraction and Parsing

We extract the symbolic grammar and fragment frequencies from this preprocessed treebank as explained in section 3.4. This is the most time-consuming step (around 160 CPU hours for the binarized treebank).

Parse trees in the training corpus are not necessarily covered entirely by recurring fragments; to ensure better coverage, we also extract all PCFG-productions not included in the set of recurring fragments.

In the extracted grammar from the Penn WSJ treebank we have in total 1,029,342 recurring fragments and 17,768 unseen CFG rules. We test several probability distributions over the fragments (§3.6.2) and various maximization objectives (§3.6.3).

We convert our PTSG into a PCFG (section 3.6.1) and use `Bitpar`[18] for parsing. The grammar extracted from the Penn WSJ treebank after smoothing consists of 1,476,941 PCFG rules.

For approximating MPP and other objectives we marginalize probabilities from the 1,000 best derivations.

3.8 Annotated Resources

In the current thesis we will adopt the English Wall Street Journal (WSJ) section of the Penn 3 treebank (Marcus et al., 1999), as the main resource for training and testing our models. The WSJ treebank, was developed in 10 years (1989-1999)

[18] `http://www.ims.uni-stuttgart.de/tcl/SOFTWARE/BitPar.html`

through a series of incremental annotation improvements and refinements (Marcus et al., 1993; Taylor et al., 2003; Marcus et al., 1994). The resource has quickly become the leading corpus for the evaluation of PS syntactic parsers. Recently, other research groups spent great efforts to develop similar resources for other languages (Bosco et al., 2000; Sima'an et al., 2001; Brants et al., 2002; Abeillé et al., 2003; Oflazer et al., 2003). In §3.10 we will test our Double-DOP model on these treebanks.[19] In table 3.1 we present the references to the material together with the details about the training, development, and test section for replicability purpose.

3.9 Evaluation Metrics

As in previous work on PS parsing we rely on parseval scores (Back et al., 1991) as the standard metric for evaluating our models. In particular we report Labeled Recall, Labeled Precision, F1 score (the harmonic mean between the two), and Exact Match score (EX). These metrics are reported in equations 3.11-3.14. We compute this metrics using `EvalB`[20] (Sekine and Collins, 1997) using parameter file *new.prm* (see also more details in Appendix A.2). We have also used `EvalC`[21] to obtain more detailed results, in particular, per category accuracy.

$$\text{Labeled Recall} = \frac{\#\text{ correct labeled brackets}}{\#\text{ of brackets in gold}} \tag{3.11}$$

$$\text{Labeled Precision} = \frac{\#\text{ correct labeled brackets}}{\#\text{ of brackets in guess}} \tag{3.12}$$

$$\text{F1} = \frac{2 \cdot \text{Labeled Recall} \cdot \text{Labeled Precision}}{\text{Labeled Recall} + \text{Labeled}} \tag{3.13}$$

$$\text{EX} = \frac{\#\text{ correct parse trees}}{\#\text{ total parse trees}} \tag{3.14}$$

3.10 Results

Maximizing Objectives We start by presenting the results we obtain with the extracted DOP grammar on the development set of the WSJ. Initially we compare the maximizing objectives presented in section 3.6.3, over the different

[19] Many thanks to Djamé Seddah and Benoit Crabbé for providing us useful advice when testing on the French treebank, and to Yoav Goldberg and Reut Tsarfaty for the Hebrew treebank.

[20] `http://nlp.cs.nyu.edu/evalb/`

[21] `http://staff.science.uva.nl/~fsangati/`

Language	Treebank	Reference	Training Set	Development Set	Test Set
English	WSJ	Marcus et al. 1999	Sections 02-21 [39,832 in total]	Section 24 [1,346 in total]	Section 23 [2,416 in total]
English	Brown	Marcus et al. 1999	All besides test [21,818 in total]	None	Tree indices 1, 11, 21, 31, ..., 241 [2,425 in total]
German	Negra	Skut et al. 1997	Tree indices 1-18,602 [18,602 in total]	Tree indices 18,603-19,602 [1,000 in total]	Tree indices 19,603-20,602 [1,000 in total]
French	FTB	Abeillé et al. 2003	File ftb_1.mrg [9,881 in total]	File ftb_2.mrg [1,235 in total]	File ftb_3.mrg [1,235 in total]
Chinese	CTB 3.0	Xue et al. 2002	Articles 1-270, 400-1151 [18,104 in total]	Articles 301-325 [352 in total]	Articles 271-300 [348 in total]
Hebrew	HTB	Sima'an et al. 2001	Tree indices 501-6000 (plaintb_minf.train) [5,241 in total]	Tree indices 1-500 (plaintb_minf.dev) [483 in total]	Tree indices 6001-6501 plaintb_minf.test [496 in total]

Table 3.1: Details of the treebanks used in the parsing results (tree indices 1 stands for the first tree in the treebank).

probability estimates over the fragments (RFE, EWE, MLE). We conclude that, empirically, MCP for the optimale choice of λ, is the best objective to maximize F1, followed by MRS, MPP, and MPD. In figure 3.15 we show the comparison of the various objectives for RFE where MCP reaches maximum performance for $\lambda = 1.15$.

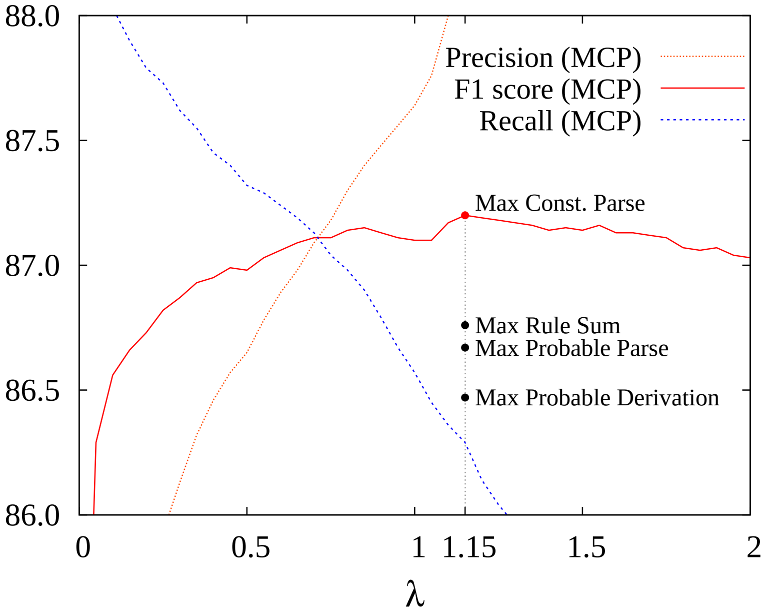


Figure 3.15: DOP results on the development section (≤ 40) with different maximizing objectives.

Probability Estimates We also compare the various estimates presented in §3.6.2, on the same development set. We find that RFE is the best estimate (87.2 with MCP for $\lambda = 1.15$) followed by EWE (86.8 with MCP for $\lambda = 0.55$), and MLE (86.7 with MCP for $\lambda = 0.95$). Our best results with MLE are obtained when removing fragments occurring less than 5 times (apart from CFG-rules) and when stopping at the second iteration. This filtering is done in order to limit the number of big fragments in the grammar. It is well known that MLE for DOP tends to assign most of the probability mass to big fragments, quickly overfitting the training data. We find surprising that EWE performs worse than RFE, contrary to previous work (Bod, 2003).

Fragments Selection We also investigate how a further restriction on the set of extracted fragments influences the performance of our model. In figure 3.16 we illustrate the performance of Double-DOP when restricting the grammar to fragments having frequencies greater than $1, 2, \ldots, 100$. We can notice a rather sharp decrease in performance as the grammar becomes more and more compact.

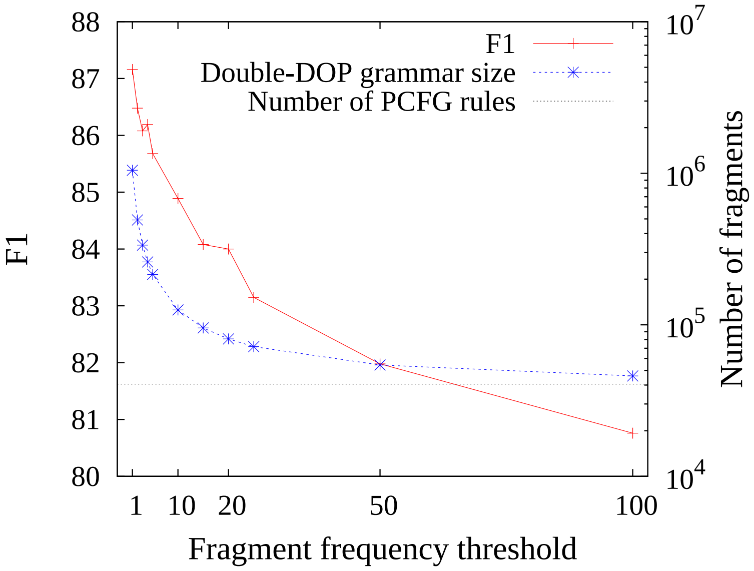


Figure 3.16: Performance (on the development set) and size of Double-DOP when considering only fragments whose occurring frequency in the training treebank is above a specific threshold (x-axis). In all cases, all PCFG-rules are included in the grammars. For instance, at the right-hand side of the plot a grammar is evaluated which included only 6,754 fragments with a frequency greater than 100 as well as $39,227$ PCFG rules.

Grammar Refinement Next, we present some results on various Double-DOP grammars extracted from the same training treebank after refining it using the Berkeley state-splitting model[22] (Petrov et al., 2006; Petrov and Klein, 2007). In total we have 6 increasingly refined versions of the treebank, corresponding to the 6 cycles of the Berkeley model. We observe in figure 3.17 that our grammar is able to benefit from the state splits for the first four levels of refinement, reaching the maximum score at cycle 4, where we improve over our base model. For the last two data points, the treebank gets too refined, and using Double-DOP model on top of it, no longer improves accuracy.

We have also compared our best Double-DOP base model and the Berkeley parser on per-category performance. Here we observe an interesting trend: the Berkeley parser outperforms Double-DOP on very frequent categories, while

[22] We use the Berkeley grammar labeler following the base settings for the WSJ: trees are right-binarized, H=0, and P=0. Berkeley parser package is available at `http://code.google.com/p/berkeleyparser/`

Double-DOP performs better on infrequent ones. A detailed comparison is included in table A.2 (Appendix A).

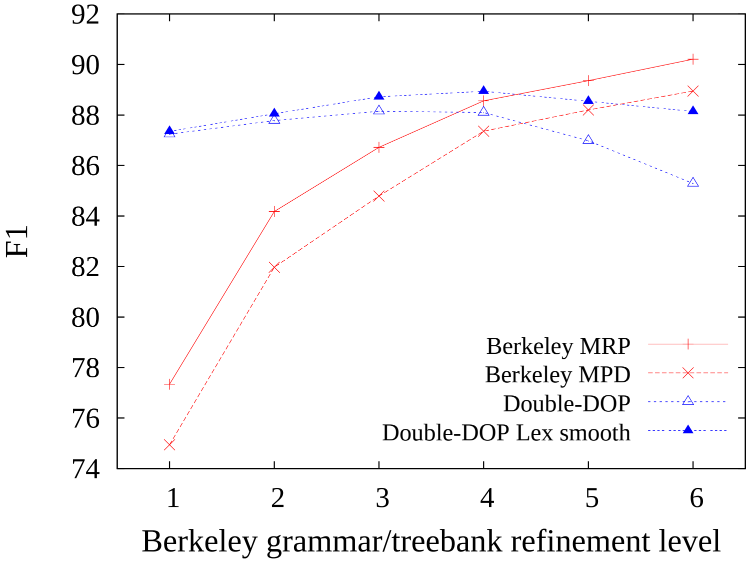


Figure 3.17: Comparison on section 24 between the performance of Double-DOP (using RFE and MCP with $\lambda = 1.15$, H=0, P=0) and Berkeley parser on different stages of refinement of the treebank/grammar.

Final WSJ Results In table 3.2 we present our results on the test set (section 23) of the WSJ. Our best model (according to the best settings on the development set) performs slightly worse than the one by Bansal and Klein (2010) when trained on the original corpus, but outperforms it (and the version of their model with additional refinements) when trained on the refined version, in particular for the exact match score.

Multilingual Results Apart from the WSJ treebank we have tested our Double-DOP model on 5 other corpora (see §3.8). Results are presented in table 3.3. For more details on the parameters used for these experiments see table A.1 (Appendix A). Results for the PCFG, Berkeley and Double-DOP were all trained and evaluated locally and therefore mutually comparable. Other reported results might have been run on other versions of the treebanks. Double-DOP consistently outperforms the baseline PCFG model and earlier parsing results, but it is consistently outperformed by the Berkeley parser.

	test (≤ 40)		test (all)	
Parsing Model	F1	EX	F1	EX
PCFG Baseline				
PCFG (H=1, P=1)	77.6	17.2	76.5	15.9
PCFG (H=1, P=1) Lex smooth.	78.5	17.2	77.4	16.0
FRAGMENT-BASED PARSERS				
Zuidema (2007)*	83.8	26.9	-	-
Cohn et al. (2010) MRS	85.4	27.2	84.7	25.8
Post and Gildea (2009)	82.6	-	-	-
Bansal and Klein (2010) MCP	88.5	33.0	87.6	30.8
Bansal and Klein (2010) MCP + Additional Refinement	88.7	33.8	88.1	31.7
THIS WORK				
Double-DOP	87.7	33.1	86.8	31.0
Double-DOP Lex smooth.	**87.9**	**33.7**	**87.0**	**31.5**
Double-DOP-Sp	88.8	35.9	88.2	33.8
Double-DOP-Sp Lex smooth.	**89.7**	**38.3**	**89.1**	**36.1**
REFINEMENT-BASED PARSERS				
Collins (1999)	88.6	-	88.2	-
Petrov and Klein (2007)	**90.6**	**39.1**	**90.1**	**37.1**

Table 3.2: Summary of the results of different parsers on the test set (sec 23). Double-DOP experiments use RFE, MCP with $\lambda = 1.15$, H=1, P=1; those on state-splitting (Double-DOP-Sp) use Berkeley cycle 4, H=0, P=0. Results from Petrov and Klein (2007) already include smoothing which is performed similarly to our smoothing technique (see §3.7). (* Results on a development set, with sentences up to length 20.)

		test (≤ 40)		test (all)	
Treebank	Parsing Model	F1	EX	F1	EX
Brown	PCFG	80.6	27.0	77.8	24.5
	Berkeley Cycle 5	88.5	43.6	87.3	41.2
	Double-DOP	86.4	38.4	84.6	36.2
Negra	PCFG	71.6	33.3	71.3	32.7
	Dubey (2005)	76.3	-	-	-
	Berkeley Cycle 4	78.9	42.3	78.5	41.6
	Double-DOP	76.5	40.2	76.0	39.5
FTB	PCFG	76.0	16.3	72.7	12.8
	Arun and Keller (2005)	79.1	21.2	75.6	16.4
	Berkeley Cycle 5	84.2	26.1	82.2	20.8
	Double-DOP	81.5	21.1	78.7	16.6
CTB 3.0	PCFG	65.2	23.4	62.6	20.1
	Bikel (2004b)	81.2	-	79.0	-
	Berkeley Cycle 5	86.0	40.5	83.0	34.8
	Double-DOP	82.1	20.7	81.1	17.8
HTB	PCFG	73.2	9.0	71.3	7.9
	Berkeley Cycle 5	81.4	15.7	80.0	13.9
	Double-DOP	77.9	13.8	75.8	12.1

Table 3.3: Parsing results on different treebanks: Brown (English), Negra (German), FTB (French), CTB (Chinese), and HTB (Hebrew).

3.11 Conclusions

We have described Double-DOP, a novel DOP approach for parsing, which uses all constructions recurring at least twice in a treebank. This methodology is driven by the linguistic intuition that constructions included in the grammar should prove to be reusable in a representative corpus of annotated productions. The extracted set of fragments is significantly smaller than in previous approaches. Moreover constructions are explicitly represented, which makes them potentially good candidates as semantic or translation units to be used in other applications.

Despite earlier reported excellent results with DOP parsers, they are almost never used in other NLP tasks: where other successful parsers often feature as components of machine translation, semantic role labeling, question-answering or speech recognition systems, DOP is conspicuously absent in these neighboring fields (but for possible applications of closely related formalisms see, e.g., Bonnema et al., 1997; Hearne and Way, 2006; Yamangil and Shieber, 2010). The reasons for this are many, but most important are probably the computational inefficiency of many instances of the approach, the lack of downloadable software and the difficulties with replicating some of the key results.

In this chapter we have addressed all three obstacles: our efficient algorithm for identifying the recurrent fragments in a treebank runs in polynomial time, and the transformation to PCFGs that we define allows us to use a standard PCFG parser, while retaining the benefit of explicitly representing larger fragments. Finally, the availability of our programs, as well as the third party software that we use, also addresses the replicability issue. Where some researchers in the field have been skeptical of the DOP approach to parsing, we believe that our independent development of a DOP parser adds credibility to the idea that an approach that uses very many large subtrees, can lead to very accurate parsers.

3.11.1 Future Directions

There is a number of extensions of the current model that are left unexplored. The set of recurring fragments used for parsing is rather big. Nevertheless there is still a certain amount of fragments that occur only once in the training set which reoccur for the second time only in the development set. Some of these are still relatively small fragments. It is then possible to augment our model with all or a random sample of small fragments that occurred only once in the training corpus; this strategy might improve performance, while harming efficiency.

A more methodological variation of the model could come from the preprocessing of the treebank. One of the steps that were essential for the success of the model was the binarization procedure. Binarization is particularly important for generative models like DOP and PCFGs, where all the daughters of an internal node are produced at once. Binarization, in fact, provides a way to generalize flat rules, by splitting it in multiple generation steps. In initial unpublished experiments, our DOP model trained on an unbinarized treebank performed rather poorly because of the abundance of flat rules. However, our current model uses a strict left binarization (see §3.7). Although the practice of resorting to binarization techniques is often considered a mere heuristic for achieving higher parsing results, we would like to stress the fact that it is fundamental for solving the under-generation limitation of the underlying PCFG model; we believe more effort should be put into validating ways to perform this step.

Moreover, there is a number of probabilistic estimates that are left unexplored in our study. In particular, while our version of EM (see §3.6.3) tends to over-fit the data as it tries to maximizes the likelihood of the training treebank, a more sensible approach would be to follow a *cross-validation* (CV) instantiation of EM (Zollmann and Sima'an, 2005; Mylonakis and Sima'an, 2008). According to EM-CV the training treebank is partitioned in several parts and EM is run on each of the partitions separately, although the frequency estimates of the fragments are averaged over the EM-rounds to prevent over-fitting on the various partitions.

Finally, we believe that our algorithm for extracting recurring fragments could be beneficial in solving a number of linguistic tasks automatically, such as the distinction between argument and adjuncts in a PS treebank (Villavicencio, 2002; Abend and Rappoport, 2010). In particular, a variation of our algorithm (Sangati et al., 2010) is able to detect recurring partial-fragments, which are less restrictive than the type of fragments used in this chapter: a partial-fragment can in fact include any connected subset of nodes of the original tree, which allows to discard in a production any number of daughters. The use of partial-fragments could give also rise to yet another interesting parsing framework, where the argument daughters of a node (which are usually no more than 3) are produced in a single step, while adjuncts are inserted one at a time with a special-purpose insertion operation (Schabes and Waters, 1995; Hwa, 1998; Bangalore et al., 2009) which is constrained to apply on more restricted contexts (see also the "Sandwich Insertion Grammar" in example 2.2.6).

3.11.2 Next steps

After having described a novel DOP model for learning phrase-structure trees, in the next chapter we will turn our attention to dependency-based syntax. As described in §1.2.5 PS and DS are complementary representations focusing on two fundamental aspect of syntax, i.e., grouping and relations respectively. The fragment-grammar we use in the Double-DOP model is able to capture a large part of the relevant syntactic constructions, such as the argument structure of many lexical items, as well as a number of idiomatic expressions, but it is strongly based on the notion of substitutability of internal phrasal categories. In the dependency-structure models, instead, we will focus on the direct relations existing between words in a sentence, which Double-DOP is not able to capture consistently.

Chapter 4
Learning Dependency-Structures

Like the body that is made up of different limbs and organs, all words must depend on each other to exist.

Adaptation of an Hindu proverb

4.1 Introduction

The aim of this chapter is to introduce syntactic dependency-structures (DS), and compare a number of computational models for obtaining the correct analysis of natural language sentences according to this representation.

We have decided to investigate dependency-structure, since it is a complementary scheme with respect to phrase-structure representation employed in the previous chapter. While PS trees are centered on abstract syntactic categories, and assign a rather marginal role to lexical productions, DS trees focus on words and on the direct dependency relations conntecting them.

Historically, the modern theoretical tradition of dependency-structures was started by Lucien Tesnière (1959), who formulated this theory long before the formal notion of phrase-structure was proposed by Chomsky (1957).[1] Nevertheless, most of the initial formal studies on dependency-structure (Hays, 1960; Gaifman, 1965) were conducted only after phrase-structure was introduced, and were presented mainly in relation to it.

We will dedicate chapter 5 to the description of a syntactic representation which is closer to the one described by Tesnière. There, we will show how Tesnière's theory incorporates aspects from both constituency and dependency theories, and makes use of special constructions for handling common linguistic phenomena such as coordination, often neglected in other representations.

In the current chapter, we will limit our analysis to the description of simple dependency-structure, where *dependency* is the only relation to link words in a sentence. This notion of DS is the one which was commonly adopted in the computational linguistics community in the '90s, also thanks to the emergence of hybrid models such as head-driven formalisms (Magerman, 1994; Collins, 1997, see also §3.2.1), which have strengthened the role that the lexicon plays in syntactic theories. Since then, DS received an increasing interest as an alternative representation to the traditional PS.

After introducing the general notion of dependency-structure, we will describe the relation that exists between PS and DS, and how it is possible to transform one representation into the other. Finally we will present a family of generative models for learning DS, and show some results when parsing the Wall Street Journal treebank using a reranking approach. The main motivation for using a reranking approach is that it allows for efficiently evaluating many generative models, differing from one another on the hypothesis for how to derive the most probable structure for a given sentence. Although efficient algorithms exist to calculate parse forests (e.g., Eisner and Satta, 1999), each choice gives rise to a different parser instantiation, while in our approach we can mimic the steps of each specific parser and employ a unique evaluation procedure (see also §2.4).

[1] In fact *Eléments de syntaxe structurale* (Tesnière, 1959) was published posthumously (Tesnière died in 1954).

4.2 Dependency-Structure

A dependency-structure of a sentence is a tree structure whose nodes are the words in the sentence. Differently from PS trees, where the order relation is only defined for a subset of nodes in the tree (e.g., the children of each node), all the nodes of a DS tree are in *precedence* relation to one-another (see the difference between figure 2.2 and 2.3). Precedence is implicitly defined by the order of the words in the sentence.

Figure 4.1 shows an example[2] of a DS. Edges indicate a dependency relation between the upper word, the *governor* (or *head* or *parent*) and the lower word, the *dependent* (or *child*). In figure 4.2 we report the same structure using an equivalent flat representation in which head-dependent relations are marked with directed edges. The tree structure must be connected, and each must have exactly one governor, except for the root node which has none.

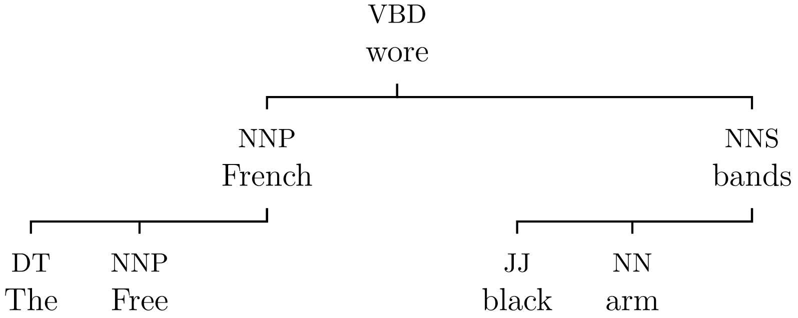


Figure 4.1: Example of a dependency-structure according to the DS representation used in this thesis.

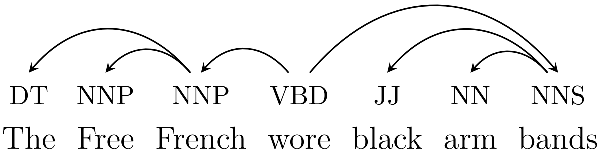


Figure 4.2: Example of the dependency-structure in figure 4.1 using an equivalent flat representation. For every arc the arrow points at the dependent of the governor-dependent relation.

[2] In our DS trees we always report part-of-speech tags together with words, since both are used for parsing, and they should therefore be considered a unique entity (tagged word). In PS, instead, pos-tags rewrite to words at the bottom of each tree. See for comparison the PS of the same sentence in figure 3.1.

The main conceptual difference between PS and DS is that a PS describes part-whole relations between constituents of a sentence, while a DS specifies the interconnections between words, viz. the governing-dependency relations (see also the discussion in §1.2.5).

In this chapter we will be focusing on *projective* DS, since our models will be tested on the English WSJ treebank converted to projective DS.[3] A DS is projective iff for every word, the set including the word and all its direct and indirect dependents forms a continuous span of the sentence (without any hole).

Moreover, we will only consider *unlabeled* DS. A labeled DS, presents specific tags on its arcs (e.g., subject, object, determiner, etc...) denominating the functional role that the dependent has with respect to the governor. These labels are useful as a further differentiation of the dependents of a certain word, and they can be used, for instance, for detecting the argument structure of verbs (who did what to whom) in parsing-related tasks (e.g., semantic role labeling). In our formalization we ignore these extra labels because we are mainly focusing on learning the bare structure of the sentence and for all the models we will deal with, functional labels do not add any contribution for solving this task.

4.3 Comparing PS with DS

In this section we present a review some of the past studies that have attempted to formally compare PS with DS. While in §4.3.1 we focus on the *structural relations* between the two representations, in §4.3.2 we describe previous attempts of defining comparisons over *grammar formalisms* for the two schemes.

4.3.1 Structural relations between PS and DS

Hays (1960) presents a first comparison between PS and DS. In particular, he formalizes a correspondence between the two types of structure: each PS maps to multiple DS, and each DS to one or more PS. Figure 4.3 shows these mappings for a generic sentence of 3 words. Here we have adopted a simplified notation: we omit the labels in the internal nodes of the PS and unary chains are avoided. In both representations part-of-speech tags are collapsed with words.

From the same figure it is immediately clear that there are more projective DS than PS for a given sentence. In both cases the number of structures grows exponentially in the length of the sentence,[4] with DS consistently outnumbering

[3] Although non-projective conversion of the same treebank exists (Johansson and Nugues, 2007) we have preferred to use more standard conversion procedures. It is commonly known that, compared to other languages, non-projective constructions in English are quite rare and mainly pertaining long-distance relations (e.g., wh-movement). For more details on non-projective DS and ways to parse them see Kuhlmann (2007).

[4] For a sentence of length $n = \{1, 2, 3, 4, 5, 6\}$ we have $\{1, 1, 3, 11, 45, 197\}$ PS trees and $\{1, 2, 7, 30, 143, 728\}$ DS trees. The number of PS trees is given by the super catalan number

PS.

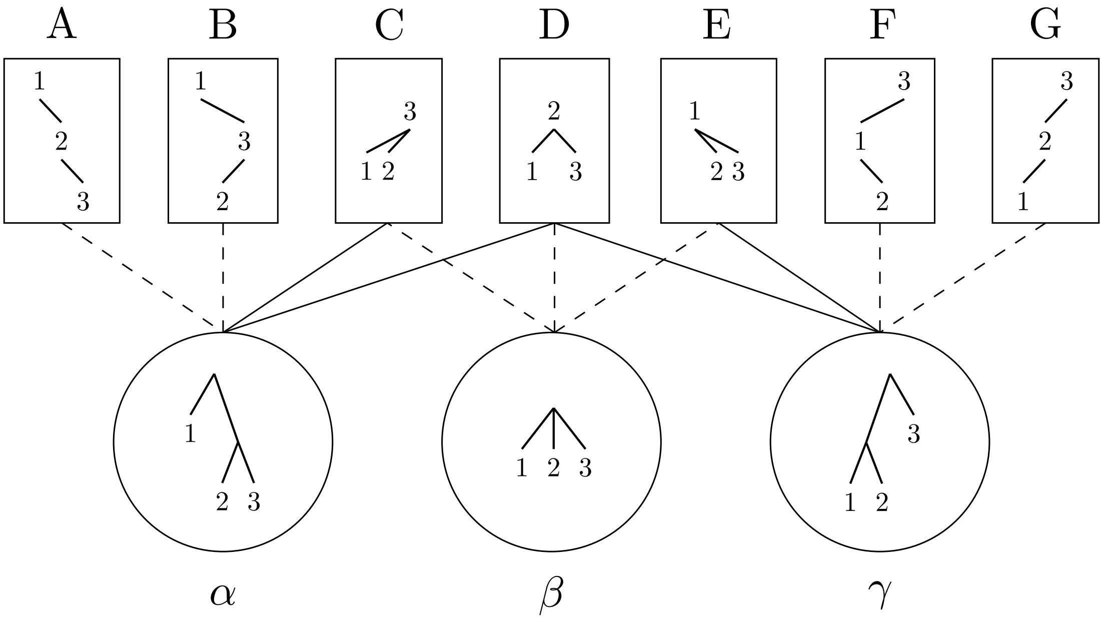


Figure 4.3: Mappings between DS (in rectangles) and PS (in circles) with 3 words. Dashed lines indicate the mappings for which the number of constituents in the PS is equal to the number of the governors in the DS.

PS to DS For English, there is no significant treebank annotated in dependency format. In later experiments we will therefore resort to a transformation from the WSJ treebank into projective dependency representation. In order to perform such transform we need to use the notion of *head*[5] as introduced in section 3.2.1: for every non-terminal node in a PS we need to select a single daughter as the head of the constituent. The following recursive procedure takes as input an head-enriched PS tree rooted in node N and outputs the corresponding DS:

1. If N is a terminal PS node, it is transformed into a single terminal DS node with no dependents.

2. If N is a non-terminal node, all its daughters are transformed into DS trees (recursive step); those obtained from the non-head daughters become dependent of the root of the DS obtained from the head daughter, following the same order as in the PS. For instance in figure 4.3 the leftmost PS (α) is mapped into the leftmost DS (A) if node 2 is chosen as the head of the constituent (2 3) and 1 as the head of the whole tree.

sequence $P(n) = 3(2n-3)P(n-1) - (n-3)P(n-2)/n$, while the number of DS trees is obtained as $D(n) = C(3n+1, n)/(n+1)$, where C is the binomial coefficient, $C(n,k) = n!/(k!(n-k)!)$.

[5] Interestingly, Hays (1960) does not refer to the notion of head in his formulation, but he indirectly refers to it as "the chosen" constituent for the transformation to apply.

According to this conversion procedure, the number of DS trees that correspond to a given PS is determined by the different ways of assigning heads to the internal nodes in the PS. This can be easily computed as the product of the number of daughters of each non-terminal node in the PS.

The reversed transformation (DS to PS) is more elaborate and less discussed in the literature. We report this procedure in Appendix B.1.

4.3.2 Relations between PS and DS grammars

Context-Free Grammars Gaifman (1965) defines a way to construct a DS tree via Context Free Grammar rules (DS-CFG). This is similar to CFGs defined for PS (see §1.3.1 and §2.2.2), except that, instead of using the rewriting symbol ($\rightarrow$), parenthesis are used to distinguish the governor from the dependents.[6] Moreover, in order to preserve the linear order of words, rules use the star symbol ($*$) to separate left and right dependents of a governing word.

In the general case, the system represents a word w with all its dependents d_1, d_2, …, d_k ($k \geq 1$) as a single rule, i.e., $w(d_1, \ldots, d_i, *, d_{i+1} \ldots, d_k)$, where word w occurs in the sentence after word d_i and before d_{i+1} ($0 \leq i \leq k$). If w has no dependent ($k = 0$), it is represented as $t(*)$. Finally, if w is the root of the sentence, the rule $*(w)$ is used. This notation has been adopted as a way to compactly represent DS trees (see also Hays, 1964). For instance trees B, D, and F in figure 4.3 can be represented with the expression $*(1(*3(2*)))$, $*(2(1*3))$, and $*(3(1(*2)*))$ respectively.

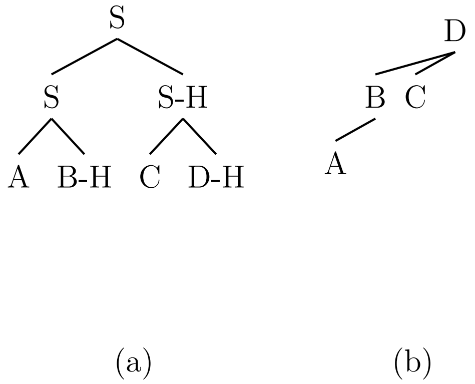
In the same paper, Gaifman (1965) investigated the correspondences between PS-CFG and DS-CFG (see also Heringer, 1967; Schneider, 1998; Nivre, 2006). The work has been later on reformulated and extended by Robinson (1967). The main result of these papers is the proof that for every PS-CFG there exists a *weakly equivalent* DS-CFG, and vice versa, i.e., the two grammars yield the same set of sentences.[7] Moreover two equivalent grammars would produce pairwise corresponding trees according to the definition originally proposed by Hays (1960).

The idea of transforming a PS-CFG into an equivalent DS-CFG, although methodologically sound, has never been adopted in real systems. One of the major problems is that the resulting DS-CFGs are usually bigger than the original one, especially for non-trivial PS-CFG presenting recursive rules.[8] An example that shows this is presented in figure 4.4: while PS-CFG can compactly represent recursive rules, DS-CFG needs to encode all possible combinations explicitly.

[6] In fact, while in PS a node is made of (rewrites to) different daughter nodes, in DS a word cannot be substituted by its dependents.

[7] An important condition for the mapping to exist is that the PS-CFG should be of *finite degree*, i.e., for every category there must be an upper bound to the smallest number of successive rule applications in order to reach a terminal symbol.

[8] We are not aware of any formal or empirical study attempting to quantify such differences.



PS-CFG	DS-CFG
	*(D)
	D(B C ∗)
	B(A ∗)
	A(*)
S → S S	C(*)
S → A B	*(B)
S → C D	B(D A ∗)
	D(C ∗)
	D(D C ∗)
	B(B A ∗)

(c)

Figure 4.4: Example of a PS (a) and a corresponding DS (b), according to the specific head assignment marked with -H in (a). The table in (c) lists in the first column the PS-CFG rules extracted from the PS (a), and in the second column the weakly equivalent DS-CFG. In this example the DS-CFG rules are about three times more numerous than the PS-CFG rules: the DS-CFG rules above the line are the ones necessary to encode the tree in (b), while the ones below the line are needed to encode other trees, yielding for instance ABAB, CDCD, ABABCD.

In general, the application of CFG-rules and PS methodologies to dependency-structures had the unfortunate effect of overlooking most of the peculiarities which are typical of dependency-structure. In particular, while PS notation defines an elegant way to compactly represent abstract syntactic rules (internal rewriting rules) that can be easily extended in a second moment to a particular choice of a lexicon (terminal rules), DS cannot easily capture the notion of abstract rules, since the whole structure is defined at the lexicon level.

Projective Bilexical Dependency Grammars In the DS-CFG described above, all dependents of each word are produced at once. In a Projective Bilexical Dependency Grammar (PBDG), instead, each dependency relation is treated individually. In this way the problem of finding a DS for a given sentence can be formulated as *finding the governor of each word*, and assuring that the final structure is connected and projective. This approach has been shown to be more suitable for learning dependency-structures, and will constitute the basic formalism when investigating generative models for DS in §4.5. PBDGs allow for a greater generalization with respect to DS-CFG. In fact, each word can be attached to any available governor,[9] irrespectively of any other dependent attached to it. This can be problematic for it allows, for instance, two objects to attach to a simple transitive verb. More sophisticated models define specific constraints to alleviate this problem: usually, the choice of attaching a specific word as a dependent of a certain node is conditioned on its previous dependents and on ancestral information. PBDGs can be easily extracted from a corpus of annotated DS trees. For instance in figure 4.5 we show the PBDG extracted from DS tree in figure 4.1.

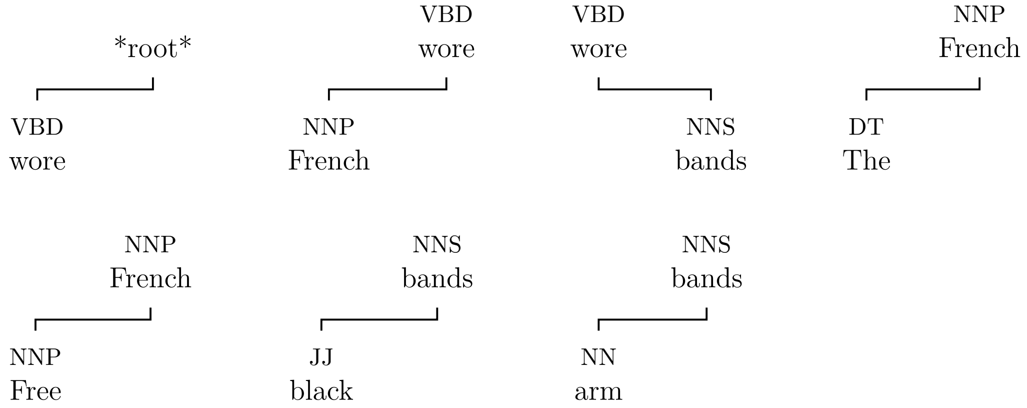


Figure 4.5: An example of Bilexical Dependency Grammar extracted from the DS tree in figure 4.1.

A study which clarifies the grammatical relations between PBDG and PS is the work of Johnson (2007). It describes a series of Unfold-Fold procedures to transform a Projective Bilexical Dependency Grammar (PBDG) into weakly equivalent Context-Free Grammars. The main motivation behind this work is to make available all known techniques developed for CFGs to PBDGs, including the possibility to use the standard CYK algorithm for parsing DSs according to a projective bilexical generative model. These transformations are shown to mimic the chart-parsing algorithm defined for PBDG in Eisner and Satta (1999).

4.4 Other related syntactic theories

In this section we briefly summarize some of the other existing syntactic theories. These, compared to standard DS, are characterized by having deeper and more

[9] A word should always look for an available governor to ensure projectivity. If, for instance, a model assigns governors of each word in the sentence in sequential order, G turns out to be an *available* governor for the current word W, if the span constituted by G and all its direct and indirect dependents is adjacent to W.

complex representations, for which there is no significant amount of annotated material; they are therefore harder to study empirically. Other formalisms (e.g., CCG) which have significant annotated resources derived either manually or automatically from existing corpora are presented in §5.6, where we will compare them with our TDS annotation scheme.

Lexical Functional Grammar (LFG)

Lexical Functional Grammar (Kaplan and Bresnan, 1982; Bresnan, 2000; Dalrymple, 2001) is a linguistic theory that in contrast to other approaches assumes two different levels of syntactic representation. It postulates a unique constituent structure (c-structure), corresponding to the superficial phrase-structure tree. In addition it defines a functional level (f-structure) which is lexicalized and can better represent the argument structure of the sentence including possible variation in the surface position of the arguments. The two representations are related to the PS and DS representations that are discussed in this thesis. However, LFG representation is usually considered more deep and therefore harder to be derived with a computational model. Previous studies that have tested LFG for parsing have restricted the model to a manually defined LFG grammar (e.g., Riezler et al., 2002) or trained on a restricted treebank (e.g., Bod and Kaplan, 2003).

Head-driven Phrase-Structure Grammar (HPSG)

Head-driven Phrase-Structure Grammar (Pollard et al., 1994) is a formal linguistic theory based uniquely on feature structures (differently for instance from LFG which uses feature structure only within the functional representation). Each feature consists of an attribute and a value, where values can themselves be subfeatures in a nested structure. Feature structures are used to represent grammar rules, principles, and lexical entries. As in LFG, HPSG is strongly lexicalized as there is a small number of principles and grammar rules, while the lexicon is rich and complex.

X-bar theory

X-bar theory (Harris, 1951; Chomsky, 1970; Jackendoff, 1977) is a theory for describing the relations between lexical and phrasal expressions of the same type (e.g., verb and verbal phrase). It can be seen as a hybrid approach between PS and DS as it captures the notion of head within phrase-structure.[10] In his formulation, Chomsky (1970) proposes the phrase-structure scheme of figure 4.6, where X ranges among the main categories (Nouns, Verbs, Adjectives, Prepositions).

[10] See also Covington (1992) for more details on the interrelation between X-bar theory and dependency-structure.

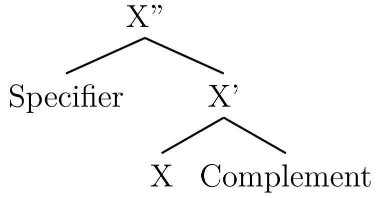


Figure 4.6: X-bar scheme

According to this scheme every main phrasal category NX is defined as a continuous projection from a word belonging to the base category X to its maximal projection (NX”) passing through a sequence of one or more single-bar projections (X’). Moreover every projected category contains a unique head being the same category with one fewer bar level (e.g., X’ is the head of X”), and non-head daughters have to be maximal projections. Finally, complements are attached as daughters of the X’ levels, while the specifier is attached to the X” level.

Word Grammar (WG)

Word Grammar (Hudson, 1991; Sugayama and Hudson, 2005; Hudson, 2010) is a theory of language structure based on dependency relations between single words. Hudson has spent great effort to affirm the validity of dependency-structures over phrase-structure. One of the strongest arguments is that while dependencies well represent the selectional preferences of a word (e.g., verb with preposition as in *depend on* and *ought to*), in PS the relation is less direct as there is always at least an abstract node (e.g, prepositional phrase) between the two words. Differently from standard DS illustrated in this chapter, in WG one word is allowed to be a dependent of more than one other word, non-projectivity is permitted and coordinated structure are treated separately (grouping the conjuncts as a list of entities), allowing to represent both deep and surface information in a single structure.

Meaning Text Theory (MTT)

Meaning Text Theory (Mel’čuk, 1979, 1988, 2003; Polguère and Mel’čuk, 2009) is a theoretical linguistic framework put forward by Alexander K. Zholkovsky and Igor Mel’čuk, and it is strongly inspired by the work of Tesnière (1959).[11] MTT distinguishes three main facets of a linguistic event: meaning (the content), text

[11] Mel’čuk was one of the few linguists who attempted to promote the work of Tesnière outside France (especially in Russia, Canada, and United States).

(the form), and a mapping between the set of meanings and the set of texts. It incorporates several representation layers including semantics, syntax, morphology, and phonology. In the syntactic layer, MTT makes use of dependency trees in two different forms, i.e., deep and surface representation. The first specifies the organization of a sentence considered from the meaning frame of reference, while the latter is more connected to the surface form and therefore the actual linear order of the words. While the inventory of the relations in the deep syntactic representation is supposed to be language-independent and related to the argument structure of the sentence (subject, object, complements), in the surface syntactic representation the relations are language specific, and the inventory is established empirically. MTT deal with coordination structures by putting each element in the coordination (conjunction or conjunct) as the governor of the following element (see also §5.3.3).

4.5 Models for parsing DS

In the following sections we will review some of the most relevant works on dependency parsing, distinguishing two main classes of models, i.e., generative and discriminative (see also section 2.5). Probabilistic generative dependency models define probability distributions over all valid dependency-structures. Discriminative models, instead, treat the problem of assigning the correct structure to a given sentence as a classification task.

In recent evaluations of supervised dependency parsing (Buchholz and Marsi, 2006; Nivre et al., 2007), generative approaches are consistently outperformed by discriminative models, on standard parsing evaluation metrics. On the other hand, generative models provide a better way to investigate linguistic hypotheses on the derivation of syntactic structures.

4.5.1 Probabilistic Generative models

Probabilistic Generative models for DS have played a minor role in computational linguistics compared to their impact in PS representation. We believe that they are still the most intuitive way to provide a formal account on how dependency-structures can be derived.

One of the most relevant existing studies concerning probabilistic generative models for DS is the work of Eisner (1996a,b), in which he proposes 4 probabilistic models (A, B, C, D) for DS parsing. The author makes use of a very efficient chart parsing algorithm (Eisner, 1997; Eisner and Satta, 1999; Eisner, 2000) for projective dependency parsing whose complexity is $O(n^3)$ (where n is the length of the sentence), and it has a strict correspondence to the CKY algorithm applied to the special transformation defined in Johnson (2007) (see §4.3.2).

Instead of relying on a chart parser, in our approach we decide to compare the

4 generative models proposed by Eisner (1996a,b) using a reranking methodology (introduced in §2.4). In section 4.6 we will implement a more elaborated model inspired by model C, and evaluate its performance on a standard evaluation benchmark.

All the models (A,B,C,D) make use of two Markovian processes to generate the left and right dependents of each node. In this process, dependents are generated inside-outwards: those which are closer to the governor node are generated first. In order to make the model consistent, the two sequences of dependents need to terminate with a special symbol, i.e., the stop symbol $\oplus$ introduced in chapter 2 (see also Collins models in §3.2.1). When the first left (or right) dependent is generated, the missing previously generated sibling is indicated with the null symbol $\oslash$. In three of the four models (A,B,D), a tagged sentence is generated in the first phase as a sequence of $\langle$word, pos-tag$\rangle$ pairs. This sequence is generated as a 2nd order Markovian process where every word (and its tag) is generated with probabilities that depend on previous 2 words (and their tags). In our illustration of those models we will skip the first phase and assume that a tagged sentence is already given.

When describing the 4 models (A,C,B,D), we will refer to the example structure illustrated in figure 4.7. The DS is rooted at the artificial node *EOS* (end of sentence), conventionally put at the end of the sentence.

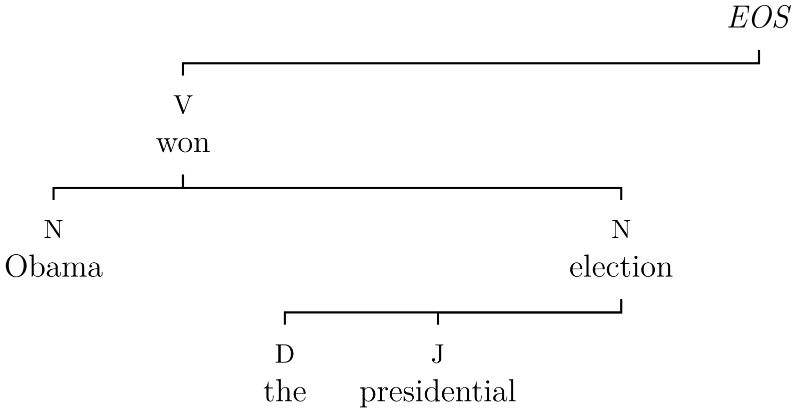


Figure 4.7: Dependency tree of the sentence *"Obama won the presidential election"*.

Model A For every word P in the sentence (from left to right), the model decides what are its left and right dependents, out of the list of the other tagged words in the sentence. More specifically, for each word D present to the left of P, the model decides whether or not D is a left dependent of P, conditioned on the previously chosen dependent S (situated on the same side). The same mechanism is done for the right dependents. For both left and right dependents, words in the sentence are examined from the one which is closest to P to the most remote one (inside-outwards). More concisely the derivation of a structure can be described as a multiset of $\langle$event, conditioning-context$\rangle$ pairs. A generic pair is described as follows:

$$\left\langle \boxed{\text{YES/NO},D,\text{left/right},P,S}, \boxed{D,\text{left/right},P,S} \right\rangle$$

Event: P has D as a left (right) dependent after S.

Conditioning-context: D is to the left (right) of P in the sentence, and S (or $\oslash$) is the previously chosen left (right) daughter of P.

Example: considering the structure in figure 4.7, the choice the model performs to decide that *Obama-N* is a left dependent of *won-V* is encoded in the $\langle$event, conditioning-context$\rangle$ pair

$$\left\langle \boxed{\text{YES,Obama-N,left,won-V},\oslash}, \boxed{\text{Obama-N,left,won-V},\oslash} \right\rangle$$

where $\oslash$ specifies that there was no other left dependent of *won-V* before *Obama-N*. Moreover, the fact that *EOS* is not a right dependent of *won-V* is encoded as

$$\left\langle \boxed{\text{NO},\mathit{EOS},\text{right,won-V,election-N}}, \boxed{\mathit{EOS},\text{right,won-V,election-N}} \right\rangle$$

Model C According to this model, words are generated top-down starting from the artificial *EOS* symbol. For each node P under consideration, the model decides the full set of its left and right dependents as a Markovian process: each dependent D of P is chosen conditioned on P and on the previously chosen left (right) dependent S. A generic $\langle$event, conditioning-context$\rangle$ pair can be described as:

$$\left\langle \boxed{D,\text{left/right},P,S}, \boxed{\text{left/right},P,S} \right\rangle$$

Event: D is chosen as the left (right) dependent of P, and S (or $\oslash$) was the previously chosen left (right) dependent of P.

Conditioning-context: S (or $\oslash$) is the last chosen left (right) dependent of D and it has a following sister (or $\oplus$).

Example: in the structure of figure 4.7, the choice that *Obama-N* is a left dependent of *won-V* is encoded as

$$\left\langle \boxed{\text{Obama-N, left, won-V,}\oslash}, \boxed{\text{left, won-V,}\oslash} \right\rangle$$

while the fact that *election-N* has no more left dependents after *the-D* is encoded as follows:

$$\left\langle \boxed{\oplus\text{, left, election-N,the-D}}, \boxed{\text{left, election-N,the-D}} \right\rangle$$

Model B This model is identical to model C in defining the list of dependents of each node. In addition, the model defines a further set of constraints to allow each node to specify certain preferences on the node that governs it. Apart from the events generated from model C the new multiset of $\langle\text{event}, \text{conditioning-context}\rangle$ pairs can be generically described as:

$$\left\langle \boxed{P\text{,}D}, \boxed{D} \right\rangle$$

Event: P is the parent node of D.

Conditioning-context: D needs a parent node.

Example: in the structure of figure 4.7, the choice that *Obama-N* wants as governor node *won-V* is encoded as

$$\left\langle \boxed{\text{won-V,Obama-N}}, \boxed{\text{Obama-N}} \right\rangle$$

Model D The last model is again similar to model C, except that this time each dependent D of a node P is selected from the words available in the sentence, conditioned on the previously chosen dependent S. More specifically D is chosen among the words which are to the left (right) of S in the sentence (plus the artificial node $\oplus$), or to the left (right) of P if S is $\oslash$ (no dependents have been chosen yet). This is substantially different from model C, since there the choice of D is made among the entire set of words including the ones not present in the sentence. A generic $\langle\text{event}, \text{conditioning-context}\rangle$ pair can be generically described as:

$$\left\langle \boxed{D\text{,left/right,}P\text{,}S}, \boxed{D\text{,left/right,}P\text{,}S} \right\rangle$$

Event: D is chosen as the left (right) dependent of P, following S (or $\oslash$).

Conditioning-context: S is the last chosen left (right) dependent of P, and D is the $\oplus$ node or it is situated to the left (right) of S (or P if S is $\oslash$) in the current sentence.

Example: in the structure of figure 4.7, the choice that *Obama-N* is a left dependent of *won-V* is encoded as

$$\left\langle \boxed{\text{Obama-N, left, won-V,}\oslash}, \boxed{\text{Obama-N, left, won-V,}\oslash} \right\rangle$$

while the fact that *election-N* has no more left dependents after *the-D* is encoded as follows:

$$\left\langle \boxed{\oplus\text{, left, election-N,the-D}}, \boxed{\oplus\text{,left, election-N,the-D}} \right\rangle$$

4.5.2 Discriminative models

In this section we will briefly review two standard approaches for dependency parsing using discriminative models: Maximum Spanning Tree (MST) and Transition-based models.

Maximum Spanning Tree

Maximum Spanning Tree (MST) models (McDonald et al., 2005; McDonald, 2006) efficiently keep track of all possible projective dependency-structures of a given sentence, and choose the one which maximizes a specific global function.

MST treats every dependency-structure as a multi-set of governor-dependent relations. Each edge is described as an high dimensional feature vector. For instance, if in a certain DS word i is the governor of word j, $\vec{v}(i,j)$ is the vector describing all the features of this relation (i.e., labels of the two words, their pos-tags, and other information including e.g., words in between them, and ancestral nodes). During the training phase the model learns a weight vector $\vec{w}$ (having the same dimension as $\vec{v}$) which is then used to find the best dependency-structure y for a given test sentence x. The score that needs to be maximized is defined as $\sum_{(i,j)\epsilon y} \vec{w} \cdot \vec{v}(i,j)$, and the best candidate is called the Maximum Spanning Tree.

During the training phase the weight vector $\vec{w}$ is calculated in order to optimize the scores of the tree structures in the training corpus. Roughly speaking, the algorithm starts with a random weight vector, and iteratively updates the current vector to a new one in order to reduce the number of errors it performs on predicting the MST for each sentence in the training treebank, until an optimal weight is found. More specifically, the algorithm employs a large-margin classifier, which, for every structure y in the training treebank, tries to keep the score of y above the scores of other incorrect structures of the same sentence by an amount which is proportional to how much they differ in accuracy.

For projective DS, both the training and parsing phase make use of a compact representation of the tree forest. For this representation it is possible to use efficient dynamic programming algorithms for computing the best structure in cubic time as discovered by Eisner (1997, 2000). This also holds for a more advanced and accurate second-order model, in which two consecutive edges are taken into consideration for each parsing decision, as opposed to a single edge in the base case.

MST can also deal with labeled DS, but, in this case, labels are added on top of the most probable DS of a given sentence as a separate task (also using a discriminative classifier). Attempts of adding labels as features of the structural model did not enhance performance on unlabeled dependency parsing. This is an important finding that justifies our choice of ignoring functional labels for detecting the bare structure of a sentence.

Transition-based models

Transition-based models (Nivre, 2003; Yamada and Matsumoto, 2003), decompose the problem of obtaining the best dependency-structure of a given sentence, as a sequence of local decisions. An input sentence is read in one word at a time, and at each step a specific operation is performed.

These models are inspired by shift-reduce parsers, where a *stack* is used to keep previously encountered words (partially connected) in a specific order. The exact set of operations vary from model to model, but generally, it includes the *reduce* operation, to pop out a word from the stack, the *shift* operation, to push the current word onto the stack, and the *attach* operation, to draw dependency relations between the current word and the word on the top of the stack (in either directions). Each operation can be applied only if certain conditions are met, to ensure that the final DS is connected and projective.

In order to deal with non-projective structures, some transition-based models make use of graph transformation (Nivre and Nilsson, 2005) as a second step or they allow permutation operations over the words in the stack (Nivre, 2009).

Transition-based parser usually employ discriminative classifiers (e.g., SVM) in order to decide what is the best transition or sequence of transitions to apply. Each transition is described by a big set of feature, and the model parameters are learned from the training corpus using standard techniques.

The main difference with respect to the MST approach is that Shift-Reduce parsers are incremental (they accept one word at a time) and therefore attempt to learn optimal local decisions, while MST has a bigger scope since it aims at finding the optimal structure as a whole.

4.6 Reranking generative models

In this section, we describe a reranking approach that combines a generative and a discriminative model and tries to retain the strengths of both. The idea of combining these two types of models through reranking is not new (see §2.5), although it has been mostly explored in constituency parsing (Collins and Duffy, 2002). This earlier work, however, used the generative model in the first step, and trained the discriminative model over its k-best candidates. In our framework we reverse the usual order of the two models, by employing a generative model to re-score the k-best candidates provided by a discriminative model. Moreover, the generative model of the second phase uses frequency counts from the training set but is not trained on the k-best parses of the discriminative model.

The reranking approach allows for efficiently evaluating many generative models, differing from one another on (i) the choice of the linguistic units that are generated (words, pairs of words, word graphs), (ii) the generation process (Markov process, top-down, bottom-up), and (iii) the features that are considered to build the event space (pos-tags/words, distance).

In our reranking perspective, all the generative model has to do is to compute the probability of k pre-generated structures, and select the one with maximum probability. In a generative model, every structure can be decomposed into a series of independent events, each mapped to a corresponding conditioning event. As an example, if a generative model chooses D as the right dependent of a certain word H, conditioned uniquely on their relative position, we can define the event as *D is the right dependent of H*, and the conditioning event as *H has a right dependent.*

As a preprocessing step, every sentence structure in the training corpus is decomposed into a series of independent events, with their corresponding conditioning events. During this process, our model updates two tables containing the frequency of events and their conditioning counterparts.

In the reranking phase, a given candidate structure can be decomposed into independent events $(e_1, e_2, \ldots, e_n)$ and corresponding conditioning events $(c_1, c_2, \ldots, c_n)$ as in the training phase. The probability of the structure can then be calculated as

$$\prod_{i=1}^{n} \frac{f(e_i)}{f(c_i)} \tag{4.1}$$

where $f(x)$ returns the frequency of x previously stored in the tables.[12]

It is important to stress the point that the only specificity each generative model introduces is in the way sentence structures are decomposed into events; provided a generic representation for the (conditioning) event space, both training phase and probability calculation of candidate structures can be implemented independently from the specific generative model, through the implementation of generic tables of (conditioning) events.

In this way the probabilities of candidate structures are exact probabilities, and do not suffer from possible approximation techniques that parsers often utilize (i.e., pruning). On the other hand the most probable parse is selected from the

[12] This probability estimate is the RFE introduced in §2.3.3.

set of the k candidates generated by the discriminative model, and it will equal the most probable parse among all possible structures, only for sufficiently high k.

4.6.1 Experiment Setup

Parser In order to generate a set of k-candidate structures for every test sentence, we use a state-of-the-art MST discriminative model[13] (McDonald, 2006). The MST parser was provided with the gold standard pos-tags of the words in the test set, and it was run in second-order and projective mode.

Treebank In our investigation, we have tested our model on the Wall Street Journal corpus Marcus et al. (1993) with sentences up to 40 words in length, converted to dependency-structures. Although several algorithms exist to perform such a conversion (e.g., Johansson and Nugues, 2007), we have followed the scheme in Collins (1999). Section 2-21 was used as training, section 22 as development set, and section 23 as test set.

Evaluation Metrics As the only evaluation metric for dependency parsing we have adopted the Unlabeled Attachment Score (UAS) (see Lin, 1995; Nivre et al., 2007), computed as:

$$\text{UAS} = \frac{\#\text{ words with correct governor}}{\#\text{ words}} \tag{4.2}$$

The root word of each parsed structure is considered to have the correct governor iff it is also the root of the sentence in the gold DS. The final UAS score is a macro-average over all the sentences in the test corpus.

4.6.2 Comparing the Eisner models

We have used the reranking framework just illustrated to compare the 4 generative models proposed by Eisner (1996a,b) presented in §4.5.1 on the development set of the WSJ. The results are shown in figure 4.8. All models show a progressive decrease in performance as the number of best candidates increases. The results for models B, C, and D are rather similar and consistently better than model A. This trend is consistent to the original results of Eisner (1996a,b) although model C in our evaluation slightly outperforms the other two (in the original work model D was the best performing one).

[13] The MST parser is available at `http://sourceforge.net/projects/mstparser/`.

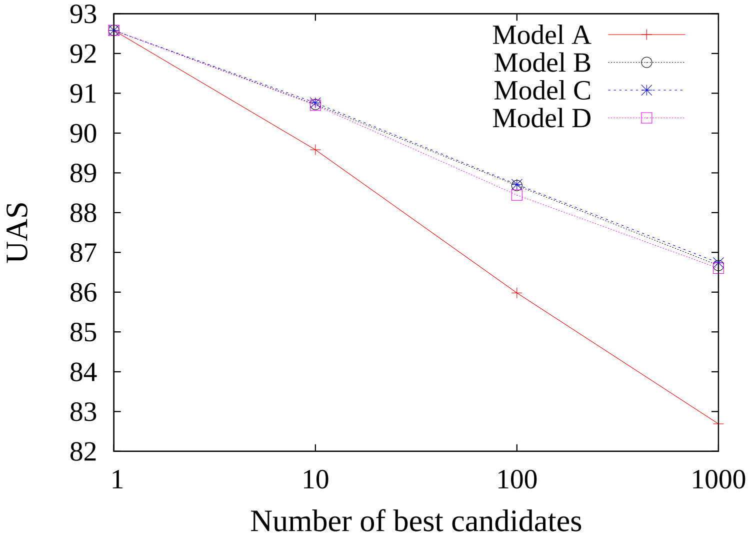


Figure 4.8: Comparison of the 4 generative models (A,B,C,D) illustrated in §4.5.1. The results for the first value in the x-axes (1) corresponds to the performance of the MST-parser (McDonald, 2006).

4.6.3 A new generative model

As a novel generative framework we have chosen to use a variation of model C. In this approach nodes are generated recursively in a top-down manner starting from the special symbol *EOS* (end of sentence). At any given node, left and right children are generated as two separate Markov chain sequences of nodes, each conditioned on ancestral and sibling information (which, for now, we will simply refer to as *context*). Every sequence ends with the stop symbol $\oplus$.

One of the relevant variations with respect to the original model is that in our version the direction of the Markov chain sequence is strictly left to right, instead of the usual inside outwards.

More formally, given a dependency-structure T, and any of its node N, the probability of generating the subtree $T(N)$ of the dependency-structure rooted in N is defined as:

$$\begin{aligned} P(T(N)) = & \prod_{l=1}^{L} P(\mathrm{N}_{\triangleleft l}|context) \cdot P(T(\mathrm{N}_{\triangleleft l})) \\ & \times \prod_{r=1}^{R} P(\mathrm{N}_{\triangleright r}|context) \cdot P(T(\mathrm{N}_{\triangleright r})) \end{aligned} \tag{4.3}$$

where L and R are the number of left and right children of N in T $(L, R > 0)$, $\mathrm{N}_{\triangleleft l}$ is the left daughter of N at position l in T (analogously $\mathrm{N}_{\triangleright r}$ for right daughters). The probability of the entire dependency-structure T is computed as $P(T(\mathit{EOS}))$.

In order to illustrate how a dependency-structure can be decomposed into events, we present in table 4.1 the list of events and the corresponding conditioning events extracted from a toy treebank with two DS trees including the one in figure 4.7. In this simple example, each node is identified with its word, and the context is composed of the direction with respect to the governor, the governor, and the previously chosen daughter (or ⊘ if it is the first). While during the training phase the event tables are updated with these events, in the test phase they are looked-up to compute the structure probability, as in equation 4.1.

Events	**Freq.**	**Conditioning Events**	**Freq.**
won L *EOS* ⊘	1	L *EOS* ⊘	2
⊕ L *EOS* won	1	L *EOS* won	1
⊕ R *EOS* ⊘	2	R *EOS* ⊘	2
Obama L won ⊘	1	L won ⊘	1
⊕ L won Obama	1	L won Obama	1
election R won ⊘	1	R won ⊘	1
⊕ R won election	1	R won election	1
⊕ L Obama ⊘	2	L Obama ⊘	2
⊕ R Obama ⊘	2	R Obama ⊘	2
the L election ⊘	2	L election ⊘	2
presidential L election the	1	L election the	2
⊕ L election presidential	1	L election presidential	1
⊕ R election ⊘	2	R election ⊘	2
⊕ L the ⊘	2	L the ⊘	2
⊕ R the ⊘	2	R the ⊘	2
⊕ L presidential ⊘	1	L presidential ⊘	1
⊕ R presidential ⊘	1	R presidential ⊘	1

Table 4.1: Events occurring when generating the dependency-structure in figure 4.7, for the event space (dependent | direction, governor, sister). The counts are extracted from a two-sentence corpus which also includes "Obama lost the election". According to the reported frequency counts, the structure has a associated probability of 1/4.

Model extension

In addition we have extended the model by including more contextual features and proper smoothing techniques for handling infrequent events and conditioning context.

In equation 4.3 we have generically defined the probability of choosing a daughter D based on specific features associated with D and the context in which it occurs. In our implementation, this probability is instantiated as in equation 4.4. The specific features associated with D are: the distance[14] $dist(H,D)$ between D and its governor H, the flag $term(D)$ which specifies whether D has more dependents, and the lexical and postag representation of D. The context in which D occurs is defined by features of the governor H, the previously chosen sister S, the grandparent G, and the direction dir (left or right).

In order to implement smoothing, equation 4.4 is factorized in four terms, each employing an appropriate backoff reduction list reported in descending priority. In the reduction lists, $wt(N)$ stands for the string incorporating both the pos-tag and the word of N, and $t(N)$ stands for its pos-tag. This second reduction is never applied to closed class words. All the notation and backoff parameters are identical to Eisner (1996a), and described in more details in Appendix B.2.

$$\begin{aligned}
& P(D|context) = \\
& P(dist(H,D), term(D), word(D), tag(D)|H,S,G,dir) = \\
& P(tag(D)|H,S,G,dir) \\
& \qquad \text{reduction list: } \begin{array}{|l|}\hline wt(H),wt(S),wt(G),dir \\ \hline wt(H),wt(S),t(G),dir \\ \hline \left\{\begin{array}{l} wt(H),t(S),t(G),dir \\ t(H),wt(S),t(G),dir \end{array}\right. \\ \hline t(H),t(S),t(G),dir \\ \hline \end{array} \\
& \times P(word(D)|tag(D),H,S,G,dir) \\
& \qquad \text{reduction list: } \begin{array}{|l|}\hline wt(H),t(S),dir \\ \hline t(H),t(S),dir \\ \hline \end{array} \\
& \times P(term(D)|word(D),tag(D),H,S,G,dir) \\
& \qquad \text{reduction list: } \begin{array}{|l|}\hline tag(D),wt(H),t(S),dir \\ \hline tag(D),t(H),t(S),dir \\ \hline \end{array} \\
& \times P(dist(P,D)|term(D),word(D),tag(D),H,S,G,dir) \\
& \qquad \text{reduction list: } \begin{array}{|l|}\hline word(D),tag(D),t(H),t(S),dir \\ \hline tag(D),t(H),t(S),dir \\ \hline \end{array}
\end{aligned} \tag{4.4}$$

4.6.4 Results

Results for the development set are reported in table 4.9, as unlabeled attachment score (UAS). The MST dependency parser obtains very high results when employed alone (92.58%), and generates a list of k-best-candidates which can potentially achieve much better results (an oracle would score above 95% when

[14] In our implementation distance values are grouped in 4 categories: $1, 2, 3-6, 7-\infty$.

k-best	Oracle best	Oracle worst	Reranked
1	**92.58**	92.58	92.58
2	94.22	88.66	92.89
3	95.05	87.04	93.02
4	95.51	85.82	93.02
5	95.78	84.96	93.02
6	96.02	84.20	93.06
7	96.23	83.62	**93.09**
8	96.40	83.06	93.02
9	96.54	82.57	92.97
10	96.64	82.21	92.96
100	98.48	73.30	92.32
1000	99.34	64.86	91.47

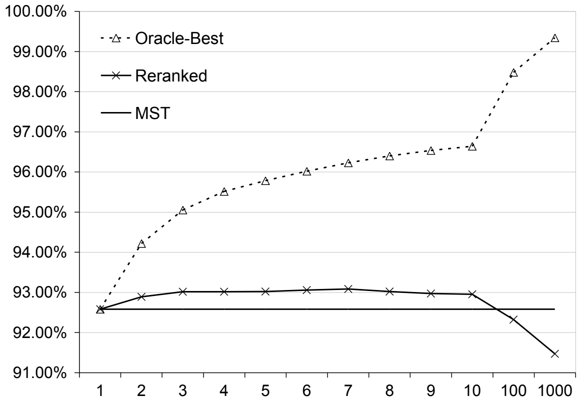


Figure 4.9: UAS accuracy of the MST discriminative and reranking parser on section 22 of the WSJ. *Oracle best*: always choosing the best result in the *k*-best, *Oracle worst*: always choosing the worst, *Reranked*: choosing the most probable candidate according to the generative model.

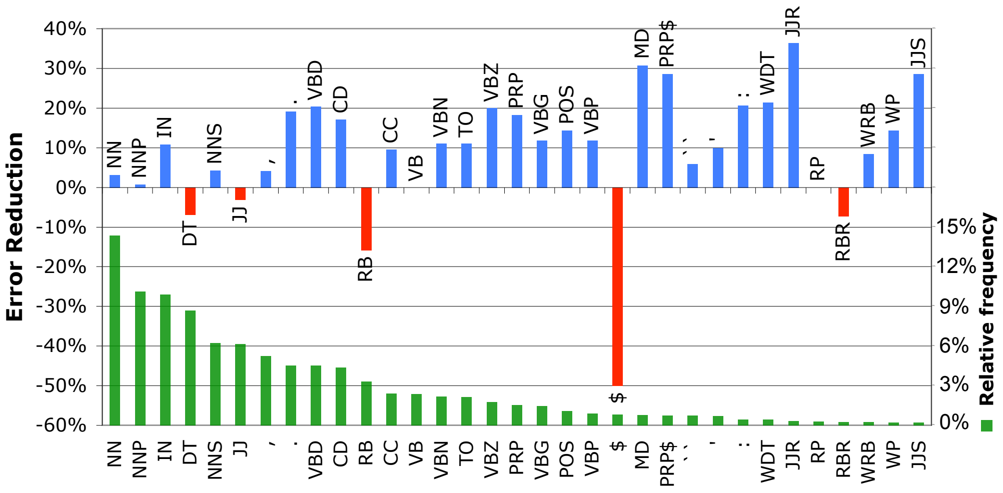


Figure 4.10: Error reduction analysis between the reranked 7-best and the MST 1-best on the different pos-tags (above), and relative frequencies of the various pos-tags in the test set (below).

selecting from the first 5-best, and above 99% from the first 1000-best). The decrease in performance of the generative model, as the number of the candidate increases, suggests that its performance would be lower than a discriminative model if used alone. On the other hand, our generative model is able to select better candidates than the MST parser, when their number is limited to a few dozens, yielding a maximum accuracy for $k = 7$ where it improves accuracy on the discriminative model by a 0.51% (around 7% error reduction). In figure 4.10 we present a per-category analysis on the error reduction of our best results compared to those of the MST parser.

We have evaluated the performance of the same model on the test set of the WSJ (sec 23), when reranking the 7-best candidates provided by the MST parser. The improvement here is smaller (0.11%: from 92.42% of the MST parser to 92.53%) but proves that the model is robust when reranking a restricted set of candidates.

4.7 Conclusions

In this chapter we have presented the notion of dependency-structure, and compared it with phrase-structure representation. Overall, the two representations can be considered as two ways to represent syntactic structures from two different perspectives: while PS defines a recursive way to group words of a sentence into constituents, DS specifies what are the dependency relations between words in a sentence.

The main contribution of this chapter was to propose a general framework for dependency parsing based on a combination of discriminative and generative models. We have used this framework to evaluate and compare several generative models, including those of Eisner (1996b) and some of their variations. Consistently with earlier results, none of these models performs better than the discriminative baseline when used alone. We have presented an instantiation of this framework in which our newly defined generative model leads to an improvement of the state-of-the-art parsing results, when provided with a limited number of best candidates. This result suggests that discriminative and generative model are complementary: the discriminative model is successfully at filtering out "bad" candidates, while the generative model is able to further refine the selection among the few best candidates. In our set-up it is now possible to efficiently evaluate many other generative models and identify the most promising ones for further investigation. Given the abundance of discriminative models for dependency parsing (Nivre, 2003; McDonald, 2006), our promising results show that generative models can still produce competitive results and therefore the pessimism about the prospects of probabilistic generative dependency models is premature.

4.8 Future Directions

In §1.2 we have illustrated how the syntactic domain, which is the main focus of the thesis, does not account for the full spectrum of phenomena which should be regarded when processing natural language expressions. In particular, although much work has supported the independence between syntax and semantic processing (Jackendoff, 2002), there is empirical evidence about the benefit of combined models for obtaining better disambiguation (Mitchell and Lapata, 2009). In particular, many mistakes of state-of-the-art parsers are on preposition attachment, and relying on semantical preferences of verbs and nouns could help to solve a big number of those ambiguities. The possibility of combining syntactic and semantics models could also contribute to shedding light on the opposite interconnection between the two domains. Current distributional semantics models are usually uninformed about the syntax underlying the set of sentences used for training. It would be therefore interesting to study how syntax can provide useful clues to obtain better semantic space models.

All parsing models that we have developed in this chapter were trained on the WSJ treebank which is relative small and domain-specific. The main shortcoming for this is that, although our DS models heavily rely on direct dependency relations between lexical items, they are quite restricted to work with a rather concise vocabulary. Possible extensions of our models could attempts to overcome this limitation by relying on semi-supervised learning techniques, in which the models can obtain additional information from surface counts extracted from large unannotated corpora such as the web (Lapata and Keller, 2004; Bansal and Klein, 2011).

Finally, in chapter 3 we have described an efficient algorithm for extracting all arbitrarily large syntactic constructions which occur multiple times in a large treebank. Although our implementation is specific for PS treebanks, it would not be difficult to extend it to the DS representations. In particular in a dependency-structure treebank it would be interesting to detect all recurring partial-fragments, i.e., any connected subset of nodes recurring in several trees of a treebank (Sangati et al., 2010). These constructs could be useful for deriving novel generative models for parsing DS.

Chapter 5

Tesnière Dependency-Structure

On peut ainsi comparer le verbe à une sorte d'atome crochu susceptible d'exercer son attraction sur un nombre plus ou moins élevé d'actants, selon qui'il comporte un nombre plus ou moins élevé de crochets pour les maintenir dans sa dépendance. Le nombre de crochets que présente un verbe et par conséquent le nombre d'actants qu'il est susceptible de régir, constitue ce que nous appellerons la valence du verbe.

We can also compare the verb to a sort of atom with hooks capable of exerting an attraction force on a certain number of elements, by means of an equal number of hooks which allow the atom to keep them under its dependence. The number of hooks of a verb and consequently the number of elements it is able to govern, constitute what we will call the valence of the verb.

Lucien Tesnière (1959, p.238)

5.1 Introduction

After having described in the previous two chapters syntactic models for learning phrase-structure and dependency-structure, in this chapter we introduce a novel syntactic representation that includes aspects from both PS and DS. The new representation was strongly inspired by the general theory of dependency syntax formulated by Lucien Tesnière (1959). Recent projects aiming at reinforcing the role of DS representation in computational linguistics (e.g., Yamada and Matsumoto, 2003; Forst et al., 2004; Johansson and Nugues, 2007) typically refer to Tesnière as the father of dependency syntax, but little attempt has been made to explain how the chosen dependency representation relates to the original work. A careful comparison reveals substantial differences: modern DS retains only the main idea proposed by Tesnière, namely the relation of dependency between words (§5.2.1), while other operations and features of the original theory are discarded or not overtly represented.

In this chapter we propose a formalization of Tesnière's theory, and derive an automatic conversion of the English Wall Street Journal Treebank into a novel representation: the Tesnière Dependency-Structure (TDS). This work is based on a joint collaboration with Chiara Mazza (i.e., Sangati and Mazza, 2009).

The current chapter is structured as follows: In §5.2 we retrace the original work by Tesnière and introduce the newly proposed TDS representation. In particular we reintroduce three key concepts: the division of a sentence into *blocks* of words, which act as intermediate linguistic units (§5.2.2), the *junction* operation, to handle coordination and other types of conjoined structures (§5.2.3), and the operation of *transference*, to generalize over the categories of the linguistic elements (§5.2.4).[1] In §5.3 we give empirical evidence that shows how this representation can incorporate all main advantages of modern PS and DS, while avoiding well known problems concerning the choice of heads and better representing common linguistic phenomena such as coordination. In §5.4 we describe the procedure used to convert the WSJ into TDS notation, and in §5.5 we propose a novel generative model for parsing the TDS treebank, and utilize a reranking framework for testing it. We also introduce a novel evaluation scheme for measuring the accuracy of the generated parse trees, focusing on the 3 main components of the TDS scheme: blocks, dependency, and coordination. Finally, in §5.6 we compare the TDS conversion with other proposed conversions of the WSJ treebank. In particular in §5.7 we report on a detail comparison between the TDS-bank and the CCG-bank (Hockenmaier and Steedman, 2007) focusing on the detection of coordination constructions.

[1] See in Tesnière (1959) part I ch. 22 on *nucléus*, part II on *jonction*, and part III on *translation*. We choose *transference* for the original French word *translation* to avoid any misunderstanding with the other meaning of the word *translation* in English. Unfortunately, 50 years after its publication, there is still no English translation of the author's work, yet there is for Russian, German, and Italian.

5.2 Dependency-Structures à la Tesnière

5.2.1 The dependency relation

The main idea behind Tesnière's model is the notion of *dependency*, which identifies the syntactic relation existing between two elements within a sentence, one of them taking the role of governor (or head) and the other of dependent (*régissant* and *subordonné* in the original terminology). Dependency is the only main feature of Tesnière's formalism used in standard dependency representations as explained in chapter 4.

Tesnière schematizes dependency relation using a graphical representation, the *stemma*, placing the governors above the dependents, as exemplified in the left diagram of figure 5.1. On the right side of the figure we present the same sentence using our notation, incorporating all the main features introduced by Tesnière, which we will explain in the following sections. The main divergence with respect to the original representation, is that in our trees we explicitly express the *linear order* of the words in the sentence, while in original stemmas Tesnière only represents the *structural order*: the dependents of a node follow a canonical order which might differ from their linear order in the sentence (for instance a verb has always the subject as the leftmost dependent).

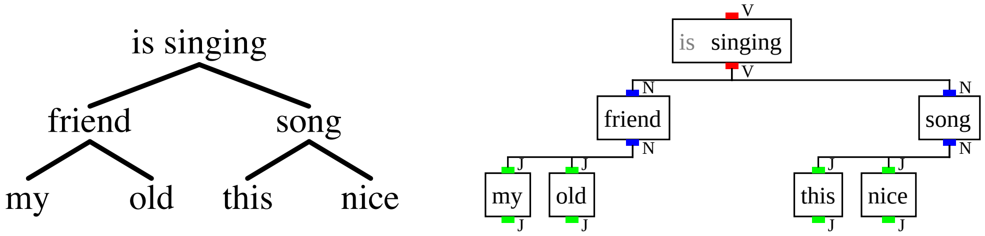


Figure 5.1: TDS of the sentence "*My old friend singed this nice song*", in Tesnière notation (left) and in our TDS representation (right).

5.2.2 Words, blocks and categories

In TDS, all words are divided into two classes: content words (e.g., nouns, verbs, adjectives, etc.), and functional words (e.g., determiners, prepositions, etc.). Each content word forms a *block*[2] which may additionally include one or more functional words, and it is on blocks that relations[3] are established. In our diagrams blocks are represented as black boxes, and functional words are written in grey to distinguish them from content words.

[2] Tesnière in (Tesnière, 1959, ch. 22) uses the term *nucléus*, and explains that it is important in order to define, among other things, the *transference* operation (see §5.2.4).

[3] We here refer to the *dependency* relation and the *junction* operation.

Tesnière distinguishes four *block categories* (or functional labels[4]), here listed together with the color and single letter notation, as reported in our diagrams: *nouns* (blue, N), *adjectives* (green, J), *verbs* (red, V), and *adverbs* (yellow, A).

Dependency relations are defined upon blocks. The verbal block represents the process expressed by the clause, and each of its arguments, representing a participant in the process, functions as a single noun. Arguments are usually compulsory within the clause, and their number is determined by the verb's *valence*. On the other hand the verb's adjuncts (or circonstants), represent the circumstances under which the process is taking place, i.e., time, manner, location, etc., and function as adverbs.

Tesnière was the first linguist borrowing the term *valency* from chemistry, to indicate the number of 'empty slots' that a verb should fulfill. So for instance intransitive verbs such as *to sleep* has valency 1 in 'the dog sleeps', since the subject is the only compulsory argument, while *to give* has valency 3 in 'he gives her a book'.

Nouns are always modified by blocks that function as adjectives, and adjectives by blocks that function as adverbs.

We will now introduce two operations, *junction* and *transference*, which allow to construct more complex clauses from simple ones.

5.2.3 Junction

The first operation is the *junction* and it is used to group elements that are coordinated in a sentence. As in the dependency relation, junction acts over blocks. In this case blocks are called the *conjuncts*, and are grouped into a unique entity which has the status of a block (a junction block). The conjuncts are horizontally connected in the TDS, belong to the same category, and are optionally connected by means of functional words, the *conjunctions*. In our TDS diagrams junction blocks are displayed in yellow to distinguish them from standard block (black). Conjunction words (or punctuation marks functioning as such) are also displayed in yellow.

Figure 5.2 displays three coordinated structures: in the first case, two nouns, subject of the same verb, are coordinated; in the second case, two adjectives, modifying the same noun, are coordinated; in the last case, two verbs, having the same subject, are coordinated. As in Tesnière's formulation, we use junction to represent standard coordination as well as apposition (e.g., *[the US president], [Obama]*). Moreover, the junction operation can work recursively: a junction block can be a conjunct of a bigger junction block (e.g., *[Mary and [either Mark or John]]*).

[4] We will use both terms interchangeably. Categories can be roughly seen as a simplification of both PoS tags and dependency relations in DS's. See also §5.3.2.

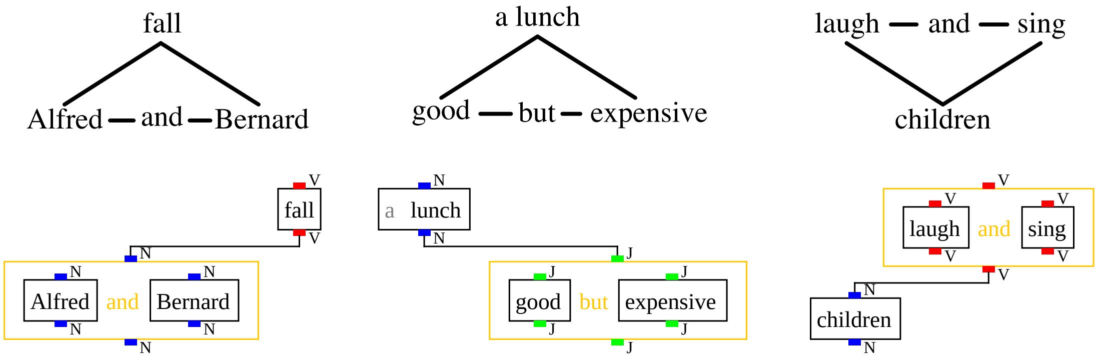


Figure 5.2: Examples of coordination. Tesnière's original notation is on top, and our notation at the bottom (we represent the junction with a yellow box).

5.2.4 Transference

The other operation is named *transference*. There are two types of transference. The *first degree transference* is a shifting process which makes a block change from the original category of the content word, to another. This process often (but not always) occurs by means of one or more functional words belonging to the same block, called *transferrers*. Figure 5.3 (left) shows an example of first degree transference. The word *Peter* is transferred from the word class *noun* and takes the functional label of an *adjective* via the possessive clitic *'s* which acts as a transferrer. In our representation (bottom), every block has two little colored boxes: the one at the bottom indicates the original category of the content word, and the one at the top indicates the category of the block after all transferences are applied.

The second degree transference occurs when a simple clause becomes an argument or an adjunct of another clause[5], maintaining all its previous lower connections, but changing its functional label within the main clause. The sentences below represent some examples of second degree transference:

(5.1) She believes *that he knows*

(5.2) The man *I saw yesterday* is here today

(5.3) You will see him *when he comes*

[5] In other words, the verb of the embedded clause becomes a dependent of another verb. This should not be confused with the case of compound verbs, which are represented as a single block, where auxiliaries are labeled as functional words (see for instance the TDS in figure 5.1).

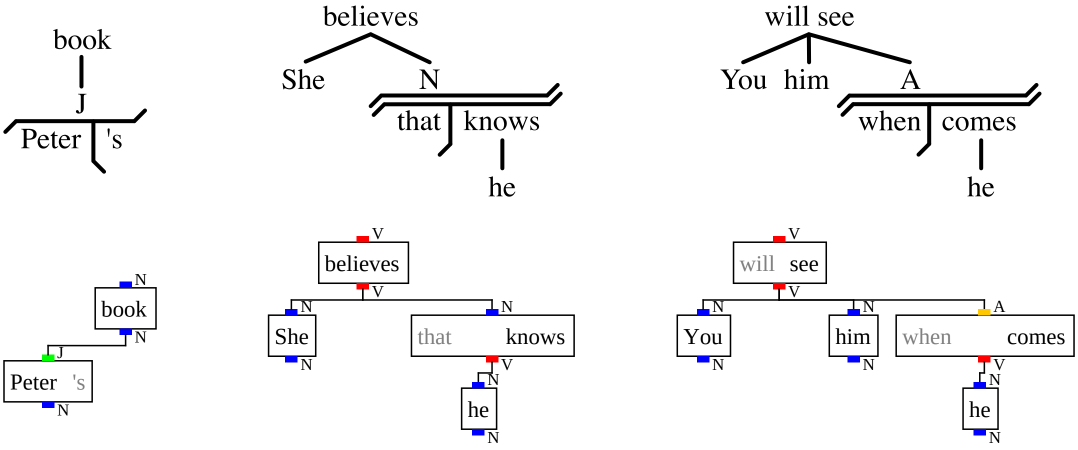


Figure 5.3: An example of *first degree transference* of the phrase "*Peter's book*" (left), and two examples of *second degree transference* of the sentence "*She believes that he knows*" (center) and the sentence "*You will see him when he comes*" (right).

In the first sentence, we have a transference *verb-to-noun* by means of the transferrer *that*. The embedded clause in italics takes the function of a noun, and becomes the object of the verb. Figure 5.3 (center) shows the corresponding TDS. The embedded clause in the second example functions as an adjective: it is a transference *verb-to-adjective* without any transferrer. The third sentence is an example of transference *verb-to-adverb*: the clause in italics has the functional role of a temporal adverb through the transferrer *when*. Figure 5.3 (right) shows the corresponding TDS.

5.3 Comparing TDS with DS

In this section we will describe three main advantages of using TDS notation as an alternative of DS representations (see chapter 4). In particular we will discuss the issue of choosing the linguistic heads in PS trees (§5.3.1), compare how the two models categorize dependency relations (§5.3.2), and how they treat coordination (§5.3.3).

In order to compare the different representations, Figure 5.4 illustrates three structures of an English sentence: the original Penn WSJ PS tree, the same structure converted to DS as in Johansson and Nugues (2007), and the TDS our conversion algorithm generates.

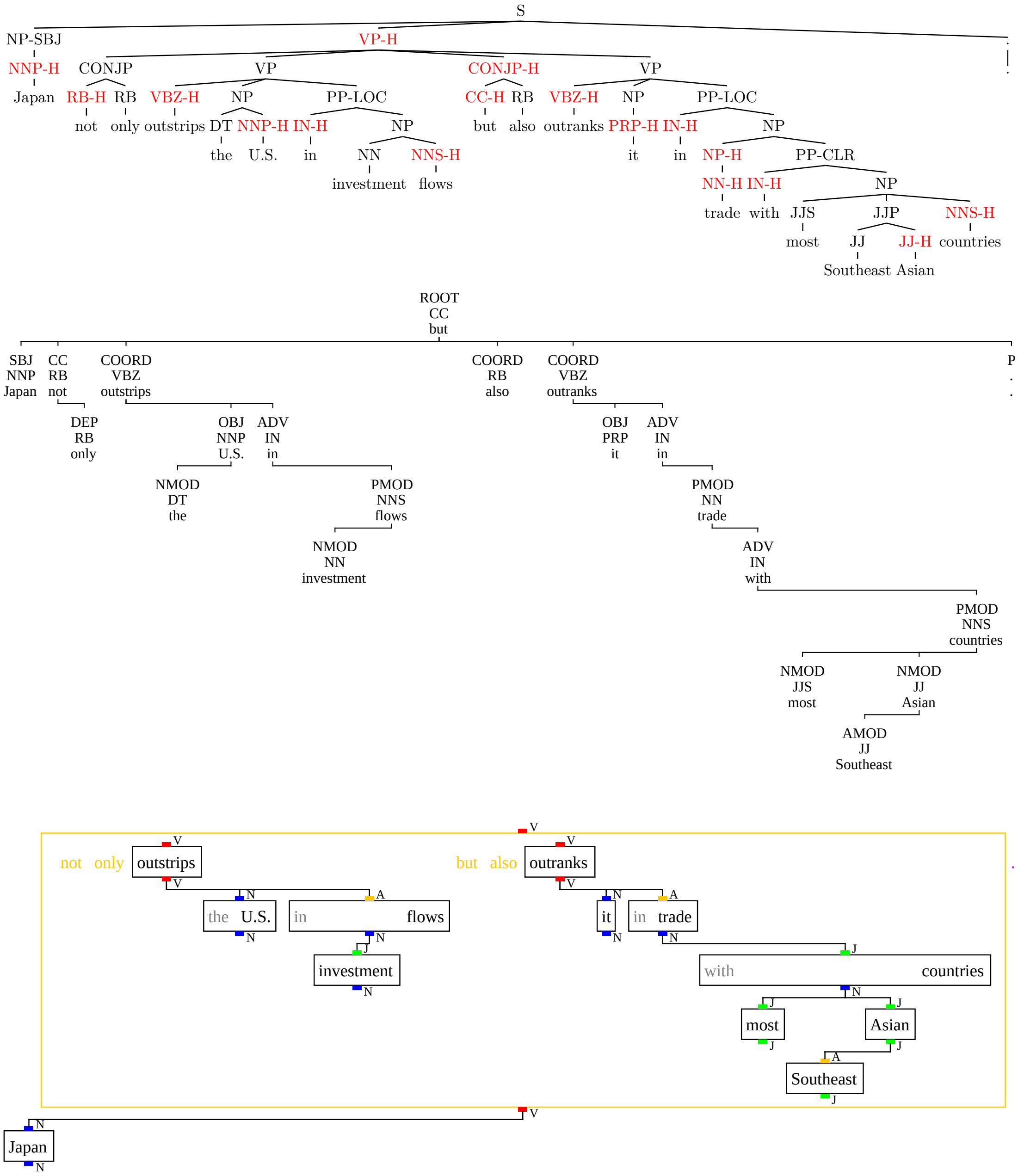


Figure 5.4: Comparison between different representations of an English sentence. **Top**: original WSJ PS taken from the WSJ sec-00 (#666). Null productions and traces have been removed. The red labels are the heads according to the DS below. **Center**: DS according to Johansson and Nugues (2007) using the *pennconverter* script in *conll2007* mode. Every word in the DS is presented together with its PoS and the label of the dependency relation with its governor. **Bottom**: TDS our conversion algorithm generates.

5.3.1 Choosing the correct heads

As for DS, the easiest way to construct a TDS tree is to derive it automatically from a PS tree. A fundamental operation that both DS and TDS transformation have in common is the annotation of the original PS tree with head labels (see also §3.2.1 and §4.3.1). This procedure has been initially proposed by Magerman (1994) and then slightly modified by others (e.g., Collins, 1999; Johansson and Nugues, 2007). If exactly one unique head is chosen for every constituent of the PS, the enriched tree can be shown to be homomorphic to a single projective DS (Hays, 1960; Nivre, 2006). For example, in figure 5.4, the DS in the center is derived from the heads marked with suffix -H in the PS at the top of the same figure.

Choosing heads in a PS tree is a critical operation: although much linguistic literature is present on this issue (cf. Corbett et al., 2006), in NLP there have been only few attempts to empirically evaluate different heads assignments (i.e., Chiang and Bikel, 2002; Sangati and Zuidema, 2009). While certain choices are less disputed (e.g., the verb is unequivocally the head of simple clauses), the majority of the decisions are more controversial and they are usually between functional and content words. The most frequent cases are listed here:

- Determiner vs. noun in nominal phrases (e.g., *the man*).
- Preposition vs. noun in prepositional phrases (e.g., *on paper*).
- Complementizer vs. verb in sub-clauses (e.g., *I believe that it is raining*).

In TDS, all these choices become irrelevant: since every functional word is included in the block together with the content word it belongs to, both decisions lead to the same TDS representation. Nevertheless, in TDS, head assignment remains essential when two or more content words are sister nodes of the same constituent, such as in "the song which I like". For these cases there is more consensus about the choice of the heads: in the example the verb is consistently marked as the head of the subclause,[6] and in our representation we follow this convention.

5.3.2 Categories and Blocks

Currently used DS representations make use of labels to identify the dependencies between words. For example *SBJ* and *OBJ* are used to mark the relation between a verb and its subject and direct object respectively, while *NMOD* is used to identify the role of a noun modifier (i.e., adjective). These labels are only partially

[6] For these cases few people, such as Bauer (1979), have proposed that the subject could also be the head of the sentence. For more discussion on this issue see also Anderson (1971) and Vater (1975).

overlapping with the four categories proposed by Tesnière as for instance noun-phrases are marked as nouns regardless of their role with respect to the verb of the sentence (e.g., subject or object). Moreover, while DS uses around a dozen of different labels, TDS uses only four. This turns out to be beneficial for a more simplified and generalized analysis.

The other difference is more subtle. In DS every word is a node, and therefore, for every node (except for the root) there is the need to identify the label of the dependency relation with its governor. The problem here is related to the above discussion about the choice of heads. If we take the example in figure 5.3 (center), one has to choose whether the complementizer or the verb is the direct object of the main verb. TDS better represents these cases, by including both elements in the same block. This choice is justified by the fact that both elements contribute to making the node an argument or an adjunct of the verb.

5.3.3 Coordination

Coordination is a very productive phenomenon in language. In the WSJ almost every other sentence presents a coordination structure. Unfortunately coordination represents one of the major problems in standard DS representation (see Nivre, 2006, p. 49). If dependency[7] is the only operation available to relate words, two main strategies are adopted, leading to 4 possible annotation schemes:

1. One conjunction (or conjunct) is the head of the other elements.
2. Each element (conjunction or conjunct) is the head of the adjacent element which follows.

The first solution is the one which is more commonly adopted in current PS-to-DS conversions. The second one is proposed by Igor Mel'čuk (1988). Both solutions are problematic in circumstances such as the one of figure 5.4 (see also figure 5.9). If the coordination includes multiple conjunctions, assigning the head to either one of the conjuncts or one of the conjunctions, leads to a strong asymmetry in the structure: either the conjuncts are not all at the same level, or the set of dependents includes both conjunctions and conjuncts. Moreover, if the coordination phrase is coordinating verbs at the top of the sentence structure, other potential blocks, e.g., the subject *Japan* in the example, will also appear in the set of dependents, at the same level with the verbs they depend on.[8] Finally the *conjunction phrase*, i.e., a group of words forming a single conjunction (e.g.,

[7] We only consider the case of single headed DS, i.e., each word should have exactly one governor.

[8] The labels of the dependency relations, such as the ones in the DS of figure 5.4, can often help to differentiate between dependents which have the same head, but differ in their functional labels. However they cannot be considered an optimal solution, since they do not eliminate the structural asymmetry.

not only in the example), is also poorly represented in DS representations, since it is not grouped into a unique entity.

Tesnière's choice of adding a special operation to handle coordination is justified if we consider how well it represents all the cases DS fails to describe consistently. Coordination in TDS can be seen as a borrowing of the notion of constituency from PS notation: the different blocks being conjoined have equal status, they govern all the blocks being dominated by the coordination block, and are dependents of all blocks the coordination structure depends on.

5.4 Converting the Penn WSJ in TDS notation

We will now present our attempt of converting the Penn WSJ treebank (Marcus et al., 1993) into TDS notation. In §5.4.1 we will introduce the elements composing each generated TDS, in §5.4.2 we will describe the conversion procedure, and in §5.6 we will compare the obtain treebank to other formalisms.

5.4.1 Elements of a TDS

Figure 5.5 illustrates the main elements, introduced in §5.2, which we need to define in order to construct TDS trees. Words are divided into content and functional words,[9] and blocks are either standard or junction blocks. A generic block contains a list of functional words, and a list of dependent blocks. In addition a standard block has to contain a unique content word,[10] while a junction block needs to specify a list of conjunction words and a list of conjunct blocks.

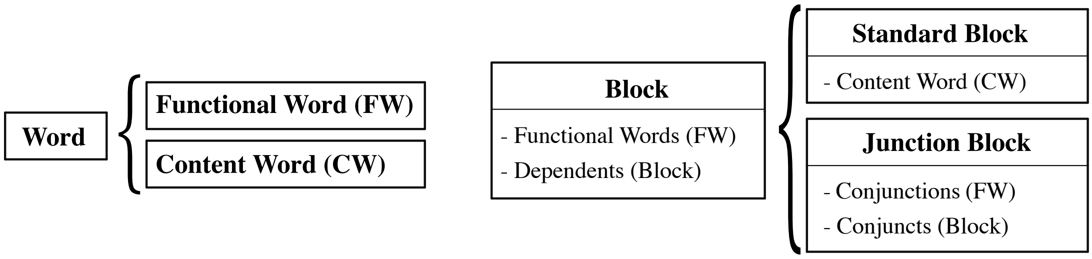


Figure 5.5: Hierarchical definition of words and block types in the TDS representation.

[9] A word is functional if its PoS is one of the following: punctuation marks, CC, DT, EX, IN, MD, POS, RP, SYM, TO, WDT, WRB. Moreover special pairs of words are marked as functional (e.g., more like, more than, even though, such as, many of, most of, rather than).

[10] As the only exception, contiguous proper nouns (e.g., first and last names) are parts of the same block.

5.4.2 The conversion procedure

In order to derive TDS trees from the Penn WSJ, we have decided to start from the original PS annotation, instead of using already converted DS trees. The main reason for this choice is that PS annotation of the WSJ is richer than currently available DS representations. This concerns in particular coordination structures, which would be much harder to reconstruct from DS notation (see §5.3.3).

Each PS in the corpus is preprocessed using the procedure described in Vadas and Curran (2007), in order to add a more refined bracketing structure to noun and adjectival phrases. Moreover, we remove null productions and traces from all trees, and enrich them with head labels. This last step is performed using a variation of the head-rules table defined in Magerman (1994) as reported in Appendix C.1.

Conversion

The pseudocode reported in algorithm 2, contains the procedure which is applied to each PS of the corpus, in order to generate the respective TDS. The algorithm recursively traverses from top to bottom each node of a PS tree, and outputs either a junction block (part A of the algorithm) or a standard block (part B). A constituent is identified as a junction structure when it presents conjunctions elements (i.e., CC, CONJP), or when it is composed of subconstituents with the same labels, such as in the cases of appositions.

For instance, in the PS tree of figure 5.4, the conversion procedures detects that the highest verbal phrase (VP) is a coordination block, as it includes in its daughter nodes two conjunction phrases (CONJP). As the same VP is the head of the whole structure, the junction block becomes the root of the TDS tree; in addition, the siblings of VP (i.e., the noun phrase ‘Japan’) will become a dependent of the whole coordination block.

Post-processing

For each converted TDS, several post-processing steps are applied:

1. Join together all *compound verbs* into a unique block (e.g., [is eating], [has been running]). All verbs except the last are marked as functional words (auxiliaries).

2. Unify in a unique standard block all *contiguous proper nouns.*

3. Define the *original category* of each block. This category is specified by the PoS of its content word if it is a standard block, and by the original category of the first conjunct block, if it is a junction structure.

```
Algorithm: $Convert(N_{PS})$
Input: A node $N_{PS}$ of a PS tree
Output: A block $N_{TDS}$ of a TDS tree
begin
    instantiate $N_{TDS}$ as a generic block;
    if $N_{PS}$ is a junction then
        // A) Coordination:  output a junction block
        instantiate $N_{TDS}$ as a junction block;
        foreach node $D$ in children of $N_{PS}$ do
            if $D$ is a conjunct then
                $D_{TDS} \leftarrow Convert(D)$;
                add $D_{TDS}$ as a conjunct block in $N_{TDS}$;
            else
                $D_{lex} \leftarrow$ lexical yield of $D$;
                if $D_{lex}$ is a conjunction then
                    add $D_{lex}$ as a conjunction in $N_{TDS}$;
                else
                    add $D_{lex}$ as a functional word(s) in $N_{TDS}$;
    else
        // B) No coordination:  output a standard block
        $N_h \leftarrow$ head daughter node of $N_{PS}$;
        if $N_h$ yield only one word $w_h$ then
            instantiate $N_{TDS}$ as a standard block with $w_h$ as its content word;
        else $N_{TDS} \leftarrow Convert(N_h)$;
        foreach node $D$ in children of $N_{PS}$ do
            if $D == N_h$ then
                continue;
            $D_{lex} \leftarrow$ lexical yield of $D$;
            if $D_{lex}$ are only functional words then
                add $D_{lex}$ as a functional word(s) in $N_{TDS}$;
            else
                $D_{TDS} \leftarrow Convert(D)$;
                add $D_{TDS}$ as a dependent of $N_{TDS}$;
    return $N_{TDS}$;
```

Algorithm 2: Pseudocode of the conversion algorithm from PS to TDS.

4. Define the *derived category* after transferences are applied. This category is specified by the original category of the governing block (if the current block is the root of the structure the category coincides with its original category). If the governing block is a noun or an adjective, the current block is an adjective or an adverb, respectively. If the governing block is a verb, the current block is either a noun or an adverb. This last decision depends on whether the original PS node, from which the current block derives, has a circumstantial label, i.e., it contains one of the following tags: ADVP, PP, PRN, RB, RBR, RBS, ADV, BNF, CLR, DIR, EXT, LOC, MNR, PRP, TMP, VOC.

The conversion procedure just described has been employed to generate a first TDS conversion of the Penn WSJ treebank. At current time a third revision of the conversion procedure has been released (version 1.2). The conversion and visualization tool, together with its technical documentation, is publicly available at `http://staff.science.uva.nl/~fsangati/TDS`.

In the following section we will describe and evaluate a generative model for parsing TDS constructions, while in §5.6 we will provide more qualitative analyses on the TDS treebank by comparing it with other proposed conversion of the WSJ treebank.

5.5 A probabilistic Model for TDS

This section describes the probabilistic generative model which was implemented in order to disambiguate TDS structures. Since no parser currently exists for the TDS representation, we have chosen the same strategy we have described in §2.4 and adopted for evaluating DS models in §4.6. The idea consists of utilizing a state of the art parser to compute a list of k-best candidates of a test sentence, and evaluate the new model by using it as a re-ranker, selecting the most probable structure among the given candidates. In order to obtain the list of k-best candidates, we utilize a state of the art parser for PS trees (Charniak, 1999), and transform each candidate to TDS.

5.5.1 Model description

In order to compute the probability of a given TDS structure, we make use of three separate probabilistic generative models, each responsible for a specific aspect of the structure being generated. The probability of a TDS structure (S) is obtained by multiplying its probabilities in the three models, as reported in equation 5.4.

The first model (equation 5.5) is the **Block Generation Model (BGM)**. It describes the event of generating a block B as a dependent of its parent block (governor). The dependent block B is identified with its categories (both original and derived), and its functional words, while the parent block is characterized

$$P(S) = P_{BGM}(S) \cdot P_{BEM}(S) \cdot P_{WFM}(S) \tag{5.4}$$

$$P_{BGM}(S) = \prod_{B \in depBlocks(S)} P(B|parent(B), direction(B), leftSibling(B)) \tag{5.5}$$

$$P_{BEM}(S) = \prod_{B \in blocks(S)} P(elements(B)|derivedCat(B)) \tag{5.6}$$

$$P_{WFM}(S) = \prod_{B \in stdBlocks(S)} P(cw(B)|cw(parent(B)), cats(B), fw(B), context(B)) \tag{5.7}$$

Table 5.1: Equation 5.4 gives the likelihood of a structure S as the product of the likelihoods of generating three aspects of the structure, according to the three models (BGM, BEM, WFM) specified in equations 5.5-5.7 and explained in the main text.

by the original category only. Moreover, in the conditioning context we specify the direction of the dependent with respect to the parent,[11] and its adjacent left sister (*null* if not present) specified with the same level of details of B. The model applies only to standard blocks.

The second model (equation 5.6) is the **Block Expansion Model (BEM)**. It computes the probability of a generic block B of known derived category, to expand to the list of elements it is composed of. The list includes the category of the content word, in case the expansion leads to a standard block. In case of a junction structure, it contains the conjunctions and the conjunct blocks (each identified with its categories and its functional words) in the order they appear. Moreover, all functional words in the block are added to the list.[12] The model applies to all blocks.

The third model (equation 5.7) is the **Word Filling Model (WFM)**, which applies to each standard block B of the structure. It describes the event of filling B with a content word (cw), given the content word of the governing block, the categories ($cats$) and functional words (fw) of B, and further information about the context[13] in which B occurs. This model becomes particularly interesting

[11] A dependent block can have three different positions with respect to the parent block: left, right, inner. The first two are self-explanatory. The *inner* case occurs when the dependent block starts after the beginning of the parent block but ends before it (e.g., *a nice dog*).

[12] The attentive reader might notice that the functional words are generated twice (in BGM and BEM). This decision, although not fully justified from a statistical viewpoint, seems to drive the model towards a better disambiguation.

[13] $context(B)$ comprises information about the grandparent block (original category), the adjacent left sibling block (derived category), the direction of the content word with respect to

when a standard block is a dependent of a junction block (such as ‘abortion’ in Figure 5.9). In this case, the model needs to capture the dependency relation between the content word of the dependent block and each of the content words belonging to the junction block.[14]

The conditional probabilities of the three models are smoothed using deleted interpolation as in the models for DS (see §4.6.3). More details are provided in Appendix C.3.

PoS & Block tagging None of the previous 3 models take into account the linear order of the words for deciding the locations of the block boundaries within a sentence. In order to improve robustness, we define 2 additional models for computing the probability of a given sequence of PoS-tags[15] and block-tags for the sequence of words of a given TDS structure. Both models are implemented as a tagging task with n-gram models as in chunking (Buchholz et al., 1999; Daelemans et al., 1999) and are shown in equations 5.8 and 5.9. In our case the possible block-tags are: N (new block independent from the previous block), I (continue the previous block), -N (new block inside the previous block), C (coordination word), and +I (continue the parent block of the previous block). An example of a pos-tagging and block-tagging is illustrated in figure 5.6.

$$P_{\textit{PoS-tags}}(S) = \prod_{i=1}^{n} P(word(i), pos(i) | word_{i-1}, pos_{i-1}, pos_{i-2}) \tag{5.8}$$

$$P_{\textit{Block-tags}}(S) = \prod_{i=1}^{n} P(\textit{block-tag}(i) | word_{i}, pos_{i}, pos_{i-1}, pos_{i-2}, pos_{i+1}) \tag{5.9}$$

5.5.2 Experimental Setup

We have tested our model on the WSJ section of Penn Treebank Marcus et al. (1993), using sections 2-21 as training and section 22 for testing. We employ the Max-Ent parser implemented by Charniak (1999), to generate a list of k-best PS candidates for the test sentences, which are then converted into TDS representation.

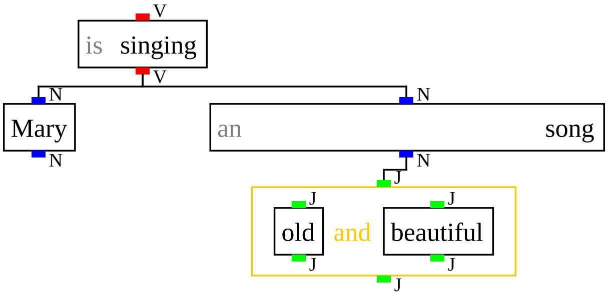
Instead of using Charniak’s parser in its original settings, we train it on a version of the corpus in which we add a special suffix to constituents which have circumstantial role.[16] This decision is based on the observation that the TDS

its governor (in this case only left and right), and the absolute distance between the two words.

[14] In order to derive the probability of this multi-event we compute the average between the probabilities of the single events which compose it.

[15] Although PoS-tags are not represented in the graphical version of TDS trees, they are kept in the internal representation from the original PS.

[16] Those which have certain function tags (e.g., ADV, LOC, TMP). The full list is reported in the post-processing procedure in §5.4.2. We were surprised to notice that the performance



Words	Mary	is	singing	an	old	and	beautiful	song
PoS-tags	NNP	VBZ	VBG	DT	JJ	CC	JJ	NN
Block-tags	N	N	I	N	-N	C	N	+I

Figure 5.6: **Above**: example of a TDS structure. **Below**: PoS-tags and block-tags associated to each word in the structure.

formalism captures the argument structure of verbs well, and we believe that this additional information might benefit our model.

We then applied our probabilistic model to re-rank the list of available k-best TDS, and evaluate the selected candidates using several metrics which will be introduced next.

5.5.3 Evaluation Metrics for TDS

The reranking framework described above, allows us to keep track of the original PS of each TDS candidate. This provides an implicit advantage for evaluating our system, viz. it allows us to evaluate the re-ranked structures both in terms of the standard evaluation benchmark on the original PS (F-score) as well as on more refined metrics derived from the converted TDS representation. In addition, the specific head assignment that the TDS conversion procedure performs on the original PS, allows us to convert every PS candidate to a standard projective DS, and from this representation we can in turn compute the benchmark evaluation for DS used in §4.6.1, i.e., the unlabeled attachment score (UAS).

Concerning the TDS representation, we have formulated 3 evaluation metrics which reflect the accuracy of the chosen structure with respect to the gold structure (the one derived from the manually annotated PS), regarding the different components of the representation:

(in terms of F-score) of the parser on this modified treebank is only slightly lower than the one obtained with standard settings (0.13%).

Block Detection Score (BDS): the accuracy of detecting the correct boundaries of the blocks in the structure.[17]

Block Attachment Score (BAS): the accuracy of detecting the correct governing block of each block in the structure.[18]

Junction Detection Score (JDS): the accuracy of detecting the correct list of content-words composing each junction block in the structure.[19]

	Beam	**F1**	**UAS**	**BDS**	**BAS**	**JDS**
Charniak	$k = 1$	89.4	92.5	95.0	89.5	77.6
PCFG-reranker	$k = 5$	89.0	92.4	95.1	89.2	77.5
PCFG-reranker	$k = 1000$	83.5	88.4	92.9	83.6	71.8
TDS-reranker	$k = 5$	**89.6**	92.4	95.0	89.4	**77.7**
TDS-reranker + PoS&Block-tags	$k = 5$	89.6	**92.5**	**95.2**	**89.5**	77.6
TDS-reranker	$k = 10$	89.0	92.1	94.7	88.9	76.5
TDS-reranker	$k = 100$	86.6	90.4	93.7	86.6	72.1
TDS-reranker	$k = 1000$	84.0	88.1	92.0	84.0	67.7
TDS-reranker + PoS&Block-tags	$k = 1000$	84.8	89.3	93.5	84.9	69.7

Table 5.2: Results of Charniak's parser, the TDS-reranker, and the PCFG-reranker according to several evaluation metrics, when the number k of best-candidates increases.

5.5.4 Results

Table 5.2 reports the results we obtain when reranking with our model an increasing number of k-best candidates provided by Charniak's parser (the same results are shown in the left graph of Figure 5.7). We also report the results relative to a PCFG-reranker obtained by computing the probability of the k-best candidates using a standard vanilla-PCFG model derived from the same training corpus. Moreover, we evaluate, by means of an oracle, the upper and lower bound of the

[17] It is calculated as the harmonic mean between recall and precision between the test and gold set of blocks, where each block is identified with two numerical values representing the start and the end position (punctuation words are discarded).

[18] It is computed as the percentage of words (both functional and content words, excluding punctuation) having the correct governing block. The governing block of a word, is defined as the governor of the block it belongs to. If the block is a conjunct, its governing block is computed recursively as the governing block of the junction block it belongs to.

[19] It is calculated as the harmonic mean between recall and precision between the test and gold set of junction blocks expansions, where each expansion is identified with the list of content words belonging to the junction block. A recursive junction structure expands to a list of lists of content-words.

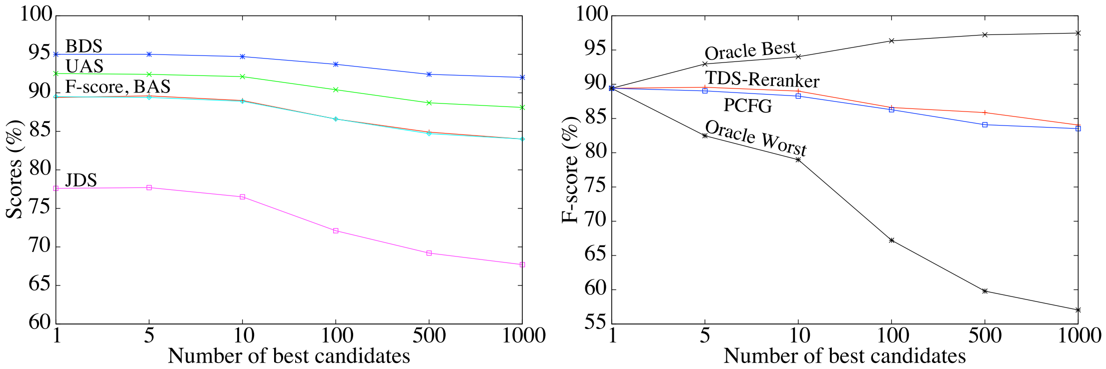


Figure 5.7: **Left**: results of the TDS-reranking model according to several evaluation metrics as in Table 5.2. **Right:** comparison between the F-scores of the TDS-reranker and a standard PCFG-reranker (together with the lower and the upper bound), with the increase of the number of best candidates. Results refer to the development section (22) of the WSJ.

F-Score and JDS metric, by selecting the structures which maximizes/minimizes the results.

Our reranking model performs rather well for a limited number of candidate structures. In particular, for $k = 5$, it is in par or slightly outperforms Charniak's model for all evaluation metrics. In general, we notice that our extended model including PoS-tagging and Block-tagging is more robust than our basic model, especially when reranking an high number of candidates.

The right graph in Figure 5.7 compares the F-score performance of the TDS-reranker against the PCFG-reranker. Our system consistently outperforms the PCFG model on this metric, as well as for UAS, BDS, and BAS. Concerning the JDS metric, as the number of k-best candidates increases, the PCFG model outperforms the TDS-reranker.

5.6 Other representations of the WSJ Treebank

In this section we illustrate other proposed representations of the Penn WSJ treebank. Some of these are obtained fully automatically, while others have made used of human annotators. In order to compare them and clarify better the contribution of each representation we will take into consideration an example tree from the WSJ treebank illustrate in figure 5.8. The corresponding TDS tree is presented in figure 5.9.

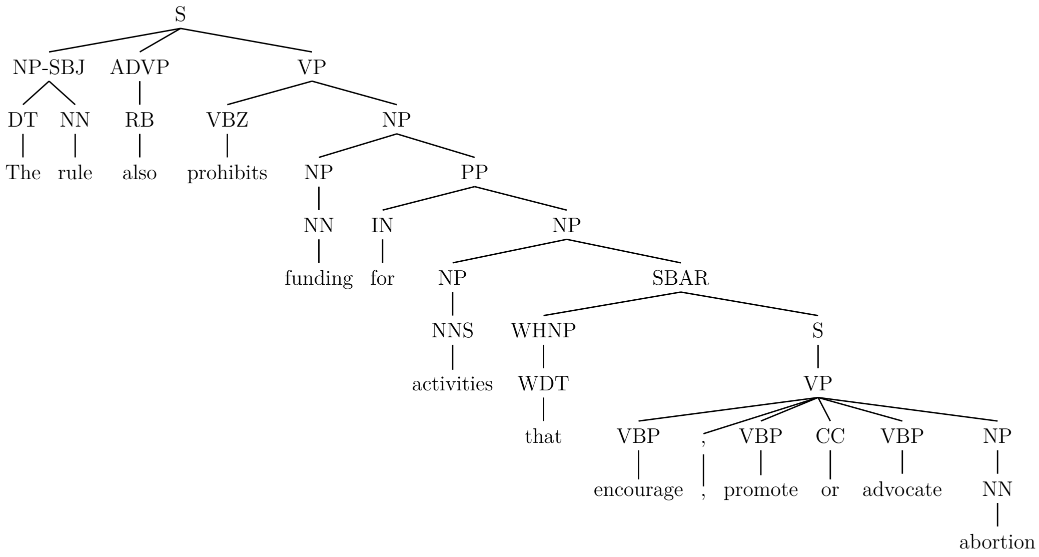


Figure 5.8: Example of a PS tree of the WSJ treebank (section 0, #977).

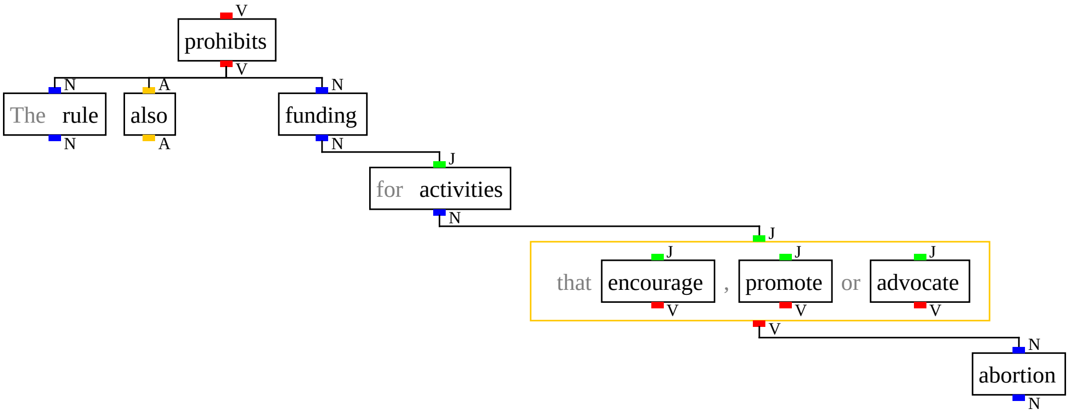


Figure 5.9: TDS tree converted from the PS tree in figure 5.8.

5.6.1 Prague English dependency treebank

The Prague English Dependency Treebank (Cinková et al., 2008, 2009), in short PEDT, consists of a semi-automatic conversion of a subset of the WSJ treebank into a dependency representation which follows the scheme of the Prague Dependency Treebank (Hajič et al., 2000). Every sentence has two annotation layers: the *analytical* layer which captures the dependency surface-syntax, and the *tectogrammatical* annotation, which encodes semantic relations between the content words in the sentence (the relations between content and functional words are only included in the analytical layer). While the former is very similar to standard labeled dependency grammar, the latter constitutes the main contribution of the proposed representation. The tectogrammatical representation derives from the Functional Generative Description tradition (Sgall et al., 1986), and aims at capturing the linguistic meaning of the sentence, with special focus on the communication role of language; for instance it includes underspecified elements which can be reconstructed from the context (e.g., in *I told you last night*, the verb has an hidden argument to indicate that *something* has been told.). This representation explicitly represents the argument structure of the sentence, and has been used to automatically extract the valency frames of English verbs (Semecký and Cinková, 2006).

Figure 5.10 shows the tectogrammatical representation[20] of the PS tree in figure 5.8. As in the TDS representation functional words (i.e., *The*, *for*) are represented together with the content words they refer to (i.e., *rule*, *activity*).

PDT representation employs only dependency relations to connect words.[21] As in standard labeled dependency treebanks, this creates some problem when dealing with complex coordination (involving multiple conjuncts and shared arguments). In the same figure, we can in fact notice that there is no explicit relation between any of the verbs in the coordinated structure (i.e., *encourage*, *promote*, *advocate*) and their arguments (i.e., *that*, *abortion*). This relations can be retrieved only indirectly from the labels (i.e., *RSTR.member*, *ACT*, *PAT*).

5.6.2 Stanford Typed Dependency Representation

The Stanford typed dependency representation (De Marneffe et al., 2006; De Marneffe and Manning, 2008) was developed with the aim of providing a suitable representation for relation extraction and language understanding tasks. The conversion is based on a completely automatic procedure which is part of the Stanford parser software.[22] The procedure takes as input any Penn-style PS tree-

[20] The PDT 2.0 treebank (both analytical and tectogrammatical layers) and visualization software are publicly available at `http://ufal.mff.cuni.cz/pdt2.0`. Many thanks to Jan Stepanek for technical assistance.

[21] Apart from standard governor-dependent relations it also encode anaphoric relations as shown in the same figure between *that* and *activity*.

[22] `http://nlp.stanford.edu/software/`

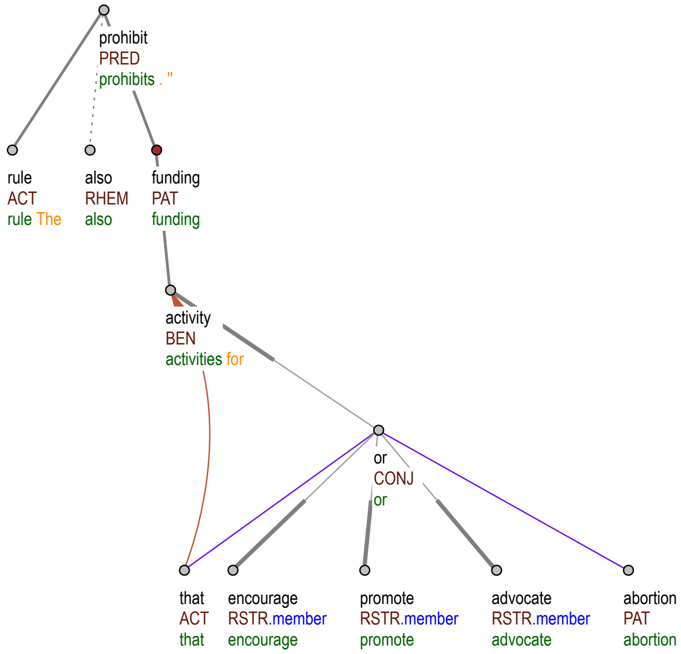


Figure 5.10: Tectogrammatical annotation of the PS tree in figure 5.8 according to the Prague English Dependency Treebank.

bank and can output dependency-structures in various formats, among which the *base format*, and the *collapsed format* which are reported in figure 5.11. The former is very similar to standard labeled dependency-structures, while in the latter most of the relations between the content words and the functional words are collapsed into the labels of the relations.[23] For instance in figure 5.11 the preposition *for* in *founding for activities* is collapsed and put into the label of the dependency relation between *founding* and *activities*. The collapsed representation in this respect is very similar to the PEDT; the two representations mainly diverge in the choice of the label hierarchy: the PEDT is more oriented towards argument structure relations, while the Stanford representation follows more traditional grammatical relations.

This representation has the same limitation as other dependency formalisms (e.g., PEDT) when dealing with coordination structures. The only main difference with respect to the PEDT, is that Stanford dependency-structures choose the first conjunct as the governor of a coordination construction.

[23] Not all the relations are collapsed. For instance prepositions are collapsed, while determiners are not.

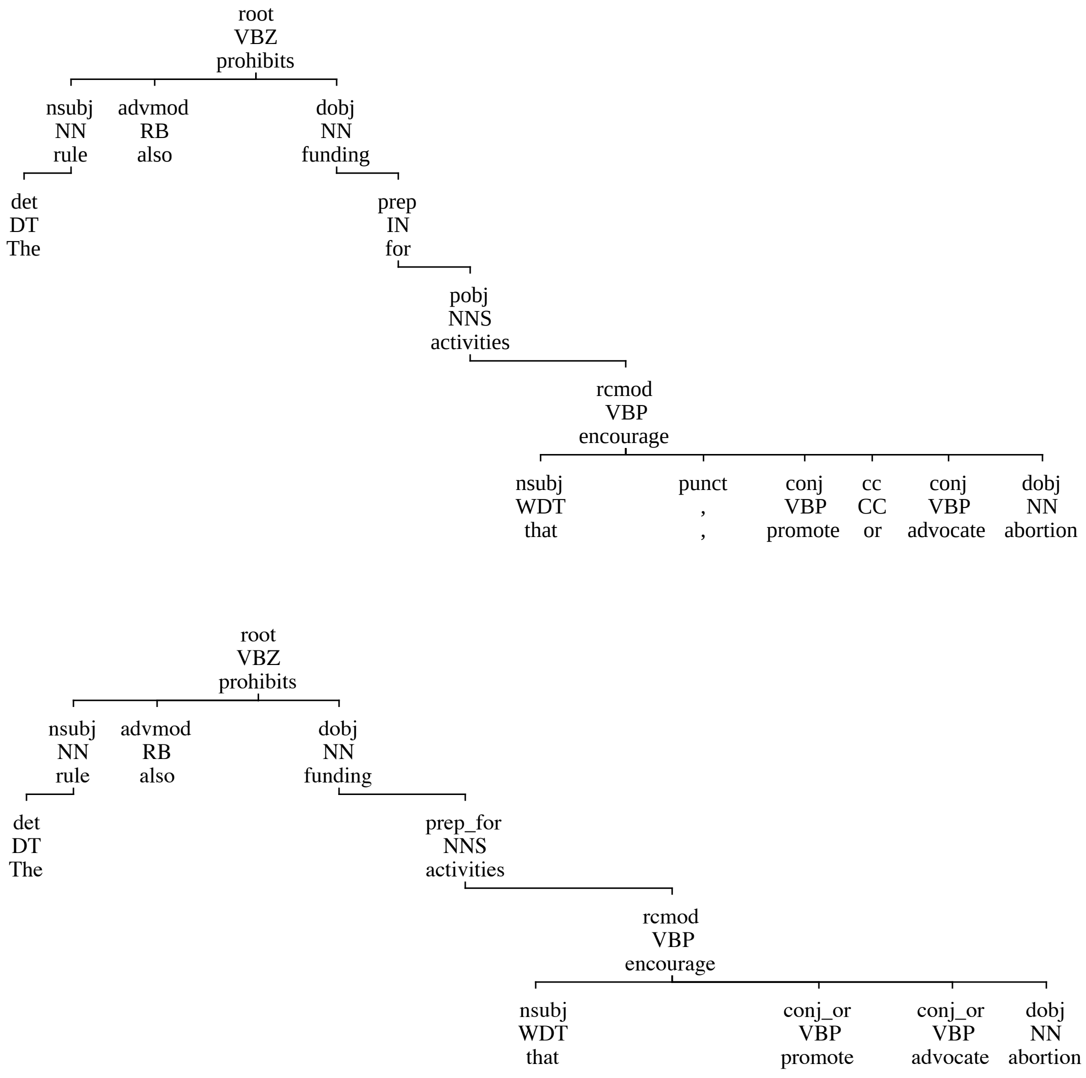


Figure 5.11: Stanford Dependency representation of part of the PS tree in figure 5.8, in basic format (above) and collapsed format (below).

5.6.3 Bubble Trees

Kahane (1997) described a general transformation of PS trees enriched with head labels to a novel representations called *bubble trees*. A bubble tree incorporates information from the original PS tree as well as the dependency relations implicit in the head annotation. Differently from standard PS to DS transforms (§4.3.1), the formalism allows to select *more than a single daughter* as head of a constituent. Figure 5.12 shows part of the bubble tree derived from PS tree in figure 5.8. In this representation every constituent is represented as a bubble (rectangles in the figure). If constituent D is the head daughter of constituent P, D is placed inside P as a sub-bubble; otherwise it is a dependent of P (indicated with an arrow going from D to P). Therefore, as in the TDS representation, dependency relations are not between words but between bubbles (blocks in TDS). If multiple daughters are co-heads of the same constituent P, they are all placed inside P as sub-bubbles. This allows to represent conjuncts of a coordination as sister node of the same bubble, as originally proposed by Tesnière. For instance in figure 5.12 the three VBP of the rightmost VP are co-heads and positioned as daughters of the VP bubble.[24] The use of both constituency relation and dependency relation, and the handling of coordination makes this formalism particularly similar to the TDS framework we have proposed. Although this transformation is insightful yet simple to derive, it was unfortunately never applied to any treebank, nor used for building parsing models.

5.6.4 The CCG-bank

Hockenmaier and Steedman (2007) developed a semi-automatic conversion of the WSJ treebank into Combinatory Categorial Grammar representation: the CCG-bank. CCG was introduced by Steedman (1996) as a generalization of Categorial Grammar theory (Ajdukiewicz, 1935; Bar-Hillel, 1953). It is a lexicalized grammar formalism, which can successfully represent a vast range of linguistic phenomena such as coordination, non local dependencies, control and raising constructions, without the use of movements and traces. Figure 5.13 represents part of the CCG-bank tree of the PS in figure 5.8. Internal constituents are assigned either basic categories (e.g., S, NP), or complex categories (e.g., S\NP, NP\NP, (NP\NP)/NP)). In general, a complex category of the form X/Y expects an argument Y to its right to produce a constituent of category X (*forward composition*), while those of type X\Y expect Y to their left to produce X (*backward composition*). For instance in the example tree of figure 5.13, NP\NP indicates a prepositional phrase which combined with an NP to its left produces an NP. Forward/backward composition are somewhat similar to the *transference* operation

[24] The same formalism allows to represent blocks of words and transference as in TDS (see §5.2.4) by assigning heads to both the functional part and the content part of a constituent (e.g., in the same figure, WHNP and S can be co-heads of SBAR).

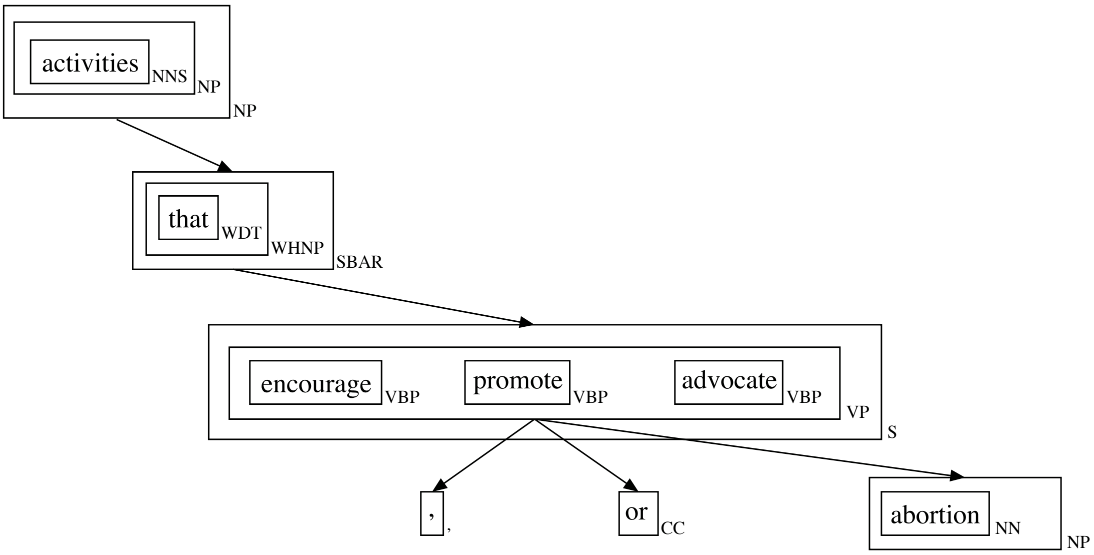


Figure 5.12: Bubble tree converted from the PS tree in figure 5.8.

in our TDS formalism (see §5.2.4). For instance in the TDS tree of figure 5.9, the preposition *for* combined with the noun *activities* returns a adjectival block.

The CCG-bank has been successfully used to train statistical parsers (e.g., Hockenmaier, 2003; Clark and Curran, 2007b), and it is an important framework for many natural language applications which need to retrieve the semantic structure of a sentence such as textual entailment and question-answering systems. In fact every CCG derivation of a sentence can be uniquely mapped into a logical form which represents the meaning of the sentence (given the correct lexical categories).

The CCG-bank follows a binary branching annotation. When dealing with a coordination of multiple conjuncts, such as the tree in figure 5.13, the construction is transformed into a right-branching structure. The intermediate categories being created are assigned the same categories of the conjuncts with an extra feature *[conj]*. Nevertheless, we conjecture that this representation of coordination might introduce some difficulties for parsing: it is very hard to capture the relation between 'advocate' and 'abortion' since they are several levels away in the structure. This discussion is related to the contrast between iterative processes, as in flat coordination structures, and recursive processes, as in embedding phrases within sentences (see also Hurford, 2004; Pinker and Jackendoff, 2005). In our TDS treebank we support the distinction between these two processes, and prefer to preserve the iterative configuration in coordination constructions.

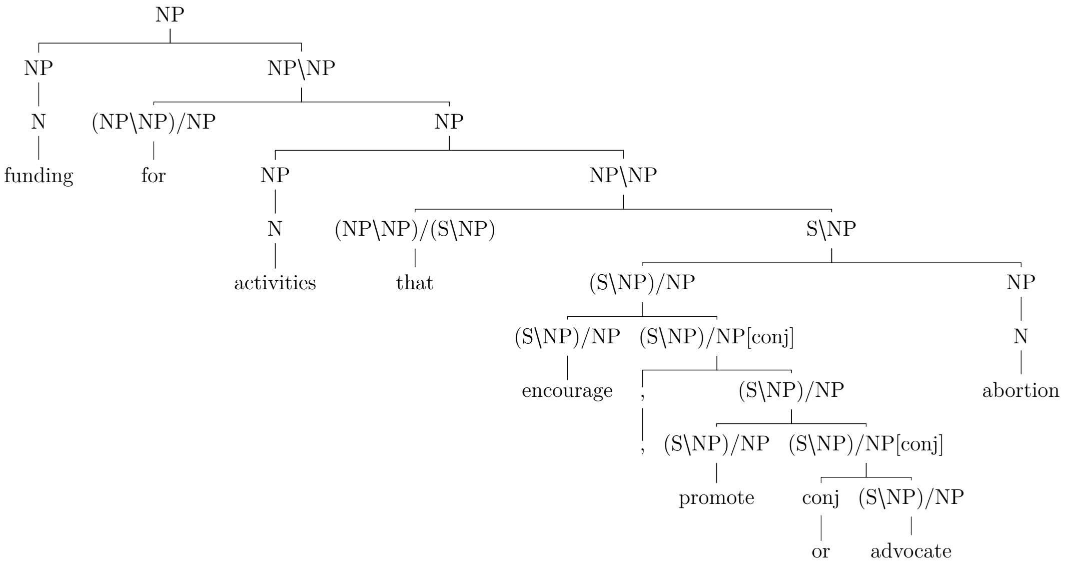


Figure 5.13: CCG representation of part of the PS tree in figure 5.8.

5.7 Assessment of the converted treebank

For a given syntactic representation it is always important to assess how well it is able to represent a specific linguistic phenomenon *in theory* and in *practice*. So far we have described how both the TDS treebank and the CCG-bank are based on theoretical formalisms which have one of the main strengths in the representation of coordination constructions. In order to complete our analysis, we report on an empirical study which compares the detection of coordination constructions between the two treebanks.

The automatic detection of coordination in the WSJ treebank is not an easy task: for instance when conjuncts and modifiers of the coordination are put at the same level (such as in the PS tree in figure 5.8), arguments can be wrongly identified as conjuncts. In order to compare the quality of the CCG and TDS treebank for what concerns the identification of coordination structures, we have identified all the coordination construction in section 0 of the WSJ treebank for which the TDS and CCG analyses where not agreeing in the detection of coordination (168 sentences). With the help of an external annotator[25] we have manually annotated the correct coordination structures of those sentence and detected the mistakes present in the two representations. As a quantitative sum-

[25] We are very grateful to Barend Beekhuizen for his help on annotating coordination constructions.

mary of this comparison, CCG presents 135 coordination mistakes,[26] TDS only 29, and there are 48 additional cases which are uncertain; more details of this comparison are illustrated in §C.2, while some examples of coordinated structures in TDS representation are reported in Appendix C.4.

5.8 Conclusion

In this chapter we have formalized a novel syntactic scheme (TDS) inspired by the work of Tesnière (1959), and developed a conversion algorithm to transform the Penn Wall Street Journal treebank into the new representation.

Corpus-based computational linguistics has often valued a good compromise between adequacy and simplicity in the choice of linguistic representation. The transition from PS to DS notation has been seen as a useful simplification, but many people have argued against its adequacy in representing frequent linguistic phenomena such as coordination. The TDS conversion presented in this chapter, reintroduces several key features from Tesnière's work: on one hand the operation of junction enriches the model with a more adequate system to handle conjoined structures (e.g., coordination); on the other hand, the blocks, the transference operation, and the category system further simplify and generalize the model.

We have presented a probabilistic generative model for parsing TDS syntactic representation of English sentences. For evaluating the parsing performance we have defined 3 metrics focusing on the different layers of the TDS representation (blocks, dependencies, coordinations). In particular we have introduced a specific metric for the detection of coordination construction, a linguistic phenomenon highly abundant in natural languages, but often neglected when it comes to evaluating parsing resources, and hope that other researchers might benefit from it in the future. Our parsing results are encouraging: the overall system, although only when the candidates are highly reliable, is in par or slightly improves on Charniak's parser on all the evaluation metrics.

Finally we have compared the TDS treebank with other existing conversions of the WSJ treebank, and assessed the quality of the conversion procedure with respect to the detection of coordination constructions.

5.9 Future directions

Given the availability of different syntactic representations for the sentences in the WSJ treebank, one could follow the same manual assessment procedure we have done for the detection of coordinating construction (see §5.7), while focusing on other shared linguistic phenomena (as in Rimell et al., 2009). The effort of comparing independently annotated resources is a stimulating task which brings

[26] In many case the CCG-bank misses the extra feature *[conj]*.

many insights in the investigated phenomena and leads to the corrections of many mistakes that occur in automatic as well as manual treebank annotations.

Moreover, as there exist a range of different syntactic representations that available parsers are able to work with, it would be ideal to derive more universally accepted evaluation criteria which could work across different representations. Several attempts have been made in the past (Clark and Curran, 2007a; Rimell et al., 2009), but they are not widely employed as they require substantial annotation effort. An easier alternative for comparing different representations on the same metrics, is to rely on transformation techniques. For instance, the availability of an automatic conversion from Penn-style PS trees to TDS representation, could allow us to test the performance of a range of state-of-the-art PS parsers on the more refined evaluation criteria we have introduced for the TDS representation, and in particular on the detection of coordination constructions.

Chapter 6

Conclusions

It was six men of Indostan
To learning much inclined,
Who went to see the Elephant
(Though all of them were blind),
That each by observation
Might satisfy his mind.

The First approached the Elephant,
And happening to fall
Against his broad and sturdy side,
At once began to bawl:
”God bless me!-but the Elephant
Is very like a wall!”

The Second, feeling of the tusk,
Cried: ”Ho!-what have we here
So very round and smooth and sharp?
To me’t is mighty clear
This wonder of an Elephant
Is very like a spear!”

The Third approached the animal,
And happening to take
The squirming trunk within his hands,
Thus boldly up and spake:
”I see,” quoth he, ”the Elephant
Is very like a snake!”

[...]

And so these men of Indostan
Disputed loud and long,
Each in his own opinion
Exceeding stiff and strong,
Though each was partly in the right,
And all were in the wrong!

John Godfrey Saxe

At the beginning of this thesis we have stressed how current computational linguistic research seems to be drifting away from linguistic theory. One of the main motivations behind our work was therefore to contribute to reversing this tendency, by investigating several computational models of syntax while keeping in mind the original linguistic view on the studied phenomena.

One of the main points of divergence between the two fields is the difficulty to translate a number of syntactic models over various tree structure representations into computational models that can be learned and tested empirically. In chapter 2 we have therefore presented a general methodology for simplifying the task of formalizing theory-independent probabilistic generative models for various tree structures. One of the main objectives behind this work was to stimulate more diversification in computational linguistics concerning the syntactic representation underlying the models and the linguistic theories for generating them. The proposed methodology is in fact able to generalize over a number of tree-based theories by providing i) a unique description of the sequence of events characterizing the generation of the syntactic analyses of a given sentence according to various models, and ii) a way to map a symbolic grammar into a probabilistic model.

We have applied this methodology to three distinct syntactic representations: simple phrase-structure (PS), dependency-structure (DS), and Tesnière dependency-structure (TDS). After comparing PS and DS on theoretical grounds, we have shown how they can be seen as two complementary representations for syntax. We have therefore formalized TDS as an intermediate representation between PS and DS which is better at representing linguistic phenomena such as coordination. We have experimented on the three schemes empirically, through the implementation of computational models for generating syntactic analyses according to the three representations.

For phrase-structure trees, in chapter 3 we have proposed a novel Data-Oriented Parsing formalism (Double-DOP) based on an explicit representation of arbitrarily large syntactic fragments. As the number of all possible constructions which can be extracted from a large treebank is extremely large, we need to resort to a restricted yet representative subset of fragments. We therefore propose a linguistically motivated methodology to automatically identify those fragments for which there is evidence about their reusability. We achieved this by means of a special-purpose application (Fragment-Seeker), based on an efficient tree-kernel algorithm.

Both Fragment-Seeker and the Double-DOP parser are made available to the linguistic community at large, and we hope they will prove useful in linguistic research as well as other NLP applications. In particular, as the identification of syntactic constructions has always represented one of the biggest challenges in linguistic theory (Fillmore et al., 1988; Goldberg, 1995; Kay and Fillmore, 1997), we believe that further investigation into reusable fragments could provide fruitful insights for studying linguistic phenomena pertaining to the identification of syntactic constructions such as subcategorization, idiomatic expressions, and multiword expressions. Regarding NLP applications, we expect that Fragment-Seeker and Double-DOP could be used as a component for competitive systems for solving linguistic tasks such as machine translation, semantic role labeling, question-answering and speech recognition systems.

For DS trees, in chapter 4 we have compared a number of probabilistic generative bilexical models initially proposed by Eisner (1996a,b) and experimented with some of their variations. Although the different models greatly differ in their linguistic hypotheses on the types of rules and features that are considered in the respective grammars, we were able to efficiently compare them by resorting to the generalized representation of the event space as presented in chapter 2, and by relying on a reranking methodology. Reranking can be seen as a useful approach for easily evaluating a number of generative models of syntax and approximating the behavior of a full-fledged probabilistic parser which would need to be specifically implemented and optimized for the chosen model. In our implementation, in fact, the reranking approach can be seen as a parser simulator which mimic all the choices a model would perform to obtain a certain analysis of a given sentence. This approach could also be employed as a linguistic tool to better understand why, under a specific probabilistic model, certain analyses are preferred over others.

Looking at the last 100 years or so, one of the main aspects of linguistic research has been the search for more adequate syntactic representations. In contrast, in computational linguistics, the difficulty of building large manually annotated treebanks has often represented a major obstacle for empirically investigating a wide range of syntactic representations. In chapter 5 we contributed to bridging this gap, by introducing a novel syntactic scheme based on the formalization of the original work on dependency syntax of Lucien Tesnière (1959), and therefore called Tesnière Dependency-Structure (TDS). In our opinion, this constitutes a very promising formalism for linguistic representation although only some aspects of it have been used in computational linguistics: modern DS retains only the main idea proposed by Tesnière, namely the relation of dependency between words, while other operations and features of the original theory are discarded or not overtly represented. More specifically, TDS can be seen as an intermediate representation between phrase-structure and dependency-structure, as it uses constituencies to groups adjacent words (chunks) and to represent coordination constructions, while it adopts dependency relations to link together the syntactic units (chunks and coordination constructions).

In order to investigate such a representation empirically, we have implemented an automatic procedure for converting the English WSJ treebank into TDS notation, and used the converted treebank to test a new computational, probabilistic model for generating TDS analyses of novel sentences. This model was implemented using a reranking framework similar to the one used for testing DS models.

Finally, in order to evaluate this model we have defined three separate metrics specific to the linguistic features represented in TDS, namely, chunks, dependencies, and coordination. As currently used evaluation metrics for parsing have often raised skepticism about the extent by which they are able to quantify the correctness of the derived analyses, we hope that our attempt to define a range of linguistically motivated evaluation metrics may be useful for the parsing community.

After the statistical revolution of the '90s in computational linguistics, most of the computational models of syntax have been focusing on phrase-structure and only more recently on dependency-structure. Deep syntactic representations (e.g., HPSG, LFG) have had more marginal roles in the parsing community because of the lack of significant annotated resources and the difficulty of deriving efficient models. By proposing a novel TDS scheme we have attempted to push forward the requirements probabilistic parsers should meet, by compromising between linguistically adequate representations and efficient generative models for parsing.

Appendix A

Phrase-Structure Models

A.1 Models parameters

In table A.1 we report the parameters used to obtain the parsing results in table 3.3. These were chosen in order to maximize results on the development section of the corresponding treebank (for parsing the Brown corpus, for which there is no standard development section, we have used the same parameters used for the WSJ). All models use the same binarization (see §3.7) with H=1, P=1.

Unknown Threshold words occurring less than this threshold are substitute with word-features.

Unknown Word Model the 5 different models as in Berkeley code to specify how unknown words are converted into word-features (5 is the most specific to English, 1 is the most generic).

σ Open-class threshold used for smoothing (see §3.7). A PoS-tag is an open class if it rewrites to at least σ different words in the training corpus. A word is an open class word if it has been seen only with open-class PoS-tags.

ϵ Low frequency count assigned to open-class $\langle$ word, PoS-tag $\rangle$ pairs not encountered in the training corpus.

λ Threshold to maximize F1 score, as illustrated in §3.6.3.

Treebank	Parsing Model	Unknown Threshold	Unknown Word Model	σ	ϵ	λ
WSJ	PCFG	4	1	50	0.01	-
	Double-DOP	4	5	50	0.01	1.10
Brown	PCFG	1	1	50	0.01	-
	Double-DOP	1	5	50	0.01	1.10
Negra	PCFG	9	4	50	0.01	-
	Double-DOP	9	4	50	0.01	0.95
FTB	PCFG	1	4	50	0.01	-
	Double-DOP	1	4	50	0.01	1.45
CTB 3.0	PCFG	1	4	50	0.01	-
	Double-DOP	1	4	50	0.01	1.05
HTB	PCFG	1	1	100	0.01	-
	Double-DOP	1	1	100	0.01	1.05

Table A.1: Parameters used to obtained the results in table 3.3.

A.2 Evaluation procedure

EvalB discards traces (and semantic tags) but does not remove redundant rules that result from the elimination of traces (and semantic tags). See for instance the example structure in figure A.1. For obtaining the cleaned version of the file i apply the following steps:

- prune traces subtrees (-NONE-)
- remove numbers in labels (e.g., NP-2 or NP=2)
- remove semantic tags (e.g., NP-SBJ)
- remove redundant rules (e.g., NP $\longrightarrow$ NP)

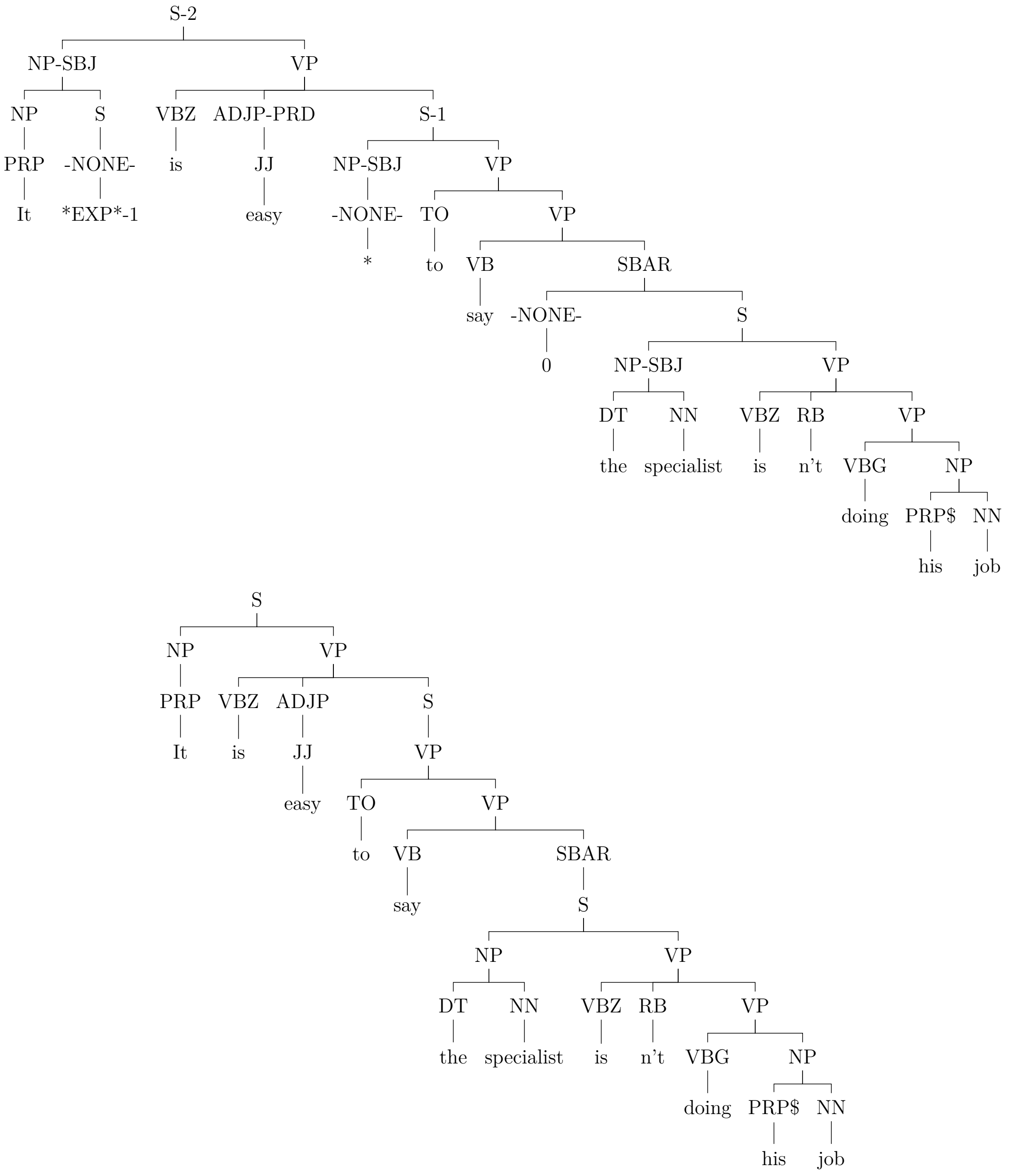


Figure A.1: (sec 23, 11th structure)

A.3 Comparing Double-DOP and Berkeley parser

Category label	% in gold	F1 Berkeley	F1 Double-DOP
NP	41.42	**91.4**	89.5
VP	20.46	**90.6**	88.6
S	13.38	**90.7**	87.6
PP	12.82	**85.5**	84.1
SBAR	3.47	**86.0**	82.1
ADVP	3.36	**82.4**	81.0
ADJP	2.32	**68.0**	67.3
QP	0.98	82.8	**84.6**
WHNP	0.88	**94.5**	92.0
WHADVP	0.33	**92.8**	91.9
PRN	0.32	**83.0**	77.9
NX	0.29	**9.50**	7.70
SINV	0.28	**90.3**	88.1
SQ	0.14	**82.1**	79.3
FRAG	0.10	26.4	**34.3**
SBARQ	0.09	84.2	**88.2**
X	0.06	72.0	**83.3**
NAC	0.06	54.6	**88.0**
WHPP	0.06	**91.7**	44.4
CONJP	0.04	55.6	**66.7**
LST	0.03	**61.5**	33.3
UCP	0.03	30.8	**50.0**
INTJ	0.02	44.4	**57.1**

Table A.2: Comparison of the performance (per-category F1 score) on the development set of the WSJ (section 24 ≤ 40) between the Berkeley parser and our Double-DOP model trained on a non-refined treebank (RFE, MCP with $\lambda = 1.00$, H=1, P=1 and lexical smoothing).

Appendix B

Dependency-Structure Models

B.1 DS to PS

The way to transform DS into PS is less discussed in the literature. The procedure illustrated here is recursive, and takes into consideration a DS rooted in node w. Three general cases can be encountered:

1. w is a terminal node (it has no dependents): in this case the resulting PS has a single terminal node W.

2. w has a single dependent. The corresponding PS has two daughter nodes: the PS obtained by transforming the dependent, and a single-node PS containing the governing word w. The two daughters will keep the same linear order as in the original DS.

3. w has k dependents $d_1, d_2, \ldots, d_k$ $(k > 1)$. In this last case w is mapped into a single terminal PS node W, and each dependent d_j is transformed into a PS D_j (recursive step). Given that w occurs after the i^{th} dependent we can represent $k+1$ trees in a single sequence of nodes as $D_1, \ldots, D_i, W, D_{i+1}, \ldots, D_k$. At this point, we need to choose how to group these $k+1$ nodes into a series of hierarchical PS trees $P_1, P_2, \ldots, P_m$ $(1 \le m \le k)$, such that:

   - P_j must form a contiguous span (for every $1 \le j \le m$).
   - P_1 must contain as daughter W and at least one other node.[1] In order to form a contiguous span this is either D_i or D_{i+1} (e.g., D_1, W, D_3 is not allowed).
   - P_{j+1} must contain P_j as daughter, together with at least another node from the initial sequence $D_1, \ldots, D_k$ (for every $1 \le j < m$). The set of selected daughters should yield a contiguous span of the sentence.

[1] P_1 could contain W alone if we allow unary chains.

- P_m must contain directly (as daughter) or indirectly (daughters of daughters) all $k + 1$ nodes in the initial sequence.

The resulting PS is rooted in P_m.

Case 3 of this procedure is the only one which requires a choice to be made. In fact, as we can see in figure 4.3, DS trees A, B, F, G are mapped into a single PS because they do not contain any node with more than one dependent. Tree D, instead, has a node with 2 dependents. Following case 3, we have three choices:

1. ($m = 1$) group the governor (2) and all the dependents (1,3) in a unique constituent, resulting in PS tree β;
2. ($m = 2$) group first the governor (2) with the left dependent (1) and the resulting constituent to the remaining dependent (3), resulting in PS tree γ;
3. ($m = 2$) group first the governor (2) with the right dependent (3) and the resulting constituent to the remaining dependent (1), resulting in PS tree α.

The general idea behind the study conducted by Hays (1960) is that PS and DS are complementary in nature. PS well represents constituents but leaves the notion of heads underspecified, while DS is based on governing-dependency relations but does not specify how to group multiple dependents of the same governor.

Gaifman (1965) claims that a given DS has a “naturally corresponding” PS, such that every DS maps to a single PS, but not vice versa (a single PS may map to multiple DS trees). This is a somehow misleading claim since it is based on an ad-hoc definition of equivalence between PS and DS. According to this definition, given a DS there exists a unique equivalent PS tree obtained by imposing $m = 1$ in the procedure above: the PS should contain the same number of constituents as there are governors in the DS, and each constituent should expand to a set of words including the associated governor together with all the words that depend directly or indirectly from it. This mapping is illustrated in figure 4.3 with dashed lines. According to this mapping we still have some PS mapping to multiple DSs (remember that DSs are always more in number), but in this case we have more restrictions than before: the head of each constituent must be chosen among the daughter node of minimum *depth*, where the depth of a node is defined as the length of the longest path connecting the node to one of the terminal node under its span.

B.2 Smoothing details

In the final model illustrated in equation 4.4, we have a complex conditional probability:

$$P(dist(H,D), term(D), word(D), tag(D)|H, S, G, dir) \tag{B.1}$$

For simplicity we will rewrite it as $P(A,B,C,D|E,F,G,H)$. We decompose the probability in 4 parts: $P(D|E,F,G,H) \times P(C|D,E,F,G,H) \times P(B|C,D,E,F,G,H) \times P(A|B,C,D,E,F,G,H)$.

As explained in equation 4.1, each of those probabilities are estimated from the training corpus with the relative frequency estimate. For instance the first term is obtained in the following way:

$$P(D|E,F,G,H) = \frac{count(D,E,F,G,H)}{count(E,F,G,H)} \tag{B.2}$$

Since this equation involves a big number of terms, it is likely that many events (and conditioning contexts) encountered during the re-ranking are never observed in the training treebank. We therefore need to resort on a series of backoff probabilities for smoothing the count on full contexts with those obtained from a coarser representation of the contexts. This is obtained by *deleted interpolation*, i.e., by deleting some elements from the context at each back-off level (see also Bikel, 2004a).

For instance in equation 4.4 the conditioning context of the first term (H, S, G, dir) is reduced in 4 incremental steps:

$$\begin{array}{|l|}
\hline
wt(H), wt(S), wt(G), dir \\
\hline
wt(H), wt(S), t(G), dir \\
\hline
\begin{cases} wt(H), t(S), t(G), dir \\ t(H), wt(S), t(G), dir \end{cases} \\
\hline
t(H), t(S), t(G), dir \\
\hline
\end{array} \tag{B.3}$$

In the first row the full context is specified (recall that $wt(N)$ stands for the string incorporating both the pos-tag and the word of N). In the second step the word of G is ignored (only its pos-tag is taken under consideration). In the third step either the word of S is ignored or the one of H. In the last step all words are ignored.

We can compute the estimates from the various backoff levels e_1, e_2, e_3, e_4 separately as shown in equation B.2. For instance the first one is obtained as:

$$\begin{aligned} e_1 &= P(tag(D)|wt(H), wt(S), wt(G), dir) \\ &= \frac{count(tag(D), wt(H), wt(S), wt(G), dir)}{count(wt(H), wt(S), wt(G), dir)} \end{aligned} \tag{B.4}$$

For the third level of backoff the contributions of the two equally reduced contexts are summed up as follows:

$$e_3 = \frac{count(tag(D), wt(H), t(S), t(G), dir) + count(tag(D), t(H), wt(S), t(G), dir)}{count(wt(H), t(S), t(G), dir) + count(t(H), wt(S), t(G), dir)} \tag{B.5}$$

Following Eisner (1996a), the estimates calculated for all backoff levels except the last one (in this case $i < 4$) are interpolated in a recursive fashion:

$$\tilde{e}_i = \frac{count(tag(D), wt(H), wt(S), wt(G), dir) + 3 \cdot \tilde{e}_{i+1}}{count(wt(H), wt(S), wt(G), dir) + 3} \tag{B.6}$$

The last backoff level $\tilde{e}_4$ is obtained as:

$$\tilde{e}_4 = \frac{count(tag(D), t(H), t(S), t(G), dir) + 0.005}{count(t(H), t(S), t(G), dir) + 0.5} \tag{B.7}$$

The low count added in equation B.7 is to guarantee that even if none of the backoff levels were observed, the resulting probability is extremely low but non-zero. The interpolation illustrated in equation B.6 guarantees higher contribution for more specific contexts, i.e., if a specific context was frequently observed in the training corpus it will largely override the other coarser estimations.

Appendix C

TDS model

C.1 Head annotation

We report below the head annotation table used to perform the conversion of the Penn WSJ Treebank into TDS representation (see section 5.4.2). For a given CFG rule in the treebank, the table describes which daughter node (in the right hand-side of the rule) is marked as head. Each parent node in the table (left hand-side of the rule) is mapped to a list of sub-rules (rows in the table). If the first sub-rule does not apply the second is considered, if not the third, and so on. In each sub-rule the daughter nodes in the rule are read according to the specified start direction in the second column (left/right). The third column refers to the priority: if it is set to D (daughters), all daughters are checked against the first element in the list, if none applies, against the second element the list, the third, and so on; if it is set to L (list) all elements in the list (from left to right) are checked against the first daughter, if it does not match, against the second daughter, the third, and so on. The special symbol * matches any label, and is inserted at the end of the last sub-rule to ensure that there is always an head-daughter for each rule.

Parent	Start	Priority	List
ADJP	left	D	NNS, QP, NN, NML, $
	right	D	JJ
	left	D	ADVP, VBN, VBG, AUX, AUXG, ADJP, JJR, JJS, JJP, FW, RBR, RBS, RB, NP, SBAR, DT, *
ADVP	right	D	RB, RBR, RBS, FW, ADVP, CD, JJR, JJ, JJP, NP, NML, JJS, NN, TO, IN, *
CONJP	right	D	RB, IN, *
FRAG	left	D	NP, NML, VP, ADJP, PP, *

INTJ	left	D	*
LST	right	D	LS, :, *
NAC	left	L	NN, NNS, NNP, NNPS, NP, NAC, NML, EX, $, CD, QP, PRP, VBG, AUX, AUXG, JJ, JJS, JJR, JJP, ADJP, FW, *
NP	right	L	NN, NNS, NX, NNP, NNPS, NML
	left	L	NP
	right	L	$ ADJP, PRN
	right	L	CD
	right	L	JJR, JJ, JJS, JJP, RB, QP
	right	D	*
NML	right	L	NN, NNP, NNPS, NNS, NX, NML, JJR
	left	L	NP
	right	L	$, ADJP, PRN
	right	L	CD
	right	L	JJ, JJS, JJP, RB, QP
	right	D	*
NX	right	D	NP, NX
	right	L	NNPS, NNS, NNP, NN, NML, JJR, JJP, *
JJP	right	L	JJ, JJR, JJS, JJP, VBG, *
PP	right	D	VBG, VBN, AUX, AUXG, RP, FW, NP, SBAR, S, IN, TO, *
PRN	left	D	VP, NP, PP, *
PRT	right	D	RP, *
QP	left	D	$, NNS, NN, NML, JJ, RB, CD, NCD, QP, JJR, JJS, JJP, IN, DT, *
RRC	right	D	VP, NP, ADVP, ADJP, PP, *
S	left	D	VP, S, SBAR, ADJP, UCP, NP, TO, IN, *
SBAR	left	D	NN, NML
	left	D	S, SQ, SINV, SBAR, FRAG, WHNP, WHPP, WHADVP, WHADJP, IN, DT, *
SBARQ	left	D	SQ, S, SINV, SBARQ, FRAG, *
SINV	left	D	VBZ, VBD, VBP, VB, AUX, AUXG, VP, S, SINV, ADJP, NP, NML, MD, *
SQ	left	D	VBZ, VBD, VBP, VB, AUX, AUXG, VP, SQ, MD, *
UCP	right	D	*
VP	left	D	VBD, VBN, VBZ, VB, VBG, VBP, AUX, AUXG, VP, ADJP, NN, NNS, NP, NML, MD, TO, JJ, JJP, *
WHADJP	left	D	WRB, JJ, JJP, ADJP, *

WHADVP	right	D	WRB, *
WHNP	left	D	NNS, NN, WDT, WP, WP$, WHADJP, WHPP, WHNP, *
WHPP	right	D	IN, TO, FW, *
X	left	D	NN, NNP, NNPS, NNS, NX, NML, JJR, JJP, *
TOP	left	D	*

C.2 Coordination

We report the result of a manual comparison between the TDS treebank and the CCG-bank (Hockenmaier and Steedman, 2007), of all coordinated structures detected in section 0 of the Penn WSJ treebank for which the two analyses differ (168 sentences). In summary there are 135 mistakes in the CCG-bank, 29 in the TDS-bank, and 48 cases which are uncertain. The CCG-bank does not annotate all sentences of the WSJ. For section 0 the following sentence numbers are missing: 114, 269, 323, 465, 1052, 1251, 1295, 1865. The indices below refer only to the sentences in the CCG bank (e.g., 114 refers to WSJ sentence number 115).

CCG-bank not detected coordination (total 50)
29 (cotton, acetate), 66 (sales, marketing), 67 (sales, service, parts, operations), 69 (sales, marketing), 131 (electronics, appliances), 133 (current, former), 168 (Securities, Exchange), 195 (research, development), 218 (movie, book), 225 (patent, copyright), 304 (Japanese, other), 306 ($, tenth), 337 (Political, currency), 352 (analysts, managers), 392 (goods, those), 454 (humble, uncomplaining, obedient), 455 (permitted, welcomed), 516 (exciting, eclectic), 654 (economic, political), 655 (economic, foreign), 705 (Scoring High, Learning Materials), 726 (run-down, inner city), 834 (pie, bar), 853 (CAT, CTBS), 848 (insurance, financial), 823 (Connecticut, Massachusetts), 947 (judges, Judge) [see figure C.5], 997 (business, government), 1042 (French, German), 1142 (first, second), 1204 (lap, shoulder), 1233 (metals, materials), 1371 (parts, controls, electronics), 1410 (government, business), 1424 (U.S., Japanese), 1425 (savings, investment), 1437 (MITI, Department), 1487 (president, officer), 1574 (Securities, Exchange), 1592 (finance, telecommunications), 1602 (Energy, Commerce), 1605 (peculiar, unintelligible), 1701 (profits, flow), 1730 (software, service), 1732 (Canadian, Kingdom), 1749 (hundreds, thousands), 1790 (change, changing) [not], 1820 (escrow, record-keeping), 1879 (U.S., London), 1912 (navigation, targeting).

CCG-bank wrongly detected conjuncts (total 19)
131 (63, chairman), 200 (47, president), 200 (37, president), 200 (40, president), 200 (45, president), 206 (Hatch, 59), 208 (Carney, 45), 308 (March, 1990), 502 (Nov., 1999), 504 (Nov., 1990), 952 (Ramirez, 44), 992 (Bromwich, 35), 1215 (Milne, 65), 1233 (Butler, 64), 1333 (Nov., 1992), 1334 (Nov., 1999), 1156 (York, agency), 1843 (Jr., $).

CCG-bank mismatched conjuncts (wrong / correct) (total 13)
91 (Trade, Ministry) / (Trade, Industry), 308 (aerospace, products) / (steel, aerospace, products), 648 (business, program) / (business, research), 651 (Public, Affairs) / (Public, Internatonal), 816 (software, worksheets) / (booklets, software, worksheets), 962 (TV, industry) / (TV, movie), 995 (manufacturing, space) / (office, manufacturing, warehousing) [2 mistakes], 1059 (research, facility) / (research, development), 1547 (Securities, Commission) / (Securities, Exchange), 1612 (morning, service) / (morning, evening), 1676 (Growth, Fund) / (Growth, Income), 1798 (Mr., Bush) / (Mr., Mrs.).

CCG-bank missing apposition (total 54)
65 (maker, Corp), 76 (pianist-comedian, Borge), 135 (Judge, Curry), 186 (Three, (Ltd, Corp., Ltd.)), 213 (countries, (China, Thailand, India, Brazil, Mexico)), 238 (negotiator, Carballo), 287 (founder, shareholder), 308 (March, period), 310 (giant, S.p.A.), 319 (craze, rash), 321 (President, Aquino), 425 (newcomer, (milk, powder)), 454 (star, Hara), 463 (home, dormitory), 482 (agency, WAFA), 490 (agency, PAP), 502 (million, million), 516 (Composer, Marder), 536 (maker, Chabrol), 593 (Gov., Wilder), 609 (Rep., Florio), 609 (Rep., Courter), 656 (members, (Thailand, Malaysia, Singapore, Indonesia, Philippines, Brunei)) [--], 719 (physicist, Townes), 719 (actress, Woodward), 726 (groups, (elite, blacks)), 834 (subskills, (symmetry, measurement, graphs)) 931 (attorney, Lefcourt), 947 (Judge, Ramirez), 959 (Cartoonist, Trudeau), 985 (Chairman, Sherwin), 1017 ((semiconductors, supercomputers), products) [--], 1033 (President, Backe), 1052 (million, million), 1118 (entrepreneur, Poore), 1125 ((foam, polyproplene, film), items) [--], 1156 (Mather, agency), 1244 (superpremiums, wines) [--], 1245 (Bordeaux, Burgundies, Champagnes, wines), 1245 (classics, (Bordeaux, Burgundies, Champagnes, wines)) [--], 1274 (Schaefer, one), 1423 (Sen., Bentsen), 1448 (products, (one, another)) [--], 1576 (points, equivalent) [--], 1602 (Fifteen, (Dingell, chairman)), 1635 (novelist, Sayers), 1643 (Rev., Hummerstone), 1607 (is, stands) [separator :], 1624 (rounds, scale), 1657 (problem, lack) [separator :], 1709 (slowdowns, environment) [--], 1758 (dissident, Lizhi), 1793 (Prof, Klein).

TDS-bank not detected coordination (total 4)
454 (humble, uncomplaining, obedient), 455 (permitted, welcomed), 516 (exciting, eclectic), 726 (run-down, inner city).

TDS-bank mismatched conjuncts [wrong/correct] (total 9)
67 (sales, service, parts, marketing) / (sales, service, parts, operations), 106 (spent, $) / ($, $) [verb ellipsis], 327 (are, will, listed) / (are, will) [verb ellipsis], 947 (ease, Judge) / (judges, Judge), 1150 (valued, $) / ($, $) [verb ellipsis], 1244 (of, with) / (limited, of, with), 1245 growths / Bordeaux, 1315 buy / warrants, 1656 (church, life, experience) / (life, experience), 1906 (pay, days) / (for, for) [verb ellipsis].

TDS-bank missing appositions (total 8)
65 (maker, Corp), 308 (March, period), 502 (million, million), 556 (ad, presence) [see figure C.5], 834 (subskills, (symmetry, measurement, graphs)), 1052 (million, million), 1602 (Fifteen, (Dingell, chairman)), 1793 (Prof, Klein).

TDS-bank wrong appositions (total 7)
98 (rival, magazine), 460 (home, dormitory), 996 (rival, Corp.), 1239 (Leap, Cellars), 1315 ($, amount), 1507 (loan, $), 1577 (1:30, time).

Uncertain [detected in CCG-bank not in TDS-bank] (total 31)
58 (Wickliffe, Ohio), 115 (Westborough, Mass.), 116 (Haven, Conn.), (Hartford, Conn), 117 (Manchester, N.H.), 144 (Rockford, Ill.), 323 (Northampton, Mass), 409 (Stamford, Conn), 947 (Sacramento, Calif), 993 (Albany, Ga.), 1041 (Heidelberg, Germany), 1066 (Lake, N.J), 1106 (Providence, R.I.), (Raleigh, N.C.), (Louisville, Ky.), 1198 (R., Mo), 1212 (Elmhurst, Ill), 1274 (Skokie, Ill.), 1274 (Pratt, director), (Skokie, Ill.), 1320 (Shelby, Ohio), 1345 (Clive, Iowa), 1400 (Westport, Conn.), 1403 (Brunswick, N.J.), 1607 (ASLACTON, England), 1729 (Hills, Calif), 1801 (Birmingham, Ala), 1839 (Killeen, Texas), (Shores, Fla.), (Heights, Ill.), (Heights, Ill.), (Boulder, Colo.), (Horsham, Pa.), 1840 (Colonsville, Miss.), (Valrico, Fla.), (Canada, Calif.), 1843 (Longwood, Fla.), (Bronx, N.Y.), (Glenham, N.Y.), (Park, N.J.), (Park, Minn.), (Nesconset, N.Y.), 1844 (Hermitage, Pa.), (Louis, Mo.), (Gaithersburg, Md.), Ridgefield, N.J.), (Aloha, Ore.), (Estates, N.Y.), 1845 (Russel, Dahl), (Hills, Calif.), (Glendale, Calif.), (Valley, Calif.), 1848 (Jackson, Miss.), (Springs, Colo.), (Rouge, La.), (Midvale, Utah), (Creek, Fla.), (Aurora, Colo.), (fine, suspension), (Providence, N.J.), (Bridgeville, Pa.), (Aurora, Colo.), (Vegas, Nev.), (City, Nev.), 1898 (D., Mont), 1901 (R., N.J).

Uncertain [detected in TDS-bank not in CCG-bank] (total 17)
79 (could, welcomed) [so], 167 (is, is) [so], 302 (are, (can, forces)) [--], 455 ('s, (return, expands, are, must)) [:], 681 (surrendered, protest) [but], 949 (reason, refusal) [:], 1244 (quality, perceived) [or], 1451 (view, went) [so], 1519 (reason, margins) [:], 1557 (spend, do) [--], 1567 (Do, needs) [--], 1608 (is, stands) [:], 1624 (rounds, scale) [--], 1672 (breathe, warn) [or], 1685 (say, has) [:], 1800 (we, blacks) [apposition], 1880 (begins, prices) [but].

C.3 Smoothing in the TDS model

In the three conditional probability equations described in §5.5.1 and reported again below, we have adopted a smoothing techniques based on deleted interpolation (Eisner, 1996a; Collins, 1999), similarly to the smoothing implemented for the DS model (see §4.6.3 and Appendix B.2).

$$P(S) = P_{BGM}(S) \cdot P_{BEM}(S) \cdot P_{WFM}(S) \tag{C.1}$$

$$P_{BGM}(S) = \prod_{B \in depBlocks(S)} P(B|parent(B), direction(B), leftSibling(B)) \tag{C.2}$$

$$P_{BEM}(S) = \prod_{B \in blocks(S)} P(elements(B)|derivedCat(B)) \tag{C.3}$$

$$P_{WFM}(S) = \prod_{B \in stdBlocks(S)} P(cw(B)|cw(parent(B)), cats(B), fw(B), context(B)) \tag{C.4}$$

The first two models (equations C.2 and C.3) are smoothed with a simple additional level of back-off which is a constant value (10^{-6}) to make the overall probability small but not zero, for unknown events. Recall that in both models all elements are encoded with only partial information, viz. categories and functional words, but no lexical information for the content words. This justifies the choice of avoiding a more refined back-off estimation.

The third model is implemented with three levels of back-off: the last is set to the same constant value (10^{-6}), the first encodes the dependency event using both pos-tags and lexical information of the governor and the dependent word, while the second specifies only pos-tags.

The different back-off levels for each probability, are interpolated with confidence weights derived from the training corpus (except for the last level which remains the constant 10^{-6}). This is differently from the DS model, where the interpolating parameters are instead constant. Specifically, each back-off level obtains a confidence weight which decreases with the increase of the *diversity of the context* $\theta(C_i)$, which is the number of separate events occurring with the context C_i (see also Bikel, 2004a). More formally if $f(C_i)$ is the frequency of the conditioning context of the current event, the weight is obtained as $f(C_i)/(f(C_i)\cdot\mu\cdot\theta(C_i))$. In our model we have chosen μ to be 5 for the first model, and 50 for the second and the third.

C.4 Examples of TDS trees

In this section we include some examples of TDS trees with coordination construction, selected from section 0 of the WSJ TDS-bank.

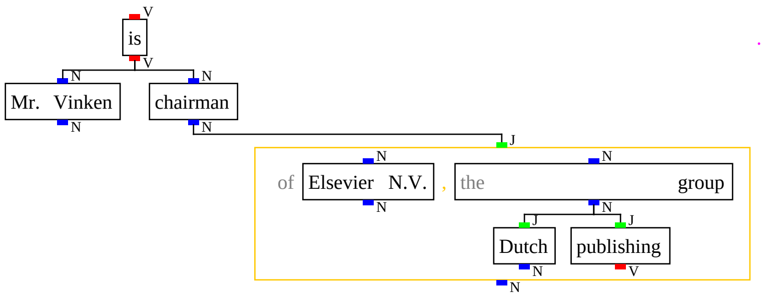


Figure C.1: TDS tree #2 (apposition construction).

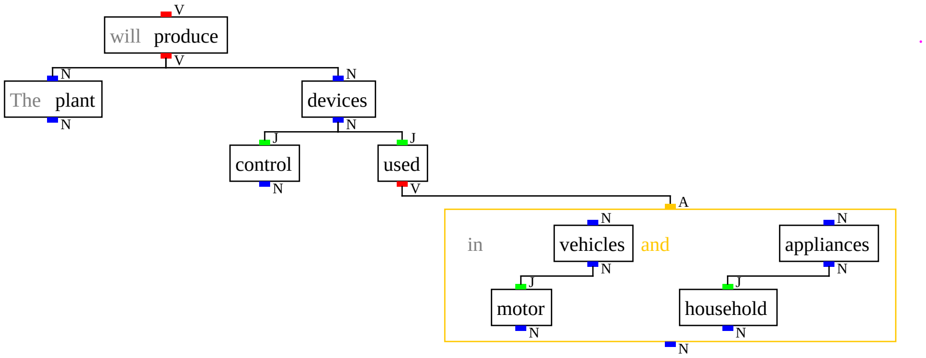


Figure C.2: TDS tree #166 (coordination construction with conjuncts modification).

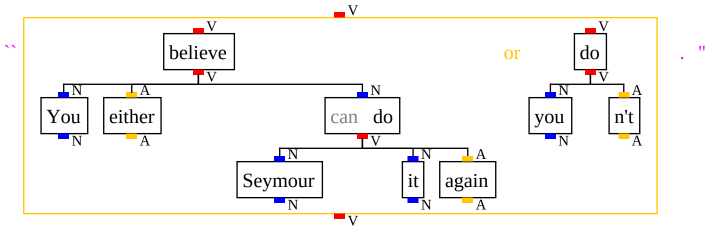


Figure C.3: TDS tree #182 (‘either’ - ‘or’ construction).

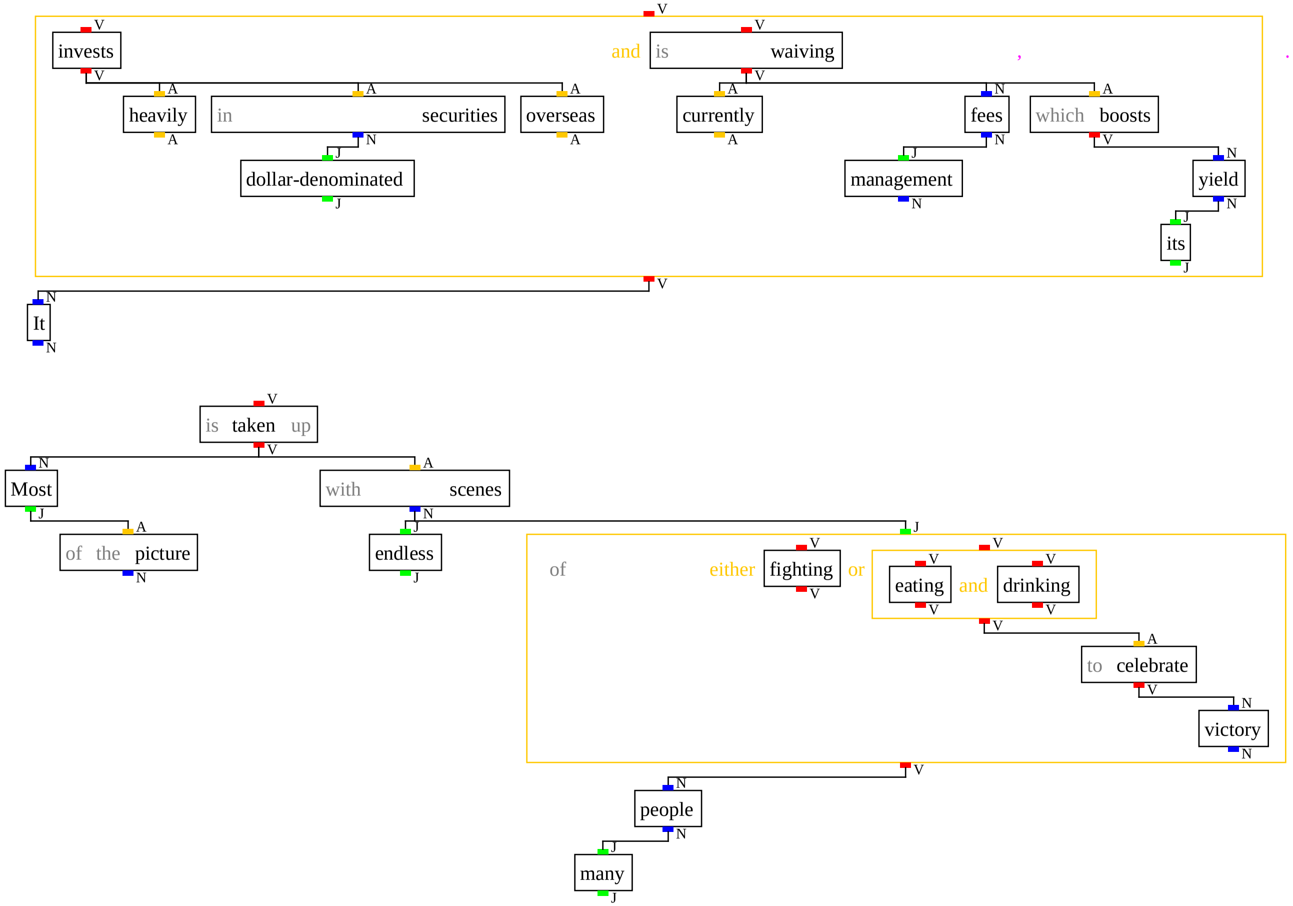


Figure C.4: Above: TDS tree #48 (compound verbs).
Below: TDS tree #552 (hierarchical coordination construction).

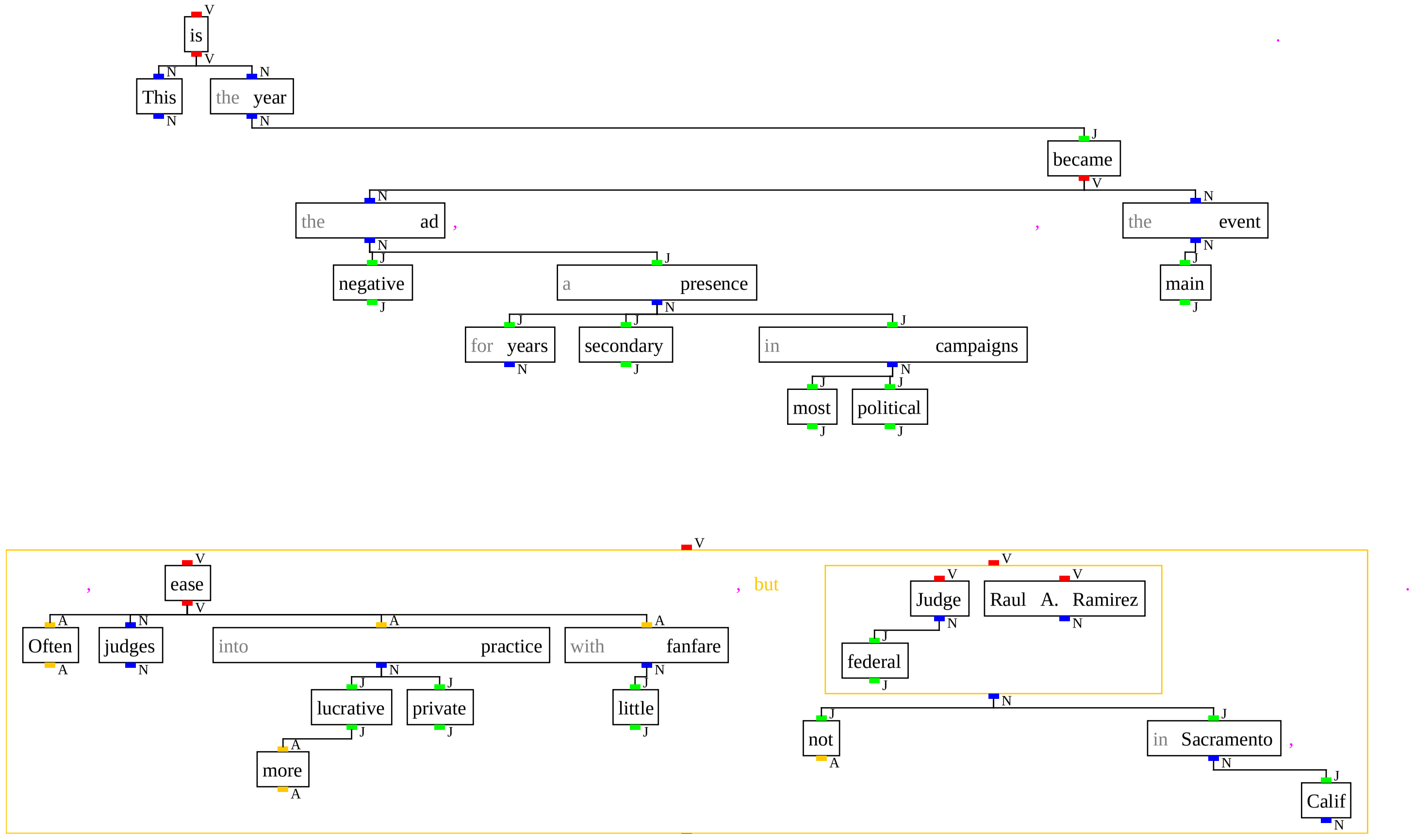


Figure C.5: Above: TDS tree #560 (not detected apposition between ‘ad’ and ‘presence’). Below: TDS tree #951 (Mismatched coordination, should have been between ‘judges’ and ‘Judge’).

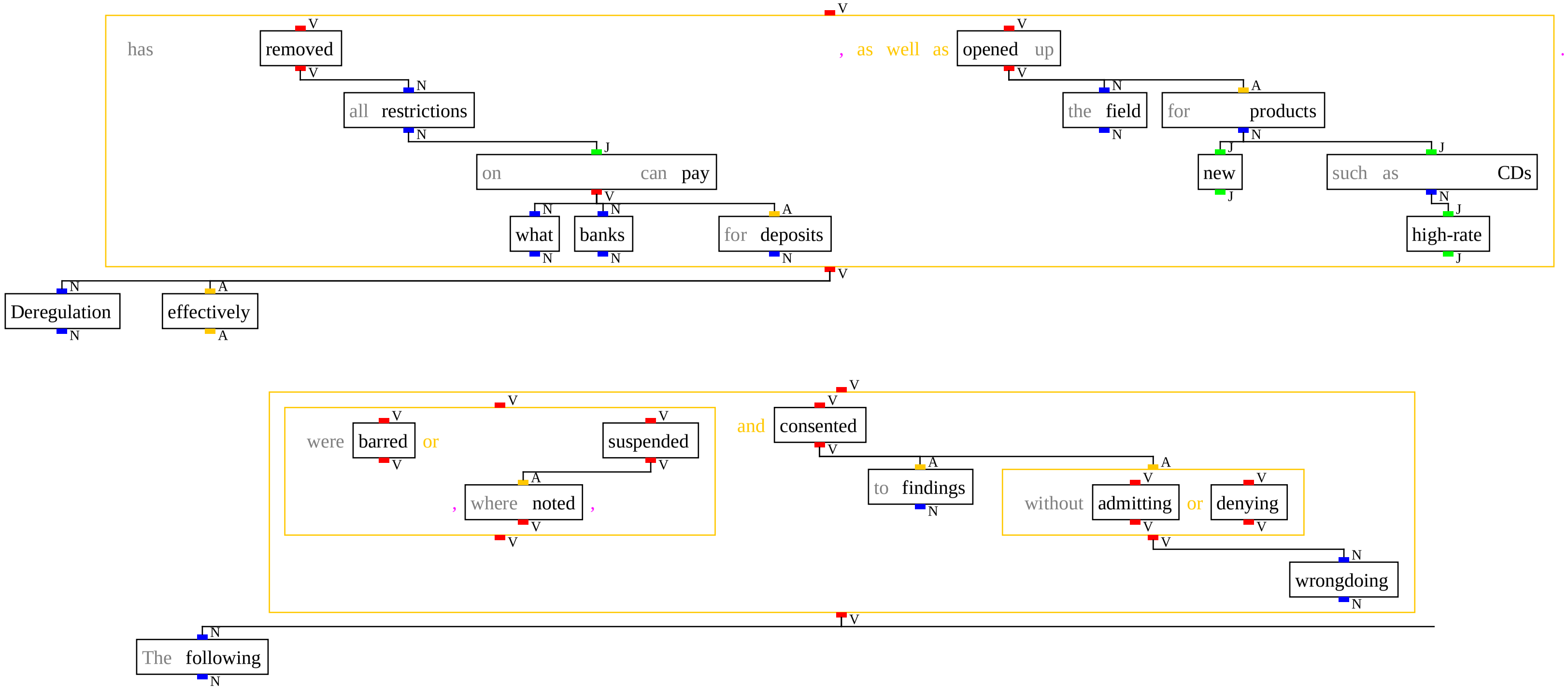


Figure C.6: Above: TDS tree #1531 (Compound verbs in coordination when adverb is in between auxiliary and first verb).
Below: part of TDS tree #1855.

Bibliography

Abeillé, Anne. *Treebanks: Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*. Kluwer Academic Publishers, 2003.

Abeillé, Anne, Lionel Clément, and François Toussenel. *Building a Treebank for French*, pages 165–188. Volume 20 of *Text, Speech and Language Technology* Abeillé (2003), 2003.

Abend, Omri and Ari Rappoport. Fully unsupervised core-adjunct argument classification. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 226–236, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

Abney, Steven. Statistical Methods and Linguistics. In Klavans, Judith and Philip Resnik, editors, *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, pages 1–26. The MIT Press, Cambridge, Massachusetts, 1996.

Aho, A. V. and S. C. Johnson. LR Parsing. *ACM Comput. Surv.*, 6(2):99–124, 1974.

Ajdukiewicz, K. Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27, 1935.

Anderson, John. Dependency and Grammatical Functions. *Foundations of Language*, 7(1):pp. 30–37, 1971.

Arnon, Inbal. *Starting big — The role of multi-word phrases in language learning and use.* PhD thesis, Stanford University, 2009.

Arnon, Inbal and Neal Snider. More than words: Frequency effects for multi-word phrases. *Journal of Memory and Language*, 62(1):67–82, 2010.

Arun, Abhishek and Frank Keller. Lexicalization in crosslinguistic probabilistic parsing: the case of French. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 306–313, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

Bachrach, Asaf. *Imaging neural correlates of syntactic complexity in a naturalistic context*. PhD thesis, Massachusetts Institute of Technology, 2008.

Back, E., S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the Fourth DARPA Speech and Natural Language Workshop*, pages 306–311, 1991.

Baldwin, Timothy and Valia Kordoni, editors. *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?* Association for Computational Linguistics, Athens, Greece, March 2009.

Bangalore, Srinivas, Pierre Boullier, Alexis Nasr, Owen Rambow, and Benoît Sagot. MICA: A Probabilistic Dependency Parser Based on Tree Insertion Grammars (Application Note). In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 185–188, Boulder, Colorado, June 2009. Association for Computational Linguistics.

Bansal, Mohit and Dan Klein. Simple, Accurate Parsing with an All-Fragments Grammar. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1098–1107, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

Bansal, Mohit and Dan Klein. Web-scale features for full-scale parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 693–702, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

Bar-Hillel, Yehoshua. A quasi-arithmetical notation for syntactic description. *Language*, 29(1):47–58, 1953.

Bauer, Laurie. Some thoughts on dependency grammar. *Linguistics*, 17:301–316, 1979.

Berger, Adam L., Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Comput. Linguist.*, 22 (1):39–71, 1996.

Bikel, Daniel M. Intricacies of Collins' Parsing Model. *Comput. Linguist.*, 30(4): 479–511, 2004a.

Bikel, Daniel M. *On the parameter space of generative lexicalized statistical parsing models.* PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 2004b. Supervisor-Marcus, Mitchell P.

Bloomfield, Leonard. *Language.* New York: Holt, 1933.

Blunsom, Phil and Trevor Cohn. Unsupervised induction of tree substitution grammars for dependency parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1204–1213, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

Bod, Rens. A Computational Model of Language Performance: Data Oriented Parsing. In *Proceedings COLING'92 (Nantes, France)*, pages 855–859. Association for Computational Linguistics, Morristown, NJ, 1992.

Bod, Rens. Using an Annotated Language Corpus as a Virtual Stochastic Grammar. In *AAAI*, pages 778–783, 1993.

Bod, Rens. Combining semantic and syntactic structure for language modeling. *CoRR*, cs.CL/0110051, 2001a.

Bod, Rens. What is the Minimal Set of Fragments that Achieves Maximal Parse Accuracy? In *Proceedings ACL-2001.* Morgan Kaufmann, San Francisco, CA, 2001b.

Bod, Rens. An efficient implementation of a new DOP model. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics - Volume 1*, EACL '03, pages 19–26, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

Bod, Rens. Unsupervised parsing with U-DOP. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 85–92, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

Bod, Rens and Ronald Kaplan. *Data-Oriented Parsing*, chapter A Data-Oriented Parsing Model for Lexical-Functional Grammar. In Bod et al. (2003), 2003.

Bod, Rens, Khalil Sima'an, and Remko Scha. *Data-Oriented Parsing.* University of Chicago Press, Chicago, IL, USA, 2003.

Boersma, P. and B. Hayes. Empirical tests of the gradual learning algorithm. *Linguistic Inquiry*, 32(1):45–86, 2001.

Bonnema, Remko, Rens Bod, and Remko Scha. A DOP model for semantic interpretation. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, ACL '98, pages 159–167, Stroudsburg, PA, USA, 1997. Association for Computational Linguistics.

Bonnema, Remko, Paul Buying, and Remko Scha. *A New Probability Model for Data Oriented Parsing*, volume Proceedings of the Twelfth Amsterdam Colloquium, pages 85–90. 1999.

Borensztajn, Gideon and Willem Zuidema. Episodic grammar: a computational model of the interaction between episodic and semantic memory in language processing. In *Proceedings of the 33rd Cognitive Science Conference*, 2011.

Borensztajn, Gideon, Willem Zuidema, and Rens Bod. Children's Grammars Grow More Abstract with Age—Evidence from an Automatic Procedure for Identifying the Productive Units of Language. *Topics in Cognitive Science*, 1 (1):175–188, January 2009.

Bosco, Cristina, Vincenzo Lombardo, Daniela Vassallo, and Leonardo Lesmo. Building a Treebank for Italian: a Data-driven Annotation Schema. In *Proceedings of the Second International Conference on Language Resources and Evaluation LREC-2000*, pages 99–105, 2000.

Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM.

Brants, Sabine, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. The TIGER Treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, pages 24–41, 2002.

Bresnan, Joan. *Lexical-Functional Syntax*. Blackwell Textbooks in Linguistics. Blackwell Publishers, September 2000.

Buchanan, Bruce G. Mechanizing the Search for Explanatory Hypotheses. *PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association*, 1982:pp. 129–146, 1982.

Buchholz, Sabine and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *In Proc. of CoNLL*, pages 149–164, 2006.

Buchholz, Sabine, Jorn Veenstra, and Walter Daelemans. Cascaded Grammatical Relation Assignment. In *EMNLP/VLC-99*, pages 239–246. ACL, 1999.

Charniak, Eugene. Tree-bank Grammars. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1031–1036, 1996.

Charniak, Eugene. Statistical Parsing with a Context-free Grammar and Word Statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 598–603. AAAI Press/MIT Press, 1997.

Charniak, Eugene. A Maximum-Entropy-Inspired Parser. Technical report, Brown University, Providence, RI, USA, 1999.

Charniak, Eugene and Mark Johnson. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proc. 43nd Meeting of Association for Computational Linguistics (ACL 2005)*, 2005.

Chiang, David and Daniel M. Bikel. Recovering latent information in treebanks. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

Chomsky, N. Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113–124, January 1956.

Chomsky, N. *Aspects of the theory of syntax.* The MIT Press Paperback Series. M.I.T. Press, 1965.

Chomsky, N. *Modular approaches to the study of the mind.* Distinguished graduate research lecture. San Diego State University Press, 1984.

Chomsky, Noam. *Syntactic structures.* Mouton, Den Haag, 1957.

Chomsky, Noam. Remarks on nominalization. In Jacobs, R. and P. Rosenbaum, editors, *Reading in English Transformational Grammar*, pages 184–221. Ginn and Co., Waltham, 1970.

Cinková, Silvie, Eva Hajičová, Jarmila Panevová, and Petr Sgall. Two Languages – One Annotation Scenario? Experience from the Prague Dependency Treebank. *The Prague Bulletin of Mathematical Linguistics*, (89):5—22, June 2008.

Cinková, Silvie, Josef Toman, Jan Hajič, Kristýna Čermáková, Václav Klimeš, Lucie Mladová, Jana Šindlerová, Kristýna Tomšů, and Zdeněk Žabokrtsky. Tectogrammatical Annotation of the Wall Street Journal. *The Prague Bulletin of Mathematical Linguistics*, (92):85—104, December 2009.

Clark, Stephen and James Curran. Formalism-Independent Parser Evaluation with CCG and DepBank. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 248–255, Prague, Czech Republic, June 2007a. Association for Computational Linguistics.

Clark, Stephen and James R. Curran. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Comput. Linguist.*, 33(4):493–552, 2007b.

Cohn, Trevor, Sharon Goldwater, and Phil Blunsom. Inducing Compact but Accurate Tree-Substitution Grammars. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 548–556, Boulder, Colorado, June 2009. Association for Computational Linguistics.

Cohn, Trevor, Phil Blunsom, and Sharon Goldwater. Inducing Tree-Substitution Grammars. *Journal of Machine Learning Research*, 11:3053–3096, 2010.

Collins, Michael. Three generative, lexicalised models for statistical parsing. In *ACL-35: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23, Morristown, NJ, USA, 1997. Association for Computational Linguistics.

Collins, Michael. Discriminative Reranking for Natural Language Parsing. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 175–182, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

Collins, Michael and Nigel Duffy. Convolution Kernels for Natural Language. In Dietterich, Thomas G., Suzanna Becker, and Zoubin Ghahramani, editors, *NIPS*, pages 625–632. MIT Press, 2001.

Collins, Michael and Nigel Duffy. New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 263–270, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

Collins, Michael and Terry Koo. Discriminative Reranking for Natural Language Parsing. *Comput. Linguist.*, 31(1):25–70, 2005.

Collins, Michael and Brian Roark. Incremental parsing with the perceptron algorithm. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

Collins, Michael J. *Head-Driven Statistical Models for Natural Language Parsing.* PhD thesis, University of Pennsylvania, 1999.

Collins, Michael John. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 184–191, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

Corbett, Greville G., Norman M. Fraser, and Scott McGlashan. *Heads in Grammatical Theory.* Cambridge University Press, New York, 2006.

Covington, Michael A. GB Theory as Dependency Grammar. Technical Report AI-1992-03, University of Georgia, Athens, Georgia, 1992.

Daelemans, Walter, Sabine Buchholz, and Jorn Veenstra. Memory-based Shallow Parsing. In *Proceedings of CoNLL-1999*, Bergen, Norway, 1999.

Dalrymple, Mary. *Lexical-Functional Grammar (Syntax and Semantics).* Academic Press, 2001.

Daumé III, Hal and Daniel Marcu. NP Bracketing by Maximum Entropy Tagging and SVM Reranking. In *The 2004 Conference on Empirical Methods in Natural Language Processing*, pages 254–261, 2004.

De Marneffe, Marie-Catherin, Bill Maccartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *In LREC 2006*, 2006.

De Marneffe, Marie-Catherine and Christopher D. Manning. The Stanford Typed Dependencies Representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, Manchester, UK, August 2008. Coling 2008 Organizing Committee.

de Saussure, Ferdenand. *Course in General Linguistics.* McGraw Hill, New York, 1915.

Dinarelli, Marco, Alessandro Moschitti, and Giuseppe Riccardi. Re-ranking models for spoken language understanding. In *EACL '09: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 202–210, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

Dubey, Amit. What to do when lexicalization fails: parsing German with suffix analysis and smoothing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 314–321, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

Eisner, Jason. Bilexical Grammars And A Cubic-Time Probabilistic Parser. In *In Proceedings of the International Workshop on Parsing Technologies*, pages 54–65, 1997.

Eisner, Jason. Bilexical Grammars and Their Cubic-Time Parsing Algorithms. In Bunt, Harry and Anton Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. Kluwer Academic Publishers, October 2000.

Eisner, Jason and Giorgio Satta. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, ACL '99, pages 457–464, Stroudsburg, PA, USA, 1999. Association for Computational Linguistics.

Eisner, Jason M. An Empirical Comparison of Probability Models for Dependency Grammar. Technical Report IRCS-96-11, University of Pennsylvania, 1996a.

Eisner, Jason M. Three new probabilistic models for dependency parsing: an exploration. In *Proceedings of the 16th conference on Computational linguistics*, pages 340–345, Morristown, NJ, USA, 1996b. Association for Computational Linguistics.

Fillmore, Charles J., Paul Kay, and Mary C. O'Connor. Regularity and Idiomaticity in Grammatical Constructions: The Case of Let Alone. *Language*, 64(3): 501–538, 1988.

Forst, Martin, Núria Bertomeu, Berthold Crysmann, Frederik Fouvry, Silvia Hansen-Schirra, and Valia Kordoni. Towards a dependency-based gold standard for German parsers - The TiGer Dependency Bank, 2004.

Frank, Stefan L and Rens Bod. Insensitivity of the human sentence-processing system to hierarchical structure. *Psychological Science*, (May), 2011.

Fraser, Alexander, Renjing Wang, and Hinrich Schütze. Rich bitext projection features for parse reranking. In *EACL '09: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 282–290, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

Frazier, Lyn. *On Comprehending Sentences: Syntactic Parsing Strategies.* PhD thesis, University of Massachusetts, Indiana University Linguistics Club, 1979.

Gaifman, Haim. Dependency systems and phrase-structure systems. *Information and Control*, 8(3):304 – 337, 1965.

Goldberg, A.E. *Constructions: A Construction Grammar Approach to Argument Structure.* University Of Chicago Press, 1995.

Goldwater, Sharon, Thomas L. Griffiths, and Mark Johnson. Distributional Cues to Word Boundaries: Context is Important. In Bamman, David, Tatiana Magnitskaia, and Colleen Zaller, editors, *Proceedings of the 31st Annual Boston University Conference on Language Development*, pages 239—250, 2007.

Goldwater, Sharon, Thomas L. Griffiths, and Mark Johnson. A Bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112 (1):21–54, 2009.

Goodman, Joshua. Efficient algorithms for parsing the DOP model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 143–152, 1996.

Goodman, Joshua. Probabilistic Feature Grammars. In *In Proceedings of the International Workshop on Parsing Technologies*, pages 89–100, 1997.

Goodman, Joshua. *Data-Oriented Parsing*, chapter Efficient Parsing of DOP with PCFG-Reductions. In Bod et al. (2003), 2003.

Goodman, Joshua T. *Parsing inside-out.* PhD thesis, Harvard University, Cambridge, MA, USA, 1998. Adviser-Shieber, Stuart.

Hajič, Jan, Alena Böhmová, Eva Hajičová, and Barbora Vidová-Hladká. The Prague Dependency Treebank: A Three-Level Annotation Scenario. In Abeillé, A., editor, *Treebanks: Building and Using Parsed Corpora*, pages 103–127. Amsterdam:Kluwer, 2000.

Hale, John. Uncertainty About the Rest of the Sentence. *Cognitive Science*, 30: 643–672, 2006.

Hall, Johan, Joakim Nivre, and Jens Nilsson. Discriminative Classifiers for Deterministic Dependency Parsing. In *ACL*. The Association for Computer Linguistics, 2006.

Hall, Keith, Jiri Havelka, and David A. Smith. Log-Linear Models of Non-Projective Trees, *k*-best MST Parsing and Tree-Ranking. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 962–966, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Harris, Zelig S. *Structural Linguistics.* University of Chicago Press, 1951.

Hays, David G. Grouping and dependency theories. In *National Symposium on Machine Translation*, pages 258–266, Englewood Cliffs, NY, USA, 1960.

Hays, David G. Dependency Theory: A Formalism and Some Observations. *Language*, 40(4):pp. 511–525, 1964.

Hearne, Mary and Andy Way. Disambiguation Strategies for Data-Oriented Translation. In *Proceedings of the 11th Conference of the European Association for Machine Translation*, pages 59–68, 2006.

Heringer, James T. Review of Gaifman (1965). *Ohio State University Working Papers in Linguistics*, 1967.

Hockenmaier, Julia. *Data and models for statistical parsing Data and models for statistical parsing with Combinatory Categorial Grammar.* PhD thesis, University of Edinburgh, 2003.

Hockenmaier, Julia and Mark Steedman. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396, 2007.

Huang, Liang. Forest Reranking: Discriminative Parsing with Non-Local Features. In *Proceedings of ACL-08: HLT*, pages 586–594, Columbus, Ohio, June 2008. Association for Computational Linguistics.

Hudson, R.A. *English word grammar.* B. Blackwell, 1991.

Hudson, Richard. *An Introduction to Word Grammar.* Cambridge University Press, 2010.

Hudson, Rodney. Zwicky on heads. *Journal of Linguistics*, 23:109–132, 1987.

Hurford, James R. Human uniqueness, learned symbols and recursive thought. *European Review*, 12(04):551–565, October 2004.

Hwa, Rebecca. An empirical evaluation of Probabilistic Lexicalized Tree Insertion Grammars. In *Proceedings of the 17th international conference on Computational linguistics*, pages 557–563, Morristown, NJ, USA, 1998. Association for Computational Linguistics.

Jackendoff, Ray. *Foundations of Language.* Oxford University Press, Oxford, UK, 2002.

Jackendoff, Ray S. *X Syntax: A Study of Phrase Structure.* The MIT Press, Cambridge, MA, 1977.

Jespersen, Otto. *Analytic syntax.* London, 1937.

Johansson, Richard and Pierre Nugues. Extended Constituent-to-Dependency Conversion for English. In *Proceedings of NODALIDA 2007*, Tartu, Estonia, May 2007.

Johnson, Mark. PCFG models of linguistic tree representations. *Comput. Linguist.*, 24(4):613–632, 1998.

Johnson, Mark. Transforming Projective Bilexical Dependency Grammars into efficiently-parsable CFGs with Unfold-Fold. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 168–175, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Johnson, Mark. How the Statistical Revolution Changes (Computational) Linguistics. In *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*, pages 3–11, Athens, Greece, March 2009. Association for Computational Linguistics.

Johnson, Mark and Ahmet Engin Ural. Reranking the Berkeley and Brown Parsers. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 665–668, Los Angeles, California, June 2010. Association for Computational Linguistics.

Johnson, Mark, Thomas Griffiths, and Sharon Goldwater. Bayesian Inference for PCFGs via Markov Chain Monte Carlo. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 139–146, Rochester, New York, April 2007a. Association for Computational Linguistics.

Johnson, Mark, Thomas L. Griffiths, and Sharon Goldwater. Adaptor Grammars: A Framework for Specifying Compositional Nonparametric Bayesian Models. In *Advances in Neural Information Processing Systems*, volume 16, pages 641–648, 2007b.

Joshi, Aravind K. Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions. In Dowty, D. R., L. Karttunen, and A. Zwicky, editors, *Natural Language Parsing*, chapter 6, pages 206–250. Cambridge University Press, New York, 1985.

Joshi, Aravind K. and Yves Schabes. Tree-adjoining grammars and lexicalized grammars. In Nivat, Maurice and Andreas Podelski, editors, *Definability and Recognizability of Sets of Trees*. Elsevier, 1991.

Kahane, Sylvain. Bubble trees and syntactic representations. In Krieger, Becker &, editor, *Proceedings 5th Meeting of the Mathematics of Language (MOL5)*. Saarbrücken: DFKI, 1997.

Kahane, Sylvain. Polarized Unification Grammars. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 137–144, Sydney, Australia, July 2006. Association for Computational Linguistics.

Kaplan, R. M. and J. Bresnan. Lexical-Functional Grammar: A Formal System for Grammatical Representation. In Bresnan, J., editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA, 1982.

Kay, Paul and Charles J. Fillmore. Grammatical Constructions and Linguistic Generalizations: the What's X Doing Y? Construction. *Language*, 75:1–33, 1997.

Klavans, Judith L. and Philip Resnik, editors. *The Balancing Act: combining symbolic and statistical approaches to language.* MIT Press, Cambridge, MA, 1996.

Klein, Dan. *The unsupervised learning of natural language structure.* PhD thesis, Stanford University, Stanford, CA, USA, 2005. Adviser-Manning, Christopher D.

Klein, Dan and Christopher D. Manning. Accurate unlexicalized parsing. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 423–430, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

Kuhlmann, Marco. *Dependency Structures and Lexicalized Grammars.* PhD thesis, Saarland University, Saarbrücken, Germany, 2007.

Labov, William, editor. *Sociolinguistic Patterns.* University of Philadelphia Press, Philadelphia, 1972.

Lapata, Mirella and Frank Keller. The Web as a Baseline: Evaluating the Performance of Unsupervised Web-based Models for a Range of NLP Tasks. In Susan Dumais, Daniel Marcu and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 121–128, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

Lari, K. and S. J. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

Lass, Roger. *Historical Linguistics and Language Change.* Cambridge University Press, New York, 1997.

Levelt, W.J.M. *Formal grammars in linguistics and psycholinguistics: Applications in Linguistic Theory*, volume 2 of *Janua linguarum: Series minor.* Mouton, 1974.

Levy, Roger. Expectation-based syntactic comprehension. *Cognition*, July 2007.

Lieberman, P. *On the origins of language: an introduction to the evolution of human speech.* Macmillan series in physical anthropology. Macmillan, 1975.

Lieven, Elena, Heike Behrens, Jennifer Speares, and Michael Tomasello. Early syntactic creativity: a usage-based approach. *Journal of Child Language*, 30 (2):333–370, 2003.

Lin, Dekang. A Dependency-based Method for Evaluating Broad-Coverage Parsers. In *In Proceedings of IJCAI-95*, pages 1420–1425, 1995.

Lindsay, Robert K., Bruce G. Buchanan, E. A. Feigenbaum, and Joshua Lederberg. *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project.* McGraw-Hill Companies, Inc, 1980.

Magerman, David M. *Natural Language Parsing as Statistical Pattern Recognition.* PhD thesis, Stanford University, 1994.

Magerman, David M. Statistical Decision-Tree Models for Parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Cambridge, Massachusetts, USA, June 1995. Association for Computational Linguistics.

Marcus, Mitchell, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: annotating predicate argument structure. In *HLT '94: Proceedings of the workshop on Human Language Technology*, pages 114–119, Morristown, NJ, USA, 1994. Association for Computational Linguistics.

Marcus, Mitchell P., Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

Marcus, Mitchell P., Beatrice Santorini, Mary Ann Marcinkiewicz, and Anns Taylor. Treebank-3. Linguistic Data Consortium, Philadelphia, 1999.

Martin, William A., Kenneth W. Church, and Ramesh S. Patil. Preliminary analysis of a breadth-first parsing algorithm: theoretical and experimental results. *Natural language parsing systems*, pages 267–328, 1987.

Matsuzaki, Takuya, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic CFG with latent annotations. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 75–82, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

McClosky, David, Eugene Charniak, and Mark Johnson. Reranking and self-training for parser adaptation. In *ACL-44: Proceedings of the 21st International*

Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, pages 337–344, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

McDonald, Ryan. *Discriminative learning and spanning tree algorithms for dependency parsing.* PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 2006.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

Mel'čuk, Igor. *Dependency Syntax: Theory and Practice.* State University of New York Press, 1988.

Mel'čuk, Igor. Levels of Dependency in Linguistic Description: Concepts and Problems. Unpublished manuscript, 2003.

Mel'čuk, Igor A. *Studies in dependency syntax.* Karoma Publishers, Ann Arbor, 1979.

Mitchell, Jeff and Mirella Lapata. Language Models Based on Semantic Composition. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 430–439, Singapore, 2009.

Moschitti, Alessandro. Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In *ECML*, pages 318–329, Berlin, Germany, September 2006. Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Proceedings.

Mylonakis, Markos and Khalil Sima'an. Phrase translation probabilities with ITG priors and smoothing as learning objective. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 630–639, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

Ng, Andrew Y. and Michael I. Jordan. On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. In *NIPS*, pages 841–848, 2001.

Nivre, Joakim. An Efficient Algorithm for Projective Dependency Parsing. In *Eighth International Workshop on Parsing Technologies*, Nancy, France, 2003.

Nivre, Joakim. *Inductive Dependency Parsing (Text, Speech and Language Technology).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

Nivre, Joakim. Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore, August 2009. Association for Computational Linguistics.

Nivre, Joakim and Jens Nilsson. Pseudo-Projective Dependency Parsing. In *ACL*. The Association for Computer Linguistics, 2005.

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan Mcdonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

O'Donnell, Timothy J., Noah D. Goodman, and Joshua B. Tenenbaum. Fragment Grammars: Exploring Computation and Reuse in Language. Technical Report MIT-CSAIL-TR-2009-013, MIT, 2009.

Oflazer, Kemal, Bilge Say, Dilek Zeynep Hakkani-tür, and Gökhan Tür. Building A Turkish Treebank. In Abeillé, A., editor, *Treebanks: Building and Using Parsed Corpora*, pages 261–277. Kluwer Academic Publishers, 2003.

Pereira, Fernando and Yves Schabes. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th annual meeting on Association for Computational Linguistics*, pages 128–135, Morristown, NJ, USA, 1992. Association for Computational Linguistics.

Petrov, Slav. *Coarse-to-Fine Natural Language Processing*. PhD thesis, University of California at Bekeley, Berkeley, CA, USA, 2009.

Petrov, Slav and Dan Klein. Improved Inference for Unlexicalized Parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York, April 2007. Association for Computational Linguistics.

Petrov, Slav, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

Pinker, Stephen and Ray Jackendoff. The faculty of language: what's special about it? *Cognition*, 95:201–236, 2005.

Polguère, Alain and Igor A. Mel'čuk, editors. *Dependency in Linguistic Description.* Studies in Language Companion Series 111. John Benjamins, Philadelphia, 2009.

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

Post, Matt and Daniel Gildea. Bayesian Learning of a Tree Substitution Grammar. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 45–48, Suntec, Singapore, August 2009. Association for Computational Linguistics.

Prescher, Detlef. A Tutorial on the Expectation-Maximization Algorithm Including Maximum-Likelihood Estimation and EM Training of Probabilistic Context-Free Grammars. In *ESSLLI*, 2003.

Prescher, Detlef. Inside-Outside Estimation Meets Dynamic EM. *CoRR*, abs/cs/0412016, 2004.

Prescher, Detlef. Head-driven PCFGs with latent-head statistics. In *Parsing '05: Proceedings of the Ninth International Workshop on Parsing Technology*, pages 115–124, Morristown, NJ, USA, 2005a. Association for Computational Linguistics.

Prescher, Detlef. Inducing head-driven PCFGs with latent heads: Refining a tree-bank grammar for parsing. In *In ECML'05*, 2005b.

Ratnaparkhi, Adwait. A Linear Observed Time Statistical Parser Based on Maximum Entropy Models. *CoRR*, cmp-lg/9706014, 1997.

Ratnaparkhi, Adwait. Learning to Parse Natural Language with Maximum Entropy Models. *Mach. Learn.*, 34(1-3):151–175, 1999.

Riezler, Stefan, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Mark Johnson. Parsing the Wall Street Journal using a Lexical-Functional Grammar and Discriminative Estimation Techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02)*, Philadelphia, PA, USA, 2002.

Rimell, Laura, Stephen Clark, and Mark Steedman. Unbounded dependency recovery for parser evaluation. In *EMNLP '09: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 813–821, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

Robinson, Jane J. Methods for obtaining corresponding phrase structure and dependency grammars. In *Proceedings of the 1967 conference on Computational* *linguistics*, COLING '67, pages 1–25, Stroudsburg, PA, USA, 1967. Association for Computational Linguistics.

Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Reviews*, pages 386–408, 1958.

Sampson, Geoffrey, Robin Haigh, and Eric Atwell. Natural language analysis by stochastic optimization: a progress report on project APRIL. *J. Exp. Theor. Artif. Intell.*, 1:271–287, October 1989.

Sangati, Federico. A Probabilistic Generative Model for an Intermediate Constituency-Dependency Representation. In *Proceedings of the ACL 2010 Student Research Workshop*, pages 19–24, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

Sangati, Federico and Chiara Mazza. An English Dependency Treebank à la Tesnière. In *The 8th International Workshop on Treebanks and Linguistic Theories*, pages 173–184, Milan, Italy, 2009.

Sangati, Federico and Willem Zuidema. Unsupervised Methods for Head Assignments. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 701–709, Athens, Greece, March 2009. Association for Computational Linguistics.

Sangati, Federico and Willem Zuidema. A Recurring Fragment Model for Accurate Parsing: Double-DOP . In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 84–95, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.

Sangati, Federico, Willem Zuidema, and Rens Bod. A generative re-ranking model for dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 238–241, Paris, France, October 2009. Association for Computational Linguistics.

Sangati, Federico, Willem Zuidema, and Rens Bod. Efficiently Extract Recurring Tree Fragments from Large Treebanks. In Chair), Nicoletta Calzolari (Conference, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010. European Language Resources Association (ELRA).

Scha, Remko. Taaltheorie en taaltechnologie: competence en performance. In de Kort, Q. A. M. and G. L. J. Leerdam, editors, *Computertoepassingen in de Neerlandistiek*, LVVN-jaarboek, pages 7–22. Landelijke Vereniging van Neerlandici, Almere, 1990. [Language theory and language technology: Competence and Performance] in Dutch.

Schabes, Yves and Richard C. Waters. Tree insertion grammar: a cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Fuzzy Sets Syst.*, 76(3):309–317, 1995.

Schneider, Gerold. A Linguistic Comparison of Constituency, Dependency and Link Grammar, 1998. MSc Thesis.

Seginer, Yoav. *Learning Syntactic Structure.* PhD thesis, Institute for Logic Language and Computation, University of Amsterdam, 2007.

Sekine, S. and M. J. Collins. EVALB bracket scoring program. `http://nlp.cs.nyu.edu/evalb/`, 1997.

Semecký, Jiří and Silvie Cinková. Constructing an English Valency Lexicon In: Proceedings of Frontiers in Linguistically Annotated Corpora. In *The Association for Computational Linguistics*, pages 111–113, Sydney, Australia, 2006.

Sgall, Petr, Eva Hajičová, and Jarmilla Panevová. *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects.* Reidel, Dordrecht, 1986.

Shen, L., A. Sarkar, and F. Och. Discriminative reranking for machine translation, 2004.

Shen, Libin, Anoop Sarkar, and Aravind K. Joshi. Using LTAG based features in parse reranking. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 89–96, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

Shieber, Stuart M. Sentence disambiguation by a shift-reduce parsing technique. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 113–118, Morristown, NJ, USA, 1983. Association for Computational Linguistics.

Sima'an, K. Tree-gram parsing lexical dependencies and structural relations. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 37–44, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

Sima'an, K., A. Itai, Y. Winter, A. Altman, and N. Nativ. Building a Tree-Bank of Modern Hebrew Text. *Journal Traitement Automatique des Langues (T.A.L.)*, 2001.

Sima'an, Khalil. An Optimized Algorithm for Data-Oriented Parsing. In *International Conference on Recent Advances in Natural Language Processing (RANLP'95)*, Tzigov Chark, Bulgaria, 1995.

Sima'an, Khalil. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of the 16th conference on Computational linguistics*, pages 1175–1180, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

Sima'an, Khalil. *Learning Efficient Disambiguation.* PhD thesis, Utrecht University and University of Amsterdam, 1999.

Sima'an, Khalil. On Maximizing Metrics for Syntactic Disambiguation. In *Proceedings of the International Workshop on Parsing Technologies (IWPT'03)*, 2003.

Skut, Wojciech, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. An Annotation Scheme for Free Word Order Languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing ANLP-97*, Washington, DC, 1997.

Smith, Noah A. *Linguistic Structure Prediction.* Synthesis Lectures on Human Language Technologies. Morgan and Claypool, May 2011.

Steedman, Mark. *Surface structure and interpretation.* Linguistic inquiry monographs, 30. MIT Press, 1996.

Steedman, Mark. On becoming a discipline. *Comput. Linguist.*, 34:137–144, March 2008.

Sugayama, Kensei and Richard A. Hudson, editors. *Word Grammar: New Perspectives on a Theory of Language Structure.* Continuum International Publishing Group Ltd, New York, 2005.

Tanner, Bernard. Parsing. *The English Journal*, 52(1):67, January 1963.

Taylor, Ann, Mitchell Marcus, and Beatrice Santorini. *The Penn Treebank: An Overview*, pages 5–22. Volume 20 of *Text, Speech and Language Technology* Abeillé (2003), 2003.

Tesnière, Lucien. *Eléments de syntaxe structurale.* Editions Klincksieck, Paris, 1959.

Vadas, David and James Curran. Adding Noun Phrase Structure to the Penn Treebank. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 240–247, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Vapnik, Vladimir N. *Statistical Learning Theory.* Wiley-Interscience, September 1998.

Vater, Heinz. Toward a generative dependency grammar. *Lingua*, 36(2-3):121 – 145, 1975.

Villavicencio, Aline. Learning to distinguish PP arguments from adjuncts. In *Proceedings of the 6th conference on Natural language learning - Volume 20*, COLING-02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

Wardhaugh, R. *An introduction to sociolinguistics*. Blackwell textbooks in linguistics. Blackwell Pub., 2006.

Wells, Rulon S. Immediate Constituents. *Language*, 23(2):pp. 81–117, 1947.

White, Michael and Rajakrishnan Rajkumar. Perceptron reranking for CCG realization. In *EMNLP '09: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 410–419, Morristown, NJ, USA, 2009. Association for Computational Linguistics.

Wu, C. F. Jeff. On the Convergence Properties of the EM Algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

Wundt, W.M. *Völkerpsychologie: bd, 1.-2. t. Die Sprache*. Völkerpsychologie: Eine untersuchung der entwicklungsgesetze von sprache, mythus und sitte. W. Engelmann, 1900.

Xue, Jing-Hao and D. Titterington. Comment on "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes". *Neural Processing Letters*, 28:169–187, 2008. 10.1007/s11063-008-9088-7.

Xue, Nianwen, Fu-Dong Chiou, and Martha Palmer. Building a large-scale annotated Chinese corpus. In *Proceedings of the 19th international conference on Computational linguistics - Volume 1*, COLING '02, pages 1–8, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

Yamada, Hiroyasu and Yuji Matsumoto. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of IWPT*, pages 195–206, 2003.

Yamangil, Elif and Stuart M. Shieber. Bayesian synchronous tree-substitution grammar induction and its application to sentence compression. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 937–947, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

Zollmann, Andreas and Khalil Sima'an. A Consistent and Efficient Estimator for Data-Oriented Parsing. *Journal of Automata, Languages and Combinatorics*, 10(2/3):367–388, 2005.

Zuidema, W. *The major transitions in the evolution of language.* PhD thesis, University of Edinburgh, 2005.

Zuidema, Willem. Parsimonious Data-Oriented Parsing. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 551–560, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Zwicky, Arnold M. Heads. *Journal of Linguistics*, 21(1):pp. 1–29, 1985.

Index

apposition, 112, 119
artificial symbols ($\mathcal{A}$), 22, 29

back-off, 52
bayesian inference, 42
bilexical dependency grammars, 91
binarization, 74
block category, 112, 116
block of words, 110, 111
bottom-up grammar, 36
bubble trees, 131

CFG, *see* context-free grammar
check conditions, 23
chunking, 123
conditioning context, 22, 23
conjunct, 112
constituent, 5
content word, 111
context-free grammar, 8, 24, 90
coordination, 112, 117

Data-Oriented Parsing, 54, 55
dependency relation, 86, 111
dependency-structure, 5, **6**, 7, 21, **85**
dependent, 6, 87
discriminative model, 44, 99
dominance relation, 50
DOP, *see* Data-Oriented Parsing
Double-DOP, 55, 73, 76, 78, 83
DS, *see* dependency-structure
DS to PS conversion, 145

Eisner's models, 95
elementary fragment, 22
EM, *see* Expectation-Maximization algorithm
equal weights estimate, 69
evaluation, 13, 76, 102, 124, 142
event space, 22
EWE, *see* equal weights estimate
Expectation-Maximization algorithm, 41, 69

functional categories, 88
functional word, 111

generative events, 22
generative grammar, 18
generative grammar examples, 30
generative model, 4, 18, 44, 95
generative process, 7, 25
Goodman transformation, 57
governor, 6, 87
grammar extraction, 30
grammar symbols ($\mathcal{N}$), 22

head, 6, 51, 87, 89, 93, 116
head-driven models, 50, 51, 86
head-driven phrase-structure grammar, 93
held-out estimation, 59

history, *see* conditioning context
horizontal markovization, 52, 54
HPSG, *see* head-driven phrase-structure grammar

Inside-Outside algorithm, 41, 69, 72
introspective approach, 3
IO, *see* Inside-Outside algorithm

junction, 112, 117
junction operation, 110

lexical functional grammar, 93
LFG, see lexical functional grammar93
likelihood, 41

max constituent parse, 71
max rule sum, 71
maximizing objectives, 70
maximum likelihood estimate, 41, 69
maximum spanning tree, 99
MCP, *see* max constituent parse
meaning text theory, 94
MLE, *see* maximum likelihood estimate
most probable derivation, 70
most probable parse, 71
MPD, *see* most probable derivation
MPP, *see* most probable parse
MRS, *see* max rule sum

null symbol (⊘), 22, 23, 29

parsing, 67, 95
parsing results, 76, 102, 105, 125
PDT, *see* Prague Dependency Treebank
phrase-structure, **5**, 7, 21, **49**
Prague Dependency Treebank, 128
probabilistic model, 10, 66
probabilistic tree-generating grammar, 38
probability distribution, 38
probability distribution estimates, 39
projectivity, 88
PS, *see* phrase-structure
PS to DS conversion, 89
PS to TDS conversion, 119

recurring fragments, 58
relative frequency estimate, 40, 69
reranking, 43, 100, 125
RFE, *see* relative frequency estimate
right sister insertion grammar, 32, 33
rule factorization, 52

sandwich insertion grammar, 34
semantics, 4
smoothing, 74, 104, 147
Stanford typed dependency representation, 128
start symbol (⊙), 22, 29
state-splitting models, 53, 79
stop symbol (⊕), 22, 29
substitution operation, 56
supervised learning, 18
symbolic grammar, 18
symbolic tree-generating grammar, 22
syntactic ambiguity, 19, 20, 38
syntactic representations, 5

TDS, *see* Tesnière Dependency-Structure
Tesnière Dependency-Structure, 10, 21, **109**
The CCG-bank, 131
transference, 110, 113
transition-based model, 100
tree probability, 39
tree structures, 20
tree-adjoining grammar, 35
tree-substitution grammar, 30

unknown words, 74
unsupervised learning, 18

valence, 12, 109, 112

wild-card symbol (⊛), 22, 23, 29
word grammar, 94

X-bar theory, 93

Samenvatting

Deze dissertatie gaat over het leren van syntactische boomstructuren aan de hand van generalisaties over geannoteerde corpora. Verschillende probabilistische modellen worden onderzocht, met drie verschillende representaties.

Corpora voor standaard zinsstructuur *(phrase-structure)* en afhankelijkheidsstructuur *(dependency-structure)* worden gebruikt om de modellen te trainen en te testen. Een derde representatie wordt geïntroduceerd, gebaseerd op een systematische maar compacte formulering van de originele afhankelijkheidstheorie zoals geïntroduceerd door Lucien Tesnière. Deze nieuwe representatie omvat alle voordelen van zinsstructuren en afhankelijkheidsstructuren, en is een toereikend compromis tussen adequaatheid en eenvoud van syntactische beschrijving.

Eén van de belangrijkste bijdragen van deze dissertatie is de formulering van een algemeen kader ('framework') voor het definiëren van generatieve modellen van syntaxis. In elk model vallen de syntactische bomen uiteen in elementaire constructies welke opnieuw gecombineerd kunnen worden teneinde nieuwe syntactische structuren te genereren door middel van specifieke combinatieoperaties.

Voor het leren van zinsstructuren wordt een nieuwe methode van Data-Georiënteerd Ontleden *(Data-Oriented Parsing; DOP)* geïntroduceerd. In navolging van het originele DOP gedachtegoed worden constructies van willekeurige grootte gebruikt als bouwstenen van het model; echter, teneinde de grammatica te beperken tot een kleine doch representatieve verzameling van constructies worden alleen constructies die meerdere keren voorkomen gebruikt als verzameling van voorbeelden *(exemplars)*. Voor het vinden van terugkerende fragmenten is een nieuwe efficiente "tree-kernel"-algoritme ontworpen.

Wat betreft de andere twee representaties: twee generatieve modellen worden geformuleerd en geëvalueerd met behulp van een systeem voor herordenen (*reranking*). Deze simpele methodologie wordt geïntroduceerd in dit werk en kan gebruikt worden bij het simuleren van alternatieve automatische ontleders en bij het (her)definiëren van syntactische modellen.

Abstract

The thesis focuses on learning syntactic tree structures by generalizing over annotated treebanks. It investigates several probabilistic models for three different syntactic representations.

Standard phrase-structure and dependency-structure treebanks are used to train and test the models. A third representation is proposed, based on a systematic yet concise formulation of the original dependency theory proposed by Lucien Tesnière (1959). This new representation incorporates all main advantages of phrase-structure and dependency-structure, and represents a valid compromise between adequacy and simplicity in syntactic description.

One of the main contributions of the thesis is to formulate a general framework for defining probabilistic generative models of syntax. In every model syntactic trees are decomposed in elementary constructs which can be recomposed to generate novel syntactic structures by means of specific combinatory operations.

For learning phrase-structures, a novel Data-Oriented Parsing approach (Bod et al., 2003) is proposed. Following the original DOP framework, constructs of variable size are utilized as building blocks of the model. In order to restrict the grammar to a small yet representative set of constructions, only those recurring multiple times in the training treebank are utilized. For finding recurring fragments a novel efficient tree-kernel algorithm is utilized.

Regarding the other two representations, several generative models are formulated and evaluated by means of a re-ranking framework. This represents an effective methodology, which can function as a parser-simulator, and can guide the process of (re)defining probabilistic generative models for learning syntactic structures.

Titles in the ILLC Dissertation Series:

ILLC DS-2006-01: **Troy Lee**
Kolmogorov complexity and formula size lower bounds

ILLC DS-2006-02: **Nick Bezhanishvili**
Lattices of intermediate and cylindric modal logics

ILLC DS-2006-03: **Clemens Kupke**
Finitary coalgebraic logics

ILLC DS-2006-04: **Robert Špalek**
Quantum Algorithms, Lower Bounds, and Time-Space Tradeoffs

ILLC DS-2006-05: **Aline Honingh**
The Origin and Well-Formedness of Tonal Pitch Structures

ILLC DS-2006-06: **Merlijn Sevenster**
Branches of imperfect information: logic, games, and computation

ILLC DS-2006-07: **Marie Nilsenova**
Rises and Falls. Studies in the Semantics and Pragmatics of Intonation

ILLC DS-2006-08: **Darko Sarenac**
Products of Topological Modal Logics

ILLC DS-2007-01: **Rudi Cilibrasi**
Statistical Inference Through Data Compression

ILLC DS-2007-02: **Neta Spiro**
What contributes to the perception of musical phrases in western classical music?

ILLC DS-2007-03: **Darrin Hindsill**
It's a Process and an Event: Perspectives in Event Semantics

ILLC DS-2007-04: **Katrin Schulz**
Minimal Models in Semantics and Pragmatics: Free Choice, Exhaustivity, and Conditionals

ILLC DS-2007-05: **Yoav Seginer**
Learning Syntactic Structure

ILLC DS-2008-01: **Stephanie Wehner**
Cryptography in a Quantum World

ILLC DS-2008-02: **Fenrong Liu**
Changing for the Better: Preference Dynamics and Agent Diversity

ILLC DS-2008-03: **Olivier Roy**
Thinking before Acting: Intentions, Logic, Rational Choice

ILLC DS-2008-04: **Patrick Girard**
Modal Logic for Belief and Preference Change

ILLC DS-2008-05: **Erik Rietveld**
Unreflective Action: A Philosophical Contribution to Integrative Neuroscience

ILLC DS-2008-06: **Falk Unger**
Noise in Quantum and Classical Computation and Non-locality

ILLC DS-2008-07: **Steven de Rooij**
Minimum Description Length Model Selection: Problems and Extensions

ILLC DS-2008-08: **Fabrice Nauze**
Modality in Typological Perspective

ILLC DS-2008-09: **Floris Roelofsen**
Anaphora Resolved

ILLC DS-2008-10: **Marian Counihan**
Looking for logic in all the wrong places: an investigation of language, literacy and logic in reasoning

ILLC DS-2009-01: **Jakub Szymanik**
Quantifiers in TIME and SPACE. Computational Complexity of Generalized Quantifiers in Natural Language

ILLC DS-2009-02: **Hartmut Fitz**
Neural Syntax

ILLC DS-2009-03: **Brian Thomas Semmes**
A Game for the Borel Functions

ILLC DS-2009-04: **Sara L. Uckelman**
Modalities in Medieval Logic

ILLC DS-2009-05: **Andreas Witzel**
Knowledge and Games: Theory and Implementation

ILLC DS-2009-06: **Chantal Bax**
Subjectivity after Wittgenstein. Wittgenstein's embodied and embedded subject and the debate about the death of man.

ILLC DS-2009-07: **Kata Balogh**
Theme with Variations. A Context-based Analysis of Focus

ILLC DS-2009-08: **Tomohiro Hoshi**
Epistemic Dynamics and Protocol Information

ILLC DS-2009-09: **Olivia Ladinig**
Temporal expectations and their violations

ILLC DS-2009-10: **Tikitu de Jager**
"Now that you mention it, I wonder. . . ": Awareness, Attention, Assumption

ILLC DS-2009-11: **Michael Franke**
Signal to Act: Game Theory in Pragmatics

ILLC DS-2009-12: **Joel Uckelman**
More Than the Sum of Its Parts: Compact Preference Representation Over Combinatorial Domains

ILLC DS-2009-13: **Stefan Bold**
Cardinals as Ultrapowers. A Canonical Measure Analysis under the Axiom of Determinacy.

ILLC DS-2010-01: **Reut Tsarfaty**
Relational-Realizational Parsing

ILLC DS-2010-02: **Jonathan Zvesper**
Playing with Information

ILLC DS-2010-03: **Cédric Dégremont**
The Temporal Mind. Observations on the logic of belief change in interactive systems

ILLC DS-2010-04: **Daisuke Ikegami**
Games in Set Theory and Logic

ILLC DS-2010-05: **Jarmo Kontinen**
Coherence and Complexity in Fragments of Dependence Logic

ILLC DS-2010-06: **Yanjing Wang**
Epistemic Modelling and Protocol Dynamics

ILLC DS-2010-07: **Marc Staudacher**
Use theories of meaning between conventions and social norms

ILLC DS-2010-08: **Amélie Gheerbrant**
Fixed-Point Logics on Trees

ILLC DS-2010-09: **Gaëlle Fontaine**
Modal Fixpoint Logic: Some Model Theoretic Questions

ILLC DS-2010-10: **Jacob Vosmaer**
Logic, Algebra and Topology. Investigations into canonical extensions, duality theory and point-free topology.

ILLC DS-2010-11: **Nina Gierasimczuk**
Knowing One's Limits. Logical Analysis of Inductive Inference

ILLC DS-2011-01: **Wouter M. Koolen**
Combining Strategies Efficiently: High-Quality Decisions from Conflicting Advice

ILLC DS-2011-02: **Fernando Raymundo Velazquez-Quesada**
Small steps in dynamics of information

ILLC DS-2011-03: **Marijn Koolen**
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

ILLC DS-2011-04: **Junte Zhang**
System Evaluation of Archival Description and Access

ILLC DS-2011-05: **Lauri Keskinen**
Characterizing All Models in Infinite Cardinalities

ILLC DS-2011-06: **Rianne Kaptein**
Effective Focused Retrieval by Exploiting Query Context and Document Structure

ILLC DS-2011-07: **Jop Briët**
Grothendieck Inequalities, Nonlocal Games and Optimization

ILLC DS-2011-08: **Stefan Minica**
Dynamic Logic of Questions

ILLC DS-2011-09: **Raul Andres Leal**
Modalities Through the Looking Glass: A study on coalgebraic modal logic and their applications

ILLC DS-2011-10: **Lena Kurzen**
Complexity in Interaction

ILLC DS-2011-11: **Gideon Borensztajn**
The neural basis of structure in language