Learning Syntactic Structure

Yoav Seginer

Learning Syntactic Structure

ILLC Dissertation Series DS-2007-05

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Plantage Muidergracht 24
1018 TV Amsterdam
phone: +31-20-525 6051
fax: +31-20-525 5206
e-mail: `illc@science.uva.nl`
homepage: `http://www.illc.uva.nl/`

Learning Syntactic Structure

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 der Universiteit
op vrijdag 12 oktober 2007, te 10.00 uur

door

Yoav Seginer

geboren te Haifa, Israël

Promotor: Prof.dr. D.H.J. de Jongh
Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Cover: Pieter Brueghel the Elder, *The Ass at School*, 1556. © bpk / Kupferstichkabinett, SMB / Jörg P. Anders. Reproduced with permission.
Printed and bound by PrintPartners Ipskamp.

ISBN-10: 90–5776–166–1
ISBN-13: 978–90–5776–166–9

Contents

Acknowledgments **ix**

1 Introduction **1**
1.1 Background 1
1.2 Some Roads not Taken 3
1.2.1 Psychological and Linguistic Approaches 3
1.2.2 Theoretical Mathematical Models 5
1.3 The Road Taken: Empirical Computational Models 6
1.3.1 The Task 7
1.3.2 Why Study Unsupervised Parsing? 7
1.3.3 On the Use of Meaning in Learning Syntax 9
1.3.4 A Brief Survey of Syntactic Induction 10
1.3.5 From Grammar Induction to Parser Induction 20
1.4 The Road Taken: Learning to Parse Incrementally 21
1.4.1 Incrementality 21
1.4.2 Skewness 22
1.4.3 The Incremental Parser 24
1.4.4 Learning and the Zipfian Distribution 24
1.4.5 Bootstrapping 25
1.5 Organization of this Thesis 25

2 Common Cover Links **27**
2.1 A Quick Preview 29
2.2 Basic Definitions 31
2.3 Common Cover Links 32
2.4 Representative Subsets 35
2.5 Shortest Common Cover Link Sets 36
2.6 Reconstructing the Bracketing 38
2.6.1 Simple Reconstruction 39

2.6.2 Paths 41
2.6.3 Incremental Reconstruction 42
2.7 Characterization of Shortest Common Cover Link Sets 43
2.8 Conclusion 46

3 Parsing 47
3.1 Incremental Parsing 48
3.2 Conditions on Incremental Parsing 50
3.2.1 The Adjacency Properties 51
3.2.2 Equality 53
3.2.3 Resolution 54
3.3 The Parser Algorithm 54
3.4 Parsing Functions 55
3.5 Conclusion 56

4 Linguistic Analysis 59
4.1 Dependency Structures 60
4.1.1 No-Loop Conditions on Dependencies 61
4.1.2 Projective Dependency Structures 62
4.2 Dependencies and Common Cover Links 63
4.2.1 Are Dependencies Common Cover Links? 65
4.2.2 Exocentric Constructions 66
4.2.3 Common Cover Link Depth 69
4.2.4 Adjacency 70
4.3 Reanalysis 72
4.4 Are Links of Depth 0 and 1 Sufficient? 75
4.5 Extensions of the Common Cover Link Representation 81
4.5.1 Non-Projective Structures 81
4.5.2 Coordination 82
4.5.3 Relative Clauses 84
4.5.4 Word Order 85
4.6 Conclusion 85

5 Mathematical Analysis and Proofs 87
5.1 Basic Definitions and Properties 88
5.1.1 Basic Properties of the Common Cover Link Set 89
5.1.2 Representative and Shortest Common Cover Link Sets . . 92
5.1.3 Directional Uniqueness 95
5.1.4 Paths 97
5.1.5 Simple Reconstruction Algorithm 98
5.2 Bracket Reconstruction 100
5.2.1 Reconstruction from Representative Subsets 101
5.2.2 A Version of Linear Transitivity 103

5.2.3 Putting It All Together 105
5.3 Characterization of Shortest Common Cover Link Sets 105
5.3.1 The Characterization 105
5.3.2 Characterization Conditions are Necessary 108
5.3.3 Characterization Conditions are Sufficient 110
5.4 Incremental Reconstruction 121
5.4.1 Generators 123
5.4.2 Correctness of Incremental Reconstruction 126
5.5 Incremental Parsing 132
5.5.1 Adjacency 133
5.5.2 Incremental Parser 136

6 Learning and Parsing **147**
6.1 The Parsing Function Family 148
6.1.1 Greedy Parsing Functions 148
6.1.2 A Lexicalized Family of Parsing Functions 150
6.2 The Learning Process 151
6.2.1 Lexicon Update 152
6.2.2 Prefix Independence 156
6.3 Basic Parsing Function Family 158
6.3.1 Adjacency Point Update 158
6.3.2 The Weight Function 162
6.3.3 Simplified Basic Parsing Function Family 167
6.3.4 On the Use of Punctuation 167
6.4 Conclusion 169

7 Experiments **171**
7.1 Introduction 171
7.2 Experiments and Evaluation 173
7.2.1 Input to the Parser 174
7.2.2 Evaluation 176
7.3 Baselines 177
7.4 Results and Discussion 179
7.4.1 Results 179
7.4.2 Baselines 183
7.4.3 Incremental Parsers Compared with Baseline 183
7.4.4 Comparing Different Incremental Parser Variants 184
7.4.5 The Effect of Punctuation 185
7.4.6 Comparison with Other Parsers 186
7.4.7 Error Analysis 188
7.4.8 Non-Binary Branching Bracketing 192
7.4.9 Parsing Speed 193
7.5 Conclusion 194

8 Conclusions **197**

Bibliography **199**

Index **207**

Samenvatting **211**

Acknowledgments

It was Dick de Jongh who suggested to me the problem of language learning as the subject of this dissertation. At the time, that was not exactly what I had in mind, but Dick managed to convince me in a very simple way: he left me Makoto Kanazawa's work to read, attaching a short note: "another interesting subject". This was the beginning of a long journey together, which Dick supervised with patience and dedication. From beginning to end, his influence was exerted by short comments and questions which had a significant influence on the outcome. I would like to thank him for all this, for his attention to detail and for much more. This work would not have existed without him.

Khalil Sima'an was the second major influence on the content of this work. I first met Khalil when I followed his course on statistical methods for natural language processing. This was my first encounter with natural language processing in general and with statistical methods in natural language processing in particular. Though it did not happen in a day, it was Khalil's influence that had made me turn from theoretical mathematical research to empirical work based on corpora. Khalil taught me most of what I know about statistical computational linguistics and much more that I probably should know, I could have learned from him.

When it came to discussing my work or anything even remotely related, it was always interesting to talk with Jelle Zuidema. Whatever the subject, Jelle always reminded me what the bigger question was. Jelle was also the organizer and the driving force behind our weekly reading group and seminar. This weekly meeting was a great place for discussion, learning new things and exchanging ideas and the discussion was always lively. It was also great fun. I would like to thank Jelle for organizing these meetings and all the participants for making them such a success: Remko Scha, Reut Tsarfaty, Henk Zeevat, Khalil Sima'an, Detlef Prescher, Rens Bod and Anouk Perquin as well as many other occasional participants. Beyond these weekly meetings, these people have always been willing to discuss and give advice whenever it was needed. I would especially like to thank Reut Tsarfaty and Jelle Zuidema for their valuable comments on various drafts of this thesis. I would also like to thank Fenrong Liu for help with the Chinese Treebank.

Jacqueline van Kampen and Pieter Adriaans read some early proposals which led to the work in this book. I would like to thank them for their useful comments and the interesting discussion. I also thank Alex Clark, Khalil Sima'an, Rens Bod and Maarten de Rijke for their comments, which have greatly improved the final version of this thesis.

It was always pleasant working at the ILLC. During this time, I have enjoyed sharing an office with Marco Vervoort, Brian Semmes, Evangelos Tzanis, Neta Spiro, Sieuwert van Otterloo, Olivia Ladinig and Joop Niekus. With Joost Joosten I have shared more than one office and many laughs. Special thanks go to Merlijn Sevenster and Aline Honingh who had been my longest office mates. I would like to thank them for accepting my Dutch as it was and for helping making it what it is, for starting the day early and having lunch at a reasonable time, and for their friendship. I would also like to thank all the many others who made conversation at lunchtime better than the food: Nick Bezhanishvili, Gideon Borensztajn, Ulle Endriss, Clemens Kupke, Fenrong Liu, Eric Pacuit, Olivier Roy, Federico Sangati, Leigh Smith and Jacob Vosmaer.

The ILLC office, Marjan Veldhuisen, Tanja Kassenaar, Jessica Pogorzelski, Karin Gigengack, Peter van Ormondt and Ingrid van Loon were always happy to help with any administrative problem. They also kept the Monday morning coffee break going, against all odds.

Finally, I thank my family, for everything.

Amsterdam   Yoav Seginer
August, 2007.

Chapter 1
Introduction

Syntactic structure plays a central role in most theories of language, but it cannot be directly observed. An important question, therefore, is whether there is a relation between syntactic structure and immediately observable properties of language, such as the statistics of the words and sentences that we hear and read. Finding such a relation has important consequences for the problem of language acquisition by children, as well as implications for the theory of syntax itself. It can also be used in engineering language processing systems.

This thesis addresses the problem of finding the relation between the surface statistics of a language and its hidden syntactic structure by developing a parser which attempts to capture this relation. The parser is tested to determine its agreement with the syntactic structure which linguists assign to utterances. While this approach does not try to model the way humans learn and process language, the design of the parser relies on some well-known properties of language and language processing by humans. By selecting both the representation of syntactic structure and the language statistics in a way which agrees well with these properties of language, the resulting relation, as coded by the parser, is simple.

1.1 Background

One of the more notable facts about language is that children can learn it without explicitly being informed of its structure and meaning, as already observed by Saint Augustine in his *Confessions* (I 8):

> Passing hence from infancy, I came to boyhood, or rather it came to me, displacing infancy. Nor did that depart (for whither went it?) and yet it was no more. For I was no longer a speechless infant, but a speaking boy. This I remember; and have since observed how I learned to speak. It was not that my elders taught me words (as,

> soon after, other learning) in any set method; but I, longing by cries and broken accents and various motions of my limbs to express my thoughts, that so I might have my will, and yet unable to express all I willed, or to whom I willed, did myself, by the understanding which Thou, my God, gavest me, practise the sounds in my memory. When they named any thing, and as they spoke turned towards it, I saw and remembered that they called what they would point out by the name they uttered. And that they meant this thing and no other was plain from the motion of their body, the natural language, as it were, of all nations, expressed by the countenance, glances of the eye, gestures of the limbs, and tones of the voice, indicating the affections of the mind, as it pursues, possesses, rejects, or shuns. And thus by constantly hearing words, as they occurred in various sentences, I collected gradually for what they stood.[1]

Saint Augustine's theory of language learning certainly may sound plausible to the modern reader,[2] but, whether correct or not, it merely describes the learning of the meaning of words (and only of some words, for that matter). This is at best a modest beginning, since to properly understand and use language, a child does not only need to learn the meaning of individual words, but must also understand how these can be combined to produce complex linguistic structures, such as sentences. The way the words are arranged together is described by the syntax of the language, and different languages not only use different words, but also have different syntactic structures. A child must, therefore, not only learn the meaning of individual words but must also learn the rules governing the syntax of the language. Because these rules are abstract, they cannot be learned simply by associating them with objects in the real world. At the same time, this abstract nature of syntactic rules and their relatively loose connection with meaning have made syntax one of the most formally described components of natural language. For these formal syntactic systems, concrete learning algorithms can be designed and tested. Such an algorithm is the subject of the present work.

When studying natural language learning, one may consider various methods by which the rules governing the use and interpretation of a language can be deduced from exposure to utterances of that language. There may be different ways and settings in which this goal can be achieved and the way children acquire their first language may represent only one possible method for doing so. Here I will use *language acquisition* to refer to *child language acquisition*, the specific process by which children learn their first language, while *language learning* will refer more generally to any method of learning.

Language acquisition by children is, of course, not merely a specific instance of natural language learning but also the most successful learning method known

[1]English translation by Edward Bouverie Pusey.
[2]who has not read Wittgenstein's *Philosophical Investigations*.

to date. This is not surprising, as it is only the successful learning of language by children which allows natural languages to exist at all and the acquisition process thus defines the range of possible languages (Deacon 1997; Kirby and Hurford 2002; Zuidema 2003). The possibility remains, however, that additional learning methods exist. This is especially true if the setting in which learning takes place is not identical to that in which children learn their first language. For example, a computer may be required to learn the syntax of a language from written text. This input is clearly very different from the speech signal which a child is exposed to when acquiring a language, and may require different learning algorithms. This does not mean, however, that such learning algorithms are not relevant to the study of child language acquisition. I will discuss this later on in the introduction.

There are many ways to approach the problem of language learning and of syntax learning in particular. These approaches differ not only in the solutions they offer, but in the questions they pose and in the way they set out to answer these questions. Of the many roads available, I have chosen to travel down one: to design and test empirically (on large corpora of text) an algorithm which learns to parse from unannotated example sentences. This thesis does not, therefore, address the question of child language acquisition directly but is concerned with the more general problem of language learning.

Before going down this road, the introduction is a fitting place to take a quick look down the roads not taken. I therefore begin with a very brief mention of other approaches to the study of language learning and acquisition: in psychology, in theoretical linguistics and in theoretical computer science. This should allow the reader to place the current work within a wider context, but the emphasis is on mentioning, rather than discussing, the main alternative approaches. I then go on to explain what one may expect to learn from the approach adopted in this thesis, which uses real language input (though not necessarily that which is available to children) to simulate learning on a computer (rather than observing it in children). Next, taking a first step down the road chosen, I look more closely at previous computer algorithms designed to learn syntactic structure in settings similar to those I use. Having thus looked at all the roads I could have taken, I conclude the introduction by describing the road I have taken and those properties of it which I find most attractive.

1.2 Some Roads not Taken

1.2.1 Psychological and Linguistic Approaches

The obvious way to study language acquisition is to observe small children as they learn their first language. This has become a thriving field of research within modern psychology (see Ingram (1989) for work up to the late 80's and Tomasello and Bates (2001) for more recent work). This research has produced an impressive body of observations and theory, but it remained largely incompatible with most of the theories linguists developed for adult language. This is no major problem when studying the first stages of language acquisition in very young children, but as older children, acquiring more complex linguistic skills, become the subject of research, the problem becomes increasingly pressing. From the point of view of many linguists, the psychological study of the development of language in children has failed to deal with the true complexities of adult language which must be acquired by children (Pinker 1984).

An alternative approach to the study of language acquisition has developed within the field of linguistics, with the end point of the acquisition process, the adult language, as its starting point. Following Chomsky (1965), linguists compare different languages to identify those properties which all languages have in common. This approach has become known as *principles and parameters* (Chomsky and Lasnik 1993). The assumption is that a learning procedure only has to be defined for the idiosyncratic properties of each language while the common properties of all languages can be assumed to be innate. This has been the main program of Chomskian linguistics, which attempts to identify a *universal grammar* for all languages and a set of parameters which distinguishes different languages and must be learned.[3] Using this framework, one could then hope to be able to go back to the child development data and discover the exact way in which children actually discover the values of the parameters for the specific language they are exposed to.

Chomsky (1965) distinguished between linguistic theories which have *descriptive adequacy* and theories which also have *explanatory adequacy.* A theory with descriptive adequacy correctly describes the structures found in different languages while a theory which has explanatory adequacy must also explain how the rules used to describe the structures of each language can be deduced from examples of that language. Chomsky (p. 26) claims that "gross coverage of a large mass of data can often be attained by conflicting theories; for precisely this reason it is not, in itself, an achievement of any particular theoretical interest or importance." Therefore, only the condition of explanatory adequacy can allow us to decide between the conflicting theories. In this way, the problem of language acquisition (and specifically, the acquisition of syntax) has moved center stage in linguistic study. Even without actually achieving explanatory adequacy (something which has not yet been achieved), the need for it has been a driving force behind linguistic research, especially within the Chomskian tradition.

One could imagine the child and adult centered approaches to the study of language acquisition working towards each other, but instead, two belligerent camps have formed, with child centered approaches attempting to stretch child language

[3]Chomsky (1965) points out that the idea of a universal grammar goes at least as far back as the 18th century.

all the way up to adult language and adult centered approaches attempting to stretch adult language all the way down to child language. Comprehensive theories covering the full process of language acquisition have been developed within both approaches (see Tomasello (2003) for an example of the child centered approach and Pinker (1984) for an acquisition theory based on formal mechanisms developed for adult language). The two approaches remain largely incompatible, probably because they seem to align well with opposing stances in traditional controversies about language and human cognition in general, such as the debate about nativism vs. empiricism. Child centered approaches to language acquisition, the modern variant of which are often *usage based theories* (Langacker 1987; Langacker 1991; Tomasello 2006), tend to take an empiricist stance and emphasize the use of general cognitive capabilities of abstraction, generalization and analogy in language acquisition and use. In contrast, adult centered theories are based on some form or other of universal grammar, which is often complex and is assumed to be an innate, language specific, human capability, thus taking a nativist stance. This debate is still raging, but as it is mainly a debate about language acquisition and not about language learning, it is of no concern to the present work.

1.2.2 Theoretical Mathematical Models

Because much of what happens in the process of learning a language is invisible to us, researchers have been concerned with identifying settings which allow languages to be learned in principle (whether children actually use those methods or not). The hope is that the range of theoretical possibilities (once identified) is sufficiently constrained to allow for the correct theory to be selected based on the observed behavior of children. Linguists have been pursuing such a goal in their search for universal grammar by attempting to identify the similarities between different existing languages. As syntactic theory became increasingly formalized in twentieth century linguistics, it became possible (and tempting) to try to formalize the process by which the syntax of a particular language can be learned from example sentences of that language. Researchers in the field of *computational learning theory* who are interested in *grammatical inference* look for mathematical models which would allow them to define learnable classes of languages and algorithms for selecting one language out of such a class based on example sentences from the target language.

The seminal theoretical work in this field is Gold (1967), which defined an abstract model of the learning process and criteria for successful learning. The learner is assumed to receive one example after the other from the target language and can, at each step, make a guess as to the grammar of that language. The learner is allowed to err at first, but after a finite number of steps must converge to a correct grammar. Gold showed in his paper that if the learner assumes that the grammar can be any context free grammar and if all the learner has to go on is an arbitrary sequence of sentences in the language, then there is no algorithm which guarantees convergence to a correct grammar. In Gold's terminology, this means that the class of context free grammars is not learnable in the limit from positive examples. Because most linguists believe that a grammar has to be at least context free to allow for the phenomena observed in natural language syntax, Gold's negative result has been widely (and sometimes wildly) cited in the linguistic and cognitive literature (see Johnson (2004) for discussion). It was used, among others, to support the innateness of language or to dismiss Gold's paradigm of learning altogether (since, as we know, languages are learnable).

While Gold's theorem was for many a final statement (for good or bad), for many others (including Gold) it was only a starting point. In the decades that followed, researchers looked for variations of the original setting which could allow linguistically relevant classes of languages to be learnable. One possible variation is to change the definition of success, for example by introducing a probabilistic success criterion (e.g. Valiant 1984). Another approach is to search for specific learnable classes of grammars within the original setting of Gold's paper (e.g. Kanazawa 1998). Gold's negative results apply to large classes of grammars, but they do not necessarily hold for sub-classes of these classes. Because many languages in the classes Gold used are not plausible candidates for human languages, sub-classes of these classes, which better describe the range of possible human languages, may be learnable. Several surveys of these results are available (Lee 1996; Sakakibara 1997; de la Higuera 2005). While this line of research has created a significant body of theoretical results, its relevance to the empirical study of language acquisition remains limited because the grammatical systems considered were usually not powerful enough by the standards of modern theoretical and computational linguistics and because even when positive learnability results were achieved, they often made unrealistic assumptions about the input and resources available to the learner (such as noiseless input or very long convergence time). While some algorithms were implemented, most were not tested on real natural language data.

1.3 The Road Taken: Empirical Computational Models

Even before Gold's theoretical work and possibly also before Chomsky's introduction of explanatory adequacy as a goal for theoretical linguistics, researchers in artificial intelligence were attempting to build computer systems which could learn grammars from text (Lamb 1961). These efforts continued from the early 60's until today, ranging from simulations (working with toy grammars and languages) to systems applied to large corpora of real language. The algorithm described in the present work falls into this last class of models and has been applied to large collections of natural language sentences (see chapter 7).

While some of these computer systems (and especially the simulations) were designed to simulate the process of language acquisition by children, an important difference between this line of research and previously mentioned approaches to the study of language learning is that often the immediate motivation behind the construction of language learning computer systems is not necessarily to shed light on the problem of language acquisition by children but, instead, the need to solve some engineering problem. This is not to say that researchers developing such algorithms ignore the question of language acquisition by children, but it does mean that they do not feel committed to psychologically plausible algorithms and are driven more by the success of the algorithm on the specified task rather than its psychological modeling accuracy.

1.3.1 The Task

When it comes to syntax, the main (but not only) engineering task studied is parsing. A parser is an algorithm which takes an utterance as input and outputs the syntactic structure of that utterance. Since the syntactic structure of an utterance cannot be directly observed, different linguistic theories may assign different syntactic structures to the same utterance. The syntactic structures defined by some of these theories may be very complex, but at the most basic level syntactic structure is either described in terms of dependency links (from one word to another) or by grouping words together into syntactic units. Dependency links indicate that some relation holds between the words (such as the relation between an adjective and the noun it modifies or between a verb and its object). The grouping of words into units reflects the observation that these groups of words can function as a unit, or a *constituent*. For example, in the sentence *the dog barked*, the two words *the dog* can be replaced by a single pronoun *it*. This implies that in some ways *the dog* is a single unit. While the most obvious examples of dependencies and constituents are non-controversial, there are many cases which are debatable. Therefore, the construction of a parser always implies a choice of syntactic theory. When working with annotated corpora, one often adopts the decisions made by the annotators as to the syntactic structure. I will do so as well, but I will also discuss some of the choices made by the annotators in chapter 4.

The learning algorithms I study here are algorithms which learn to parse a language by examining unannotated example sentences. This can be referred to as *unsupervised parsing*, in contrast to *supervised parsing*, where an algorithm learns to parse a language from syntactically annotated examples.

1.3.2 Why Study Unsupervised Parsing?

Both supervised and unsupervised parsers use some form of learning to replace the traditional method of designing parsers by manually writing a set of grammar rules for each language. While using a supervised parser reduces the effort involved in writing grammar rules, it also requires a significant amount of manual labor because for each new language (and domain) to be parsed, one needs to syntactically annotate a large enough corpus of text. In an unsupervised learning approach, however, all that one needs to do is feed the learning algorithm with sufficiently many unannotated examples of the target language and domain. Since large amounts of electronic text are now available for many languages, this approach is by far the cheapest method. It is therefore appealing, from an engineering point of view, to have algorithms which can learn to parse in an unsupervised way. While such algorithms are yet to achieve parsing accuracy even close to that achieved by other methods, recent years have seen a new surge of interest in the development of unsupervised parsing algorithms, with significant improvement over past results (Klein and Manning 2002; Klein and Manning 2004; Dennis 2005; Bod 2006a; Bod 2006b).

Beyond the engineering motivation, the implementation of learning computer systems remains highly relevant to the study of language learning because it is the only approach which can test what happens when a proposed algorithm interacts with real language input. While psychological and linguistic theories are mostly based on individual examples picked out by the researchers and while mathematical models only make general theoretical assumptions about the input available to the learner, implementing computer models and testing them on large corpora of real language allows the cumulative effect of numerous examples to be studied. In this respect, this approach is the closest to studying language acquisition in its natural settings.

Even when a computational model is clearly not psychologically realistic, its success in learning syntactic structure has important implications to the study of language and language acquisition because such successful learning indicates a relation between the surface structure of a language and its hidden syntactic structure. Even if the method by which this relation is established is not actually used by children acquiring a language, the relation is still an empirical property of the language and may be used by children in some other way in the process of language acquisition.

This brings us to another reason for developing learning algorithms for syntactic structure, one which goes back to Harris (1946). In that paper, Zellig Harris proposed a "formalized procedure for describing utterances directly in terms of sequences of morphemes" which covers "an important part of what is usually included under syntax." The procedure proposed by Harris, which groups together sequences of words by the contexts in which they appear, is known as the *distributional* method and has become the starting point of many modern grammar induction algorithms. Harris himself, however, was not interested in the problem of language acquisition but rather in providing an explicit procedure for describing syntactic structure to replace "the use of diverse undefined terms and a reliance on semantic rather than formal differentiation" in the description of syntactic structure. Harris thus attempted to establish syntactic analysis as an empirical science which has sequences of words uttered by a speaker rather than the cognitive processes taking place in that speaker's mind as its subject matter. This approach, which is behaviorist in nature, lost much of its popularity (together with behaviorism in general) after the cognitive revolution of the 50's. Beyond the general shift in linguistics from the study of surface structure to the study of the cognitive processes involved in language processing, one of the reasons that linguists abandoned distributional methods in the study of grammar is their failure to achieve significant results, just as grammar induction algorithms failed for many years to achieve even modestly good results on real language input. Whether a purely distributional approach can indeed teach us anything important about the syntactic structure of language remains to be seen, but I believe that if successful algorithms can be designed to infer syntactic structure from unannotated examples then this should certainly have implications for the theory of syntax and can serve as an empirical method (which does not rely on human judgments) for discovering the syntactic structure of language. Just as one does not need to be a behaviorist to study behavior, one does not need to deny the relevance of cognition to linguistics in order to use distributional methods in the study of language. While current methods are still too weak to contribute directly to the study of syntax in the way Harris envisioned it, advances made in recent years may indicate that algorithms can discover at least some of the syntactic structure of a language. In this way, Harris's original program of coming up with a formalized procedure for describing an "important part of what is usually included under syntax" remains a valid goal for research with significant benefits to our understanding of language; if it succeeds.

1.3.3 On the Use of Meaning in Learning Syntax

Whatever the approach taken, acquisition of language by children remains relevant because the fact that children can learn language without getting explicit information about its structure means that language learning is possible, at least in principle. Of course, there is significant debate as to the exact information available to children when they acquire their language (and specifically syntax) and it is clear that a computer can never be exposed to the full experiences of the child. One central question is whether syntax can be learned independently from meaning (semantic or pragmatic). Many theories of language acquisition, such as Pinker (1984), are based on *semantic bootstrapping*, where the child first learns the meaning of some words and then uses this acquired knowledge to deduce the syntactic structure of sentences in which these words appear. While this seems to be a simple process for acquiring syntactic knowledge, it has also been shown (Landau and Gleitman 1985; Gleitman 1990) that knowledge of syntactic structure is necessary for the correct learning of the meaning of many words, when a linguistic utterance can only be mapped onto an observed situation if the syntactic structure of the utterance is known to the child. This process of *syntactic bootstrapping* can work together with semantic bootstrapping to learn both meaning and syntax. How the two may be combined and which role is played by each component remains an open question.

Implementing semantic bootstrapping in a fully formalized system has been attempted by several researchers, both theoretically, within Gold's paradigm of learning (Hamburger and Wexler 1975; Tellier 1998; Dudau-Sofronie et al. 2001; Oates et al. 2003), and in actual computer systems (Anderson 1977). The main problem with all these systems is that they do not learn the semantics as a child does, but take the semantic representation as input together with the utterance describing the situation. These semantic representations are stipulated by the designers of the algorithms, and can all be suspected of encoding syntactic information which needs to be learned by the child. The discussion whether these properties are semantic or syntactic is irrelevant, as the question remains how a child can learn them from the input available. Moving the syntax into the semantics does not solve the problem, but only avoids it. This is also the reason why computer systems designed to use semantics (such as Anderson's) were toy systems designed to prove a cognitive theory rather than systems designed towards a specific application. From an engineering point of view, to use the semantic information required by these systems would require semantically annotated corpora, so for all practical purposes it is simpler to use syntactic annotations to begin with.

Because the most readily available input for a computer program attempting to learn the syntax of a language is unannotated sentences (without any information about their meaning or context) most algorithms remain entirely distributional in nature. The objection that this is not the way children learn their language does not detract from the importance of these algorithms, if they are successful. Success of such algorithms is both useful in constructing language processing systems and in understanding the relations between the surface structure of language and its syntactic structure. This is also the approach I adopt in the present work.

1.3.4 A Brief Survey of Syntactic Induction

Over the years, many systems for learning the syntax of natural languages were proposed and implemented. This section is a brief survey of these systems and the principles used in their design. I discuss only systems which were actually implemented and which take unannotated example utterances of a language as input.

Until recently, most of these systems learned a *context free grammar*. A context free grammar (CFG) consists of a finite set of rules of the form $X \to Y_1, \ldots, Y_n$, which can be used to replace (rewrite) the symbol X by a sequence of symbols $Y_1, \ldots, Y_n$. Beginning with a single *start symbol* S, rules can be applied repeatedly until a sequence of words is formed (X cannot be a word, so words cannot be rewritten). Given a sentence to be parsed, a CFG parser looks for a sequence of rule applications which generates that sentence from S. The sequence of rule applications defines the syntactic units (constituents) of the sentence: the sequence of words which was generated from a single symbol X is a unit and X is the label of that unit (e.g. X = noun phrase). Since there may be more than one way to generate a sentence with the same CFG, the parser must have a way to select one of the possible parses. A standard way of doing so is to use a *probabilistic context free grammar* (PCFG) in which every rule is assigned a probability (the probabilities of all rules with the same left hand side must sum to 1). The rule probabilities induce a probability for each parse and the parser selects the most probable one.[4]

Most of this section is dedicated to the description of various algorithms which learn the syntax of a language by inducing a (probabilistic) context free grammar. At the end of this section I describe some more recent algorithms which do not use CFG induction but instead define various ways of inducing a parser directly. These algorithms turn out to be far more successful than the older CFG induction algorithms and I conclude the section with a short discussion of what is, in my opinion, the main reason for this difference.

Distributional Clustering

Many grammar induction algorithms use a method of *distributional clustering* which may be traced back to Harris (1946):

> The procedure [...] consists essentially of repeated substitution: e.g. *child* for *young boy* in *Where did the — go?*. To generalize this, we take a form A in an environment C —— D and then substitute another form B in place of A. If, after such substitution, we still have an expression which occurs in the language concerned, i.e. if not only CAD but also CBD occurs, we say that A and B are members of the same substitution-class, or that both A and B fill the position C —— D, or the like.

Because Harris was not interested in language induction but in structural description, he allowed the decision as to whether CAD and CBD are in the language to be taken by a linguist. When this procedure is used to induce a grammar, the decision whether A and B are members of the same substitution class is based not on the linguistic judgments of a linguist but on the occurrence of both CAD and CBD in a corpus of utterances in the language being learned. This method has been the cornerstone of many grammar induction algorithms

[4]For more detailed definitions of CFG and PCFG, see, for example, Jurafsky and Martin (2000).

beginning with Lamb (1961), and has since been used also by many others (Cook et al. 1976; Wolff 1982; Mori and Nagao 1995; Adriaans et al. 2000; van Zaanen 2000; Clark 2001; Solan et al. 2005). Some additional algorithms from the 60's and 70's based on this method are mentioned in the survey of Pinker (1979). The term *environment* used by Harris has been replaced by *context* in the more recent literature.

While the idea seems simple and straightforward, there are several fundamental problems in implementing it successfully. Several of these were already mentioned in Harris's original paper. The first problem that Harris mentions is that:

> In some languages, relatively few morphemes occur in exactly the same environments as others: *poem* occurs in *I'm writing a whole — this time* but *house* does not. Both morphemes, however, occur in *That's a beautiful —*. Shall we say that *poem* and *house* belong in general to the same substitution class, or that they have some environments in common and some not?

This problem worsens when it is not a linguist who has to decide whether a certain sentence appears in a language but a corpus is used to make such decisions. Even a large corpus contains only a small fraction of the utterances which may reasonably be produced in a language and even if two sequences of words can, in principle, appear in a certain environment, it may very well be that no evidence for this will be found in the corpus. To solve this problem, algorithms generally do not require that sequences of words appear in exactly the same contexts in order to cluster them together and some overlap of contexts is considered sufficient. The exact criterion used may vary from simple clustering of any two sequences appearing in the same context (van Zaanen 2000) to complex algorithms based on the combination of different contexts (Adriaans et al. 2000).

Part-of-Speech Induction

One task on which the clustering by context technique has proven successful is the induction of parts-of-speech, that is, the assignment of a class label to each word. This is a subtask of the general clustering task because it only considers single words (rather than sequences of words) for substitution. Using only the most frequent words in the corpus as contexts, various clustering methods have been used to induce part-of-speech tags (Schütze 1995; Clark 2000). This is not only a useful result in itself but may also serve as a first step in the induction of a grammar. For this reason, most recent syntactic induction algorithms take sequences of part-of-speech tags rather than words as their input. This considerably simplifies the problem of identifying sequences appearing in identical contexts, because both the sequences and the contexts are drawn from a much smaller set of possible symbols. In practice, the part-of-speech tag sequences are often taken from an annotated corpus rather than being induced.

Identifying Constituents

A second problem with the distributional method identified by Harris is that when *sequences* of words are considered for substitution, the substitution classes created by the method may contain sequences of words which are not constituents at all:

> Since our procedure now permits us to make any substitution of any sequences, it may become too general to produce useful results. For example, we might take the utterance *I know John was in* and substitute *certainly* for *know John*, obtaining *I certainly was in.* This substitution conceals the fact that the morphemes of *I know John was in* can be said as two utterances instead of as one.

Harris goes on to mention other respects in which *certainly* and *know John* differ and then suggests that "substitution of sequences be so carried out as to satisfy all manipulations of that environment which forms the frame of the substitution." Even if such a procedure can be carried out by a linguist, it certainly cannot be carried out by an algorithm which only has a small subset of the utterances in the language to work with and does not know how to identify all permissible manipulations of a given environment.

For this reason, some clustering-based induction algorithms (Mori and Nagao 1995; Clark 2001) explicitly define a procedure to distinguish between sequences (of part-of-speech tags) which are constituents and those which are not. Mori and Nagao make the assumption that sequences (of part-of-speech tags) which represent constituents are less constrained as to what precedes and follows them than non-constituent sequences and implement this by setting a threshold on appropriate conditional entropy functions for the right and left contexts of a sequence. Clark uses a criterion which is based on the assumption that the mutual information between the left and right context of a sequence is higher for constituents than for non-constituents. These two methods seem to implement similar intuitions in different ways. The algorithm of Solan et al. (2005) does not explicitly distinguish between constituents and non-constituents but does seem to use a method similar in nature (but not in detail) to that of Mori and Nagao (1995) in order to detect "significant patterns" which eventually become the constituents of the analysis. Other clustering algorithms do not have an explicit procedure for identifying constituents, but instead rely on the grammar rule construction procedure (described below) to implicitly prefer rules describing constituents.

Inducing the Grammar Rules

Having clustered sequences of symbols and possibly having determined which of these are candidate constituents, all sequences in a cluster can be replaced, wherever they appear in the corpus, by a new single symbol representing the cluster. This is represented by defining a set of context free rules which have the new symbol as their left hand side and the sequences in the cluster as their right hand side. Because the sequences in different clusters may overlap, replacing all occurrences in the corpus of sequences from one cluster by a single symbol can destroy the sequences which are part of another cluster. For this reason, the induction algorithm must determine which cluster to substitute first. Having performed the substitution, the process can be repeated.

Most algorithms (Cook et al. 1976; Wolff 1982; Mori and Nagao 1995; Clark 2001) use an objective function to decide which grammar rule to create at each step. These objective functions are all similar in nature (but not necessarily in detail) and may be traced back to Solomonoff (1964), who defined a Bayesian probability function which has to be maximized by the grammar induction algorithm. This probability function is $P(D|G)P(G)$, where $P(G)$ is the a-priori probability of the grammar and $P(D|G)$ is the probability of the observed data (corpus) given the grammar. The a-priori distribution is usually taken to be such that smaller grammars have higher probability. Maximizing the Bayesian probability function is equivalent to minimizing $-\log(P(D|G)) - \log(P(G))$ which is a description length criterion. The quantity $-\log(P(G))$ is seen as describing the size of the grammar and $-\log(P(D|G))$ is seen as the length of the data after being encoded by the grammar. This is often interpreted as a compression criterion because a good grammar which captures the regularities of a language should allow the data to be encoded compactly. While details vary, most algorithms use some variant of this function (either in its Bayesian or description length form) for rule selection. An exception is Solan et al. (2005), who uses the "most significant pattern" (which resembles the constituency criterion of Mori and Nagao 1995) as a criterion for substitution.

Because substituting a single symbol for a constituent immediately destroys all non-constituent sequences which overlap (but do not contain) that constituent, the process of rule selection can potentially eliminate non-constituent clusters. The burden of doing so correctly is placed on the objective function by which rule selection is determined. By filtering out non-constituent clusters before the rule selection step, Mori and Nagao (1995) and Clark (2001) increase the chances of this happening.

The algorithms of van Zaanen (2000) and Adriaans et al. (2000) are an exception to this process in that they do not substitute and re-cluster after each substitution but instead continue to use the clustering on the original text. While van Zaanen (2000) proposes different heuristics to decide between conflicting constituents in the text, Adriaans et al. (2000) simply create a set of rules from their clustering without going back to the original text (and thus do not have to deal with conflicting constituents). Of course, when parsing with these rules, only non-crossing units are created, but it is not entirely clear whether there is any mechanism in the algorithm which allows constituent clusters to be preferred over non-constituent clusters.

Syntagmatic and Paradigmatic Merging

When a clustering algorithm creates a grammar rule and substitutes all sequences in a cluster by the left hand side (non-terminal) symbol of the rule, it actually makes two decisions: first, it identifies each of the sequences as a constituent and, second, it identifies these constituents as being substitutable for each other. Borrowing structuralist terminology, some authors (Wolff 1982; Stolcke 1994) refer to these operations as syntagmatic merging (grouping words into syntactic units) and paradigmatic merging (grouping units into substitution classes). Stolcke (1994) also names them *chunking* and *merging* (respectively). In a chunk (syntagmatic merge) step, a single sequence of symbols is replaced, wherever it appears in the corpus, by a new non-terminal symbol and an appropriate context free rule is added to the grammar. In a merge (paradigmatic merge) step, several different non-terminals are merged into a single non-terminal. This approach was already applied in the algorithms of Cook et al. (1976) and Wolff (1982). Because chunking is used, one can restrict merging (clustering) to single symbols appearing in the same context, rather than having to cluster sequences of different lengths as in the original Harris method. Cook et al. make use of this and only allow merging of single symbols while Wolff seems to retain the possibility of merging sequences of symbols (in addition to chunking). Both algorithms decide which of the many possible chunking or merging operations to perform at each step based on the improvement on an objective function resulting from such an operation.

Stolcke (1994) goes one step further and does not use context at all as a criterion for merging. Instead, merging can be performed between any two non-terminals and which merge or chunk to perform depends only on the improvement on an objective function resulting from such a merge. Thus, the full burden of success is put on the shoulders of the objective function and its correct design becomes critical. It is interesting to note that Stolcke (p. 88) observes that chunking often must be combined with merging to achieve an improvement on the objective function. This seems to suggest that while the separation of merging and chunking is conceptually elegant, the two operations must be performed together and the separation is in practice undone.

Highest Likelihood PCFG

The induction algorithms mentioned so far are sometimes referred to as *structure search* algorithms, because they search for the grammar which optimizes the objective function by constructing a set of grammar rules. An alternative is *parameter search*, in which the set of possible rules is fixed and only the probability of each rule (in a probabilistic context free grammar) has to be determined. The search is then for the probability distribution which maximizes the likelihood of the observed data. This process may assign some rules a zero (or very small) probability, thus eliminating them effectively from the grammar. It is therefore possible to start with a relatively large set of possible rules and hope that the parameter search will only assign large probabilities to a small subset of them.

Of the two components of the objective function used in the structural search algorithms, we are left with only one: $P(D|G)$, the likelihood of the data given the probabilistic grammar. The a-priori probability of the grammar is no longer used. This may seem to suggest that the a-priori probability of the grammar is not needed to begin with, but this is not true. Parameter search algorithms must restrict the possible grammar rules they allow because, otherwise, the maximum likelihood is achieved by the trivial grammar in which every sentence in the corpus is generated by a single rule and the probability of the rule is equal to the relative frequency of the sentence in the corpus. The selection of the initial set of grammar rules to which a non-zero probability may be assigned becomes a critical issue in the design of parameter search algorithms. While this may be difficult to achieve in a way which is not biased towards specific languages, it is also probably not reasonable to assume that all context free grammars should remain a-priori possible, as is assumed by most structure search algorithms.

One advantage of parameter search algorithms is that a relatively efficient algorithm has been developed for finding a local maximum for the likelihood function. This algorithm, called the inside-outside algorithm (Baker 1979; Lari and Young 1990), begins with some initial setting of the rule probabilities and re-estimates these probabilities on a corpus until a local maximum of the corpus likelihood is reached. While this seems encouraging at first, attempts to induce grammars using this algorithm (Carroll and Charniak 1992) proved disappointing. One reason for failure which the authors propose is that the algorithm tends to converge to local maxima which are not good grammars. A different reason, suggested in Klein and Manning (2002), is a poor choice of the set of possible grammar rules in these experiments. Later experiments (Pereira and Schabes 1992; Schabes et al. 1993) showed that this algorithm works successfully when it is trained on bracketed sentences, but no successful application of the algorithm to the induction of PCFGs from unannotated text is known to me.

Non-CFG Syntactic Induction

Despite some slow progress, the performance of algorithms which induce a context free grammar (probabilistic or not) remains disappointing. While some algorithms were reported to successfully induce toy grammars, none seemed to succeed on the task when confronted with real linguistic data. The standard syntactic task in computational linguistics is parsing and it is therefore reasonable to evaluate grammar induction algorithms on the parsing accuracy they achieve. Even when algorithms were able to produce some output on real language input, the accuracy of the parses remained low.

In the last decade, new induction algorithms have been proposed which no longer rely on context free grammars. Instead, various probabilistic models of syntactic structure are used and induction is performed by searching for the parameters which maximize the likelihood of the corpus data. The parse assigned to a sentence is then simply the structure with the highest probability (given the induced parameters). These algorithms use either a constituency (bracketing) representation of syntactic structure (Klein and Manning 2002; Bod 2006a; Bod 2006b; Bod 2007a) or a dependency (link) representation of syntactic structure (Yuret 1998; Paskin 2002; Klein and Manning 2004; Smith and Eisner 2005; Smith and Eisner 2006).

The probability assigned by these models to a syntactic structure is based on the product of the probabilities assigned to the "building blocks" of the structure. In the case of CCM (Klein and Manning 2002), these building blocks are the constituent and non-constituent sequences of parts-of-speech in the structure as well as the contexts of these sequences. The probability distributions induced by the algorithm then specify the probability of a certain sequence of parts-of-speech (or context) as a constituent or a non-constituent (see figure 1.1 for details). In the case of the different variants of U-DOP (Bod 2006a; Bod 2006b; Bod 2007a), the building blocks are subtrees of the syntactic structure and the probabilities are the probability of using each subtree in a derivation (see figure 1.2 for details). Both these models require the syntactic trees to be binary branching. When dependency models are used, the building blocks are the dependency links and the probability distribution describes the probability of two parts-of-speech (or words, in the case of Yuret 1998) being joined by a link. In DMV, Klein and Manning (2004) also added a probability describing the non-attachment of a head beyond its last argument (see figure 1.1 for details). This model has also been used by Smith and Eisner (2005) and Smith and Eisner (2006) with different likelihood maximization techniques.

Many of these recent algorithms perform significantly better than context free grammar induction algorithms. While no CFG induction algorithm has ever been reported to do better on English than the right-branching heuristic (which simply brackets every word together with all words to its right), many recent algorithms (Klein and Manning 2002; Klein and Manning 2004; Smith and Eisner 2005;

CCM:

S - sentence (part-of-speech sequence): ${}_0\, NN\, {}_1\, NNS\, {}_2\, VBD\, {}_3\, IN\, {}_4\, NN\, {}_5$

B - bracketing (boolean matrix): $[\,[\,{}_0\, NN\, {}_1\, NNS\,]\, {}_2\, [\, VBD\, {}_3\, [\, IN\, {}_4\, NN\, {}_5\,]\,]\,]$

$$B_{ij} = true \iff \text{bracket from } i \text{ to } j\text{:} \quad \begin{array}{cccccc|c} 0 & 1 & 2 & 3 & 4 & 5 & \\ & t & t & & & t & 0 \\ & & t & & & & 1 \\ & & & t & & t & 2 \\ & & & & t & t & 3 \\ & & & & & t & 4 \\ & & & & & & 5 \end{array}$$

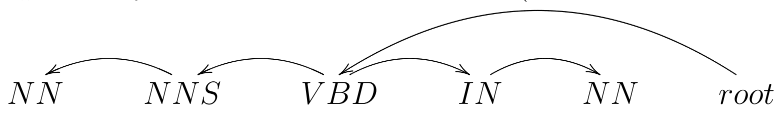
α_{ij} - parts-of-speech from i to j (e.g. $\alpha_{02} = NN\ NNS$).
x_{ij} - the context of α_{ij} (e.g. $x_{02} = \diamond - VBD$).

CCM defines a probabilistic model $P(S,B) = P_{bin}(B)P(S|B)$ with P_{bin} a uniform distribution over all binary branching bracketings and

$$P(S|B) = \prod_{i<j} P(\alpha_{ij}|B_{ij})P(x_{ij}|B_{ij})$$

DMV:

Projective dependency structure D of sentence S (see section 4.1 for definitions):



Each dependency d is a link from a head h to a dependent a.

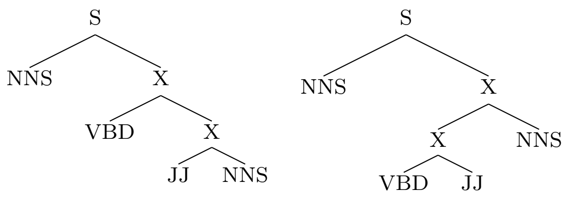
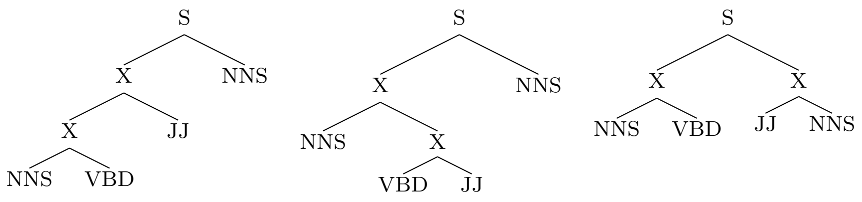
DMV defines the following generative probabilistic model for $P(D,S)$:

$D(h)$ - dependency structure rooted at h ($D = D(root)$).
$deps_D(h, l/r)$ - dependents of h (in D) to the left/right of h.
$adj = true$ iff no dependent has yet been generated in the current direction.

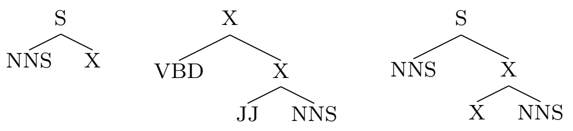
$$P(D(h)) = \prod_{dir \in \{l,r\}} \left(\prod_{a \in deps_D(h,dir)} P_{\mathrm{STOP}}(\neg\mathrm{STOP}|h, dir, adj)\ P(a|h, dir)P(D(a)) \right) \times P_{\mathrm{STOP}}(\mathrm{STOP}|h, dir, adj)$$

Figure 1.1: Klein and Manning's CCM (2002) and DMV (2004) models. The EM algorithm (with the sentences S as observed and bracketing B or dependencies D as unobserved) is used to search for the model parameters which (locally) maximize the likelihood of the (unannotated) corpus. Each sentence is assigned the most probable structure (bracketing/dependency) according to these parameters.

Every sentence (part-of-speech sequence) in the input corpus is assigned all possible binary trees:

All subtrees are extracted:



etc.

Each subtree t in this collection is assigned a probability:

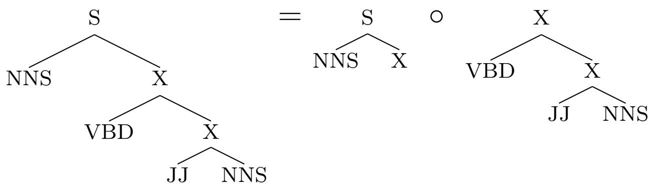
U-DOP:
$$P(t) = \frac{|t|}{\sum_{t':r(t')=r(t)} |t'|}$$

where $r(t)$ is the root node of t (S or X) and $|t|$ is the number of times t appears in the subtree collection.

UML-DOP: Expectation maximization beginning with U-DOP's estimates.

U-DOP*: Using the DOP* estimator of Zollmann and Sima'an (2005).

A derivation constructs a tree from subtrees:



The probability of a derivation is the product of the probabilities of the subtrees it uses: $P(t_1 \circ \ldots \circ t_n) = \prod_i P(t_i)$. The probability of a tree is the sum of probabilities of all its possible derivations: $P(T) = \sum_{\{t_1 \circ \ldots \circ t_n = T\}} \prod_i P(t_i)$. In practice, only the most probable derivations are summed.

The parse assigned to a sentence is the tree with the highest probability.

Figure 1.2: Bod's U-DOP (Bod 2006b), UML-DOP (Bod 2006a) and U-DOP* (Bod 2007a) algorithms.

Smith and Eisner 2006; Bod 2006a; Bod 2006b; Bod 2007a) do significantly better than this baseline (see chapter 7 for details).

1.3.5 From Grammar Induction to Parser Induction

The move away from context free grammars has significantly improved the parsing accuracy of unsupervised parsers. While some of this improvement can be attributed to the details of the design or to the use of the expectation maximization technique, I would like to suggest that it is the move from *grammar induction* to *parser induction* which has contributed most to the improvement.

We have seen that the construction of a context free grammar requires two types of decisions to be made: syntagmatic (which sequences of words are constituents) and paradigmatic (which sequences can be substituted for each other). These are two aspects inherent to what we expect from any grammar and neither can be ignored in the process of induction. In contrast, unlabeled parsing, which only requires the parser to identify the constituents (or dependency links) but does not require them to be labeled, is purely syntagmatic (by definition). A parser induction algorithm can therefore focus on learning to detect syntactic units while ignoring substitutability. Indeed, none of the recent successful algorithms (Klein and Manning 2002; Klein and Manning 2004; Bod 2006a; Bod 2006b; Bod 2007a) can determine which constituents are substitutable. Even when contexts are used (as in the CCM algorithm of Klein and Manning 2002) they are only used to determine the probability that the sequence appearing inside the context is a constituent and not to decide which sequences can be substituted for each other. Another example is the memory based algorithm of Dennis (2005), which uses alignment just as in older clustering algorithms but stops short of creating substitution classes. Instead, it directly uses the alignments to make parsing decisions.

In contrast to these parser induction algorithms, grammar induction algorithms need to perform both syntagmatic and paradigmatic induction. In practice, the emphasis was always on the paradigmatic aspect of the induction. This is implied in Stolcke's (1994) comment that syntagmatic merges must usually be followed by paradigmatic merges to produce any improvement on his objective function. This shows that while formally both syntagmatic and paradigmatic relations are learned, it is only the paradigmatic relation which is the driving force behind the induction process. It is not surprising therefore that such algorithms produce poor parsers. The few grammar induction algorithms that did incorporate some explicit mechanism to distinguish constituents from non-constituents (Mori and Nagao 1995; Clark 2001) seem to have gained in parsing accuracy from this. Still, it was only when the focus shifted completely from substitutability to the detection of constituents that parsing accuracy began to improve significantly. Substitutability, the essential idea of the Harris method, which has been seen as a starting point for the induction process for so long, turns out to be unnecessary in unsupervised parsing.

This does not mean that substitutability is not an important linguistic notion or that grammars are not an important linguistic tool (in generating new sentences, for example) but it does mean that the first step in the learning of syntax, the discovery of the structure of the utterances, can be done without them. Substitutability can then be learned based on this syntactic structure rather than being used to determine it.

This having been said, the notion of substitutability still plays one important role in recent unsupervised parsing algorithms: they all use part-of-speech sequences in place of words as their input (with the exception of Yuret 1998). One may wonder whether this is necessary. In this thesis I suggest that the answer is probably no and I present an unsupervised parser which completely does away with substitutability, even at the word level.

1.4 The Road Taken: Learning to Parse Incrementally

The present thesis is about parser induction. It takes the view that the identification of syntactic units and relations in an utterance does not require the notion of substitution or the definition of a grammar. It makes this explicit by defining a non-deterministic parser and learning a *parsing function* which decides among the various parsing options open to the non-deterministic parser.

When designing an unsupervised parser, it is useful to look at the way humans process language even if one is not interested in cognitive modeling and it is useful to look at the common properties of languages even if one is not looking for a universal grammar. Of the many properties of language and language processing discovered by researchers, I have chosen to make primary use of three: the incrementality of human language processing, the skewness of syntactic tree structures and the Zipfian distribution of words. All these are fundamental and universally accepted properties of language. The use of these properties leads to a greedy parser in which both parsing and learning are local. As a result, learning and parsing are fast, but not at the expense of parsing accuracy, which remains high by current unsupervised parsing standards.

1.4.1 Incrementality

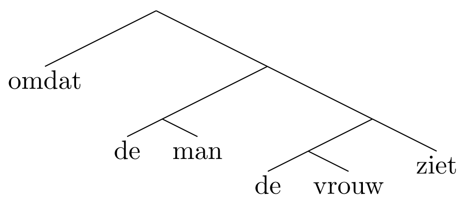
Humans interpret language as it is being heard or read, and do not have to wait for the end of an utterance to determine the structure and meaning of its beginning. This is referred to as the incrementality of human language processing, and has been thoroughly studied by psycholinguists (see e.g. Crocker et al. 2000). While incrementality is widely acknowledged to be a property of human language processing, most grammars are not specifically designed to be applied incrementally and most standard parsers are not incremental. Even in a grammatical framework such as combinatory categorial grammar (Steedman 2000), which is supposed to easily accommodate incremental parsing, wide coverage parsers (Hockenmaier and Steedman 2002; Clark and Curran 2004) are not incremental. Thus, in computational linguistics, incrementality is usually seen as an additional burden on the design of a system rather than as a useful tool in its development. But incrementality can be most useful, because it considerably constrains the possibilities a language interpreter has to consider (Church 1980). In the specific case discussed in the present work, this interpreter is a parser which has to determine the syntactic structure of an utterance. While in most standard parsing algorithms the end of the utterance can potentially affect the parse of the beginning of the utterance, this cannot happen in an incremental parser. As a result, an incremental parser has fewer possibilities to consider at each step. This not only restricts the search space for the parser but also simplifies the task of the learning algorithm because the learning algorithm only has to learn to distinguish between the possibilities the parser may choose from.

The problem encountered by incremental parsers is that in some utterances the structure of the beginning of the utterance remains ambiguous until a disambiguating word is reached. This seems to be a problem for incremental parsing, but is actually dependent on the syntactic representation chosen: a structure which is ambiguous in one representation is not necessarily ambiguous in another representation, which may leave the ambiguous feature underspecified until the disambiguating word is reached. Not every ambiguity may be solved in this way and linguists have long been aware of the fact that humans can easily handle some ambiguities while having problems processing others (Bever 1970). Psycholinguists have developed various explanations for this difference between ambiguities and some of these proposals are representational in nature: only the difficult ambiguities are ambiguous in the proposed representations (Weinberg 1993; Weinberg 1995; Gorrell 1995a; Gorrell 1995b; Sturt and Crocker 1996). This will be discussed in section 4.3. In the present work I adopt a similar approach and develop a new link based representation of syntactic structure which is well suited for incremental parsing.

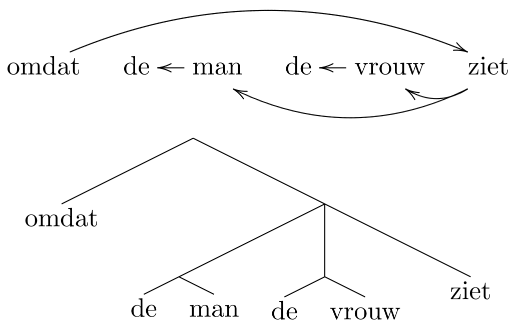
1.4.2 Skewness

The syntactic structure of natural language is skewed. This simply means that when the syntactic structure of an utterance is represented by a tree, each node in the tree has at least one short branch (figure 1.3a). The shorter the shortest branch is, the greater the skewness. In chapter 4, I examine several syntactically annotated corpora to show that a significant degree of skewness can be found in those annotations. The syntactic representation I introduce in this thesis easily captures this skewness.

In contrast, phrase based representations of syntactic structure, such as con-



(a) This syntactic tree of a Dutch phrase is skewed because under every node there is a branch of length at most 2. The shortest branch is sometimes on the left and sometimes on the right.



(b) When the syntactic tree (bottom) is derived from the dependency structure (top) by creating a node for every head and all its direct and indirect dependents, the tree is even more skewed because under every node there is a branch of length 1.

Figure 1.3: An example of skewed syntactic trees.

text free grammars, allow (a-priori) any tree structure and, therefore, a learning algorithm for such representations must discover by itself the skewness property of syntactic trees. However, if this property is indeed universal, there is no need to burden the learning algorithm with its discovery and it is possible to code skewness directly into the parser.

The other extreme is taken by dependency structures (see section 4.1), in which a head word is connected by links to all its dependents (which may, in turn, be heads of other dependents). The straightforward way to construct constituents from a dependency structure is to create for each head word a constituent covering it together with all its direct and indirect dependents (figure 1.3b). The resulting tree is skewed because every head word is attached immediately under the node it heads. This skewness is too strong, however, especially for sentential constructions that combine a subject with a predicate (see the example in figure 1.3 and chapter 4 for details). Therefore, the skewness defined by dependency structures must be relaxed.

The syntactic representation I introduce here is based on links between words, and can easily capture the skewness of syntactic structure in a way similar to that of dependency structures. However, by labeling each link by a number (its *depth*) the representation allows the degree of skewness to be lower than that of dependency structures. I will argue in chapter 4 that the resulting skewness is close to that which is actually observed in natural language.

1.4.3 The Incremental Parser

Having defined a representation for syntactic structure, the next step is to define an efficient parser for that representation, that is, an algorithm which takes an utterance as input and outputs the syntactic structure of that utterance. While the syntactic representation I use was chosen to facilitate incremental parsing, it is the parser I describe which actually implements this incrementality. By doing so, it also defines an exact notion of incrementality (since there are multiple ways of doing so). From now on, I will refer to this simply as the *incremental parser*.

The syntactic formalism used by the parser ensures that the parser can only output skewed syntactic structures, thus eliminating many spurious candidate structures from the search space. The incrementality of the parser further restricts the search space, thus simplifying parsing even further. If the incrementality and skewness coded in the syntactic representation and parser roughly resemble those of natural languages then this reduction of the search space should not come at the expense of the accuracy of the parser.

The basic incremental parsing algorithm is non-deterministic: at each step it specifies a set of links which may be added to the parse, but does not determine which of these links to add. This is not surprising, since different languages require different parsing decisions to be made. Classically, such idiosyncratic properties of a language are coded for the parser by a grammar of the language. In the case of the incremental parser, this is replaced by a *parsing function* which selects, at each step, one of the options available to the parser. It is the parsing function which has to be learned by the induction process. The learning process is simplified if the parsing function only needs to code the idiosyncratic properties of a language and not the universal properties of language parsing. In the present work, skewness and incrementality were coded as universal properties.

1.4.4 Learning and the Zipfian Distribution

To learn the parsing function, the algorithm I present here makes use of the Zipfian distribution of words. Zipf's law states that words in a language obey a power law probability distribution, which roughly means that there is a small number of words which are very frequent and many words which are extremely infrequent. This has often been seen as a curse in computational linguistics, because it means that many words are too infrequent to collect meaningful statistics for. I suggest, however, that one should not see the glass as half empty, but as half full: a relatively small number of frequent words appears almost everywhere and most words are never too far from such a frequent word. The frequent words can therefore guide the parsing and learning process. This is also the principle behind successful part-of-speech induction.

The Zipfian distribution is a property of words, not of parts-of-speech (which cluster many infrequent words, such as nouns, under a single tag). Therefore, in contrast to most modern syntactic induction algorithms, it is not only possible but also desirable to use the algorithm I present here directly on words and not on part-of-speech sequences. No clustering is performed at any level and the algorithm works entirely locally. Instead of using parts-of-speech, the algorithm labels each side of each word by its neighbors in the text and, recursively, by the labels of these neighbors. Parsing is then directly guided by these labels. Due to the Zipfian distribution of words, high frequency words dominate the lists of labels and parsing decisions for words of similar distribution are guided by the same labels. This not only simplifies the induction process, but also allows much greater flexibility, since the exact label used at each parse step may depend on the parsing context. In addition, the labels on the left and right side of each word may remain independent.

1.4.5 Bootstrapping

The final ingredient in the learning process is bootstrapping. The learning process is nothing more than a simple process of collecting statistics which result from the parsing process: as an utterance is parsed, the parse determines for the learning process which statistics to collect (a somewhat similar idea can be found in Yuret 1998). The statistics of each word are simply collected from the properties of words which are adjacent to it according to the parse. The notion of adjacency depends on the parse assigned to the utterance and will play a central role in the algorithm.

Because learning is merely the collection of statistics resulting from parsing, the learning process is open-ended and additional training text can always be added without having to re-run the learner on previous training data. Learning does not slow parsing much and experiments show that parsing (which is at the rate of thousands of words per second) is slowed down by about 20% when learning is turned on. This means that, potentially, learning can always remain turned on. This is appealing both for engineering purposes and for cognitive modeling.

One risk of using a bootstrapping process, where learning is influenced by what has been learned before, is that incorrect conclusions reached at the beginning of the learning process reinforce themselves through bootstrapping and cannot be gotten rid of. This is similar in some respects to the problem of search algorithms getting stuck at local minima. I will argue (section 6.2.2) that the learning algorithm I propose does not have this problem.

1.5 Organization of this Thesis

The parsing and learning algorithms are described in chapters 2, 3 and 6. Chapter 2 introduces the basic definitions of *common cover links*, the syntactic representation being used, and some of their main properties. The main algorithm in this chapter (Algorithm 2.6.5) converts common cover link structures into equivalent bracketings. This can be done incrementally, in parallel with parsing and allows the output of the parser to be compared with standard annotation.

Chapter 3 introduces the non-deterministic incremental parsing algorithm (Algorithm 3.3.1) and proves that it can indeed construct every bracketing incrementally. Next, parsing functions are introduced (Definition 3.4.1) and these functions are used to define a deterministic parsing algorithm (Algorithm 3.4.2).

Chapter 6 completes the description of the algorithm. It describes a framework for inducing a parsing function based on a family of greedy parsing functions (Definition 6.1.3). The learning process selects one of the functions in this family based on a statistics update algorithm (Algorithm 6.2.2). This framework leaves some aspects of the algorithm unspecified and section 6.3 specifies a simple instantiation of this framework, given by a lexical update algorithm for learning (Algorithm 6.3.1) and a weight function (section 6.3.2) for the parsing functions.

Chapters 4 and 5 describe the syntactic representation and parser in more detail. Chapter 4 discusses the linguistic properties of the common cover link representation and of the incremental parser. It also discusses in detail the skewness of syntactic structure. Chapter 5 details all the mathematical properties of the representation and the incremental parser and proves all claims made in previous chapters. The chapter is technical and can be skipped in first reading. It was written to be self-contained, so statements (such as definitions, claims and algorithms) given in previous chapters are repeated in this chapter. To make it easier to locate these statements, they are assigned a number in each chapter in which they appear and both numbers are indicated when the statement is made.

Finally, chapter 7 reports on experiments conducted with the algorithm on several real language corpora.

A short description of some of the main contributions of this work was previously published in Seginer (2007).

Chapter 2

Common Cover Links

This chapter is about syntactic structure at its most basic level, that which is usually represented by unlabeled bracketing (constituency structure) or unlabeled dependencies. It is about the representation of the structure without specifying any mechanism (such as a grammar) for generating it. While this structure is probably not a complete syntactic description, it is part of almost any syntactic theory. Moreover, many basic linguistic tests can be used to detect the existence of a constituent or a dependency without naming it. The naming of the detected structure is then often more theory dependent than the detection itself. Therefore, unlabeled syntactic structure is a structure in its own right. This is the structure I will be looking at here.

Almost all linguistic theories use either bracketing or dependencies to describe this basic syntactic structure. The two representations are similar, but not equivalent. The goal of this chapter is to introduce a new representation which shares some of the advantages of both bracketing and dependencies but also has additional properties not shared by either. The differences are small, but significant. Subsequent chapters will show how these differences allow incremental parsing and aid in learning of the structure from unannotated example utterances.

After giving a quick informal preview of the syntactic representation which will be used and the way it is used in parsing, I formally define the representation. This representation, called *common cover links*, is based on directed links between words and is therefore similar (but not identical) to dependency structure. The chapter begins by defining the common cover links assigned to an utterance based on the bracketing (constituency) structure of that utterance. It is then shown that certain subsets of the common cover links (called *shortest common cover link sets*) contain all the information needed to reconstruct the bracketing on which they are based. Because the construction of a shortest common cover link set from the bracketing may involve some free choices, the shortest common cover link set representation is actually slightly more expressive than bracketing. The relation between shortest common cover link sets and dependencies will be evident in the examples given and will be discussed in detail in chapter 4.

Every common cover link has a depth. An empirical observation central to the present work is that links of depth 0 and 1 seem sufficient to describe the syntactic structures which appear in natural language. This is a simple and succinct description of the property usually referred to as the skewness of syntactic trees: every syntactic sub-tree of natural language has at least one short branch. Restricting the depths of the common cover links to 0 and 1 is a simple way of building this property into the syntactic model. Many structures which are not possible syntactic structures are thus removed immediately from the set of structures which should be considered when parsing and learning. As a result, both parsing and learning are greatly simplified. Except for the beginning of the present chapter, where the general case with no restrictions on the depths of links is discussed, I will concentrate throughout the rest of this work only on common cover links with depth 0 or 1. This simplifies the analysis and keeps it focused on the linguistically relevant structures.

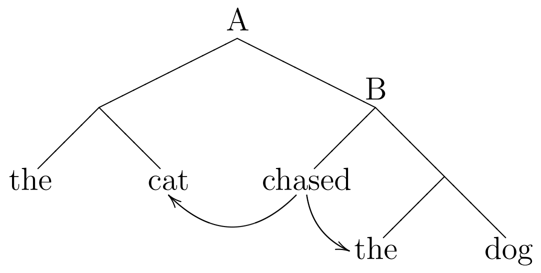
Using bracketing to define the shortest common cover link sets means that the definition cannot be used to construct shortest common cover link sets when the bracketing is not known. To be able to use common cover links directly for parsing, a characterization of shortest common cover link sets which is based only on the relations between the links is needed. Such a characterization is given in the last part of this chapter. Since this characterization does not make any use of bracketing, it makes the shortest common cover link sets an independent representation of syntactic structure and allows them to be used directly in parsing (chapter 3). This characterization is used as the *definition* of the shortest common cover link sets in all subsequent chapters. In fact, the original definition based on bracketing was only given to motivate the choices made in the characterization of the shortest common cover link sets.

One of the important consequences of the characterization of shortest common cover link sets is that the restriction of a shortest common cover link set to a segment of the utterance is itself a shortest common cover link set for that segment. This allows common cover link structures to be defined for incomplete utterances (which are not uncommon in actual language use) and allows incremental parsing (chapter 3).

This chapter only gives the main mathematical properties of the common cover links which are relevant for parsing and linguistic processing. This is accompanied by some linguistic examples, but both thorough linguistic and mathematical analysis are postponed until after the common cover link parser is described in chapter 3. Chapter 4 then discusses the linguistic aspects of the system and chapter 5 provides the full mathematical analysis, including proofs of all claims made in the present chapter.

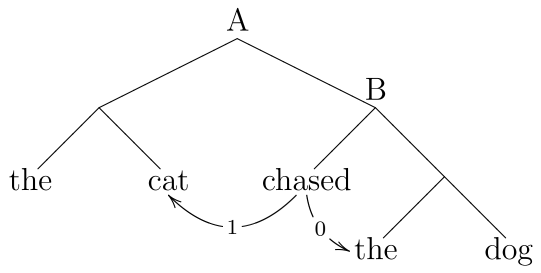
2.1 A Quick Preview

In a tree, each subtree has words (leaf nodes) which are highest up in the subtree. In the present work, I define *common cover links* to point from a highest leaf node in a subtree to other leaf nodes in the same subtree. For example, in the following tree structure, the word *chased* is one of three words which are highest in the tree rooted at A and the only word which is highest in the subtree rooted at B. The diagram shows two (out of several) common cover links which are induced by the tree structure:



The relation described by common cover links plays an important role in many linguistic theories. For example, it is closely related to the property of *c-command*, which is central to Government and Binding theory (Haegeman 1994). It is also a weaker version of the head-dependent relation in dependency theories: while dependency describes a relation of direct domination between words, the common cover link relation is close to the relation of domination, the transitive closure of direct domination. The relationship between common cover links and dependency structures is the subject of chapter 4.

In the present work I use the common cover links not merely as a property derived from tree structures but as a means of fully describing these structures. To be able to reconstruct the original tree from the common cover links, it is important to know how deep the source node of the link is in the subtree for which it was created. This is the depth of the link. It is not important, however, to know how deep the target of the link is (this can be discovered from other links). Continuing the previous example, the depth of the link from *chased* to *cat* is larger than the depth of the link from *chased* to *the* because *chased* is deeper down in the tree rooted at A than in the subtree rooted at B:



A bracketing and a tree structure are two equivalent ways of representing the structure of a sentence. This example can, therefore, also be given in bracketing notation (which will be used from now on):

$$\begin{tikzcd}\lbrack\,\lbrack\,\text{the} & \text{cat}\,\rbrack & \lbrack\,\text{chased} \arrow[l, bend right, "1"'] \arrow[r, bend left, "0"] & \lbrack\,\text{the} & \text{dog}\,\rbrack\,\rbrack\,\rbrack\end{tikzcd}$$

Common cover links are so named because a link from x to y of depth d means that every bracket, except for d brackets, which covers x must also cover y. Let us consider a simple link structure:

$$x \xrightarrow{0} y \xrightarrow{0} z$$

These links have a simple transitivity property: if every bracket which covers x also covers y and every bracket which covers y also covers z, then every bracket which covers x covers z. This means that the only bracket which covers x must also cover y and z. If there are no additional links, the resulting bracketing structure is:

$$[\ x \xrightarrow{0} [\ y \xrightarrow{0} z\ ]\ ]$$

It can be seen here that the bracket which covers x, y and z is not created by a single link but is the result of combining the two links. In fact, the link $x \overset{0}{\to} y$ does not, in itself, determine any single bracket. At the same time, each common cover link may depend on several brackets.

Parsing with common cover links (chapter 3) is a greedy process and links are added one by one and (roughly) from left to right. In the above example, when the parser first adds the link $x \overset{0}{\to} y$, it makes a decision which is only part of what will eventually become the decision to create a bracket covering x, y and z. The way a common cover link parser constructs this bracket is very different from the way this is done in phrase based parsing. While a phrase based parser must first create a bracket covering y and z (if it is bottom-up) or hypothesize such a bracket (if it is top-down), the common cover link parser first determines that the bracket covering x must also cover y and only afterwards determines the structure of the subtree y is part of. In this way, the incremental (left to right) common cover link parser splits the parsing problem into different steps than phrase based parsers (including incremental parsers such as shift-reduce parsers).

An additional example shows even more clearly the difference between common cover link parsing and phrase based parsing. The common cover link parser may create the following structure:

$$x \xrightarrow{0} y \xleftarrow{0} z$$

The two links imply that the parser has determined that there is a bracket which covers x and y and that there is a bracket which covers z and y. This, however, leaves several possibilities open, including $[\ [\ x\ y\ ]\ z\ ]$ and $[\ x\ [\ y\ z\ ]\ ]$. To decide between these possibilities, the parser needs to add additional links. For example, adding a link from x to z results in the following bracketing:

$$\begin{tikzcd}\lbrack\ x \arrow[r, "0"] \arrow[rr, bend left, "0"] & \lbrack\ y & z\ \rbrack\ \rbrack \arrow[l, "0"']\end{tikzcd}$$

As in the previous parsing example, the way parsing is split into steps in this case is very different from the way in which a phrase based parser can split the problem. In the present work I will argue that this offers several important advantages.

2.2 Basic Definitions

I begin with several basic definitions. Let W be a finite set of *word types*. An *utterance* is a sequence of words $U = \langle x_1, \ldots, x_n \rangle$ in W. The order of the words in the sequence is the order in which the words are uttered. I will use the (culturally biased) convention of saying that a word x is to the left of word y if x precedes y in the utterance. Since the sequence $\langle x_1, \ldots, x_n \rangle$ is a function from $\{1, \ldots, n\}$ to W, the utterance U is equal to the set $\{\langle i, x_i \rangle\}_{i=1}^{n}$. Each element $\langle i, x_i \rangle$ in this set is a *word token*. When no confusion can arise, I will refer to both word types and word tokens as words. Any sub-sequence $\langle x_i, \ldots, x_j \rangle$ of consecutive words ($1 \leq i \leq j \leq n$) is a *bracket* over the utterance. I will use lower-case letters (x, y, …) to denote word tokens and upper-case letters (X, Y, …) to denote brackets. I will treat brackets as sets of word tokens. The bracket Y *covers* the bracket X if $X \subset Y$ (strict inclusion) as sets of word tokens. Similarly, the bracket X *covers* the word token x if $x \in X$. I will write $[x, y]$ for the bracket covering all word tokens between x and y including x and y. I will also write $[x, y)$ (or $(y, x]$) for $[x, y] \setminus \{y\}$ and (x, y) for $[x, y] \setminus \{x, y\}$. The order of x and y is not important, so $[x, y) = (y, x]$. Two brackets X and Y are *non-crossing brackets* if either $X \cap Y = \emptyset$, $X \subseteq Y$ or $Y \subseteq X$.

2.2.1 (5.1.1). Definition. [bracketing] A bracketing of an utterance is a set of non-crossing brackets over that utterance such that every word in the utterance is covered by at least one bracket.

2.2.2 (5.1.2). Definition. [depth in a bracketing] Let $\mathcal{C}$ be a bracketing over an utterance U. The word x is of *depth* d under B in $\mathcal{C}$ if $x \in B \in \mathcal{C}$ and d is the maximal number of brackets $X_1, \ldots, X_d \in \mathcal{C}$ such that $x \in X_1 \subset \ldots \subset X_d \subset B$. In particular, if $x \in B \in \mathcal{C}$ and there is no bracket $X \in \mathcal{C}$ such that $x \in X \subset B$ then the depth of x under B in $\mathcal{C}$ is zero.

Notation I write $d_B^{\mathcal{C}}(x)$ for the depth of x under B in $\mathcal{C}$. When the bracketing is fixed by the context, I will simply write $d_B(x)$.

Notation I write $B_d^{\mathcal{C}}(x)$ for the unique bracket $B \in \mathcal{C}$ such that x is of depth d under B in $\mathcal{C}$, if such a bracket exists. Whenever the bracketing is determined unambiguously by the context I will simply write $B_d(x)$ instead of $B_d^{\mathcal{C}}(x)$.

Because the brackets in a bracketing $\mathcal{C}$ are non-crossing, a word x cannot be of the same depth under two different brackets in $\mathcal{C}$ and therefore $B_d^{\mathcal{C}}(x)$ is unique. Clearly, $B_d^{\mathcal{C}}(x)$ does not always exist but, since the definition of a bracketing requires that every word in the utterance be covered by at least one bracket, it follows that $B_0^{\mathcal{C}}(x)$ always exists. This is the smallest bracket covering x in $\mathcal{C}$.

Different words covered by a bracket B may have different depths under B. The words of minimal depth under B play an important role in the definition of common cover links. This is related to the fact that the head of a linguistic phrase is of minimal depth under the bracket representing the phrase (see more in chapter 4). Because not every word of minimal depth is also a linguistic head and because linguistic theories differ in the words they identify as heads (chapter 4) I avoid here using the term *head* and instead use the neutral term *minimal depth*:

2.2.3 (5.1.3). DEFINITION. [word of minimal depth] Let $\mathcal{C}$ be a bracketing. A word x is of minimal depth under $B \in \mathcal{C}$ if $x \in B$ and for every $y \in B$, $d_B^{\mathcal{C}}(x) \leq d_B^{\mathcal{C}}(y)$.

A bracket B may cover several words which are of minimal depth under B. All these words must, of course, have the same depth under B. This depth is therefore a characteristic of the bracket, which I will refer to as the *height* of the bracket B:

2.2.4 (5.1.4). DEFINITION. [bracket height] Let $\mathcal{C}$ be a bracketing and $B \in \mathcal{C}$. The *height* of B in $\mathcal{C}$ is $\min_{x \in B} d_B^{\mathcal{C}}(x)$.

As will turn out later on, one of the important characteristics of syntactic structures is that the heights of all brackets are restricted to either 0 or 1.

2.3 Common Cover Links

The common cover links provide a link representation of the syntactic structure of utterances. This representation should contain at least all the information which is contained in the constituency structure (bracketing) of an utterance. Therefore, a common cover link representation of an utterance should determine a unique bracketing of the utterance. Moreover, it should be possible to represent any bracketing in this way. To represent a bracket B, common cover links are defined from each word x of minimal depth under B to all other words in B. The word x may be of minimal depth under different brackets, but for each such bracket this depth must be different. To distinguish the links belonging to these different brackets, the links are assigned a depth.

2.3.1 (5.1.5). Definition. [common cover link] A *common cover link* of depth d over an utterance U is a triple $(x, y, d) \in U^2 \times (\mathbb{N} \cup \{0\})$ where $x \neq y$. I write $x \xrightarrow{d} y$ for the link (x, y, d), and the link is said to be from x to y. The word x is the *base* of the link and y is its *head*.

Notation Back and forth links between two words play an important role in the theory which will be presented here. Moreover, Lemma 5.1.14 shows that, in all common cover link sets which will be used to describe linguistic structures, back and forth links between two words must be of equal depth. I will therefore write $x \overset{d}{\rightleftarrows} y \in L$ to indicate that for a set of common cover links L, both $x \xrightarrow{d} y \in L$ and $y \xrightarrow{d} x \in L$. When describing a set L of common cover links it is often convenient to be able to indicate whether L contains any common cover link, of whatever depth, between two given words. I therefore write $x \rightarrow y \in L$ to indicate that there exists some d such that $x \xrightarrow{d} y \in L$. Similarly, I write $x \rightarrow y \notin L$ to indicate that there does not exist any d such that $x \xrightarrow{d} y \in L$.

2.3.2 (5.1.6). Definition. [common cover links of a bracketing ($R_\mathcal{C}$)] Let $\mathcal{C}$ be a bracketing over an utterance U. The set $R_\mathcal{C}$ of common cover links for $\mathcal{C}$ is the set of common cover links over U such that $x \xrightarrow{d} y \in R_\mathcal{C}$ iff x is of minimal depth d under the smallest bracket $B \in \mathcal{C}$ such that $x, y \in B$.

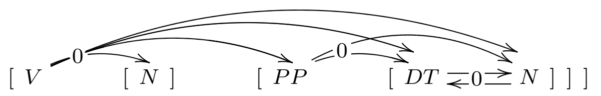
Let $B_0^\mathcal{C}(x), \ldots, B_n^\mathcal{C}(x)$ be the brackets under which x is of minimal depth. From the definition of $\mathcal{C}$ it follows that there are links of depth 0 from x to all other words in $B_0^\mathcal{C}(x)$ and links of depth $d > 0$ from x to all words in $B_d^\mathcal{C}(x) \setminus B_{d-1}^\mathcal{C}(x)$:

$$\begin{tikzcd}
{[\, x_{i_1}} & \ldots & x_{i_2} & {[\ldots} & x \arrow[llll, bend right, "d"'] \arrow[ll, bend right, "d"'] \arrow[rr, bend left, "d"] \arrow[rrrr, bend left, "d"] & {\ldots]_{B_{d-1}(x)}} & x_{i_3} & \ldots & {x_{i_4}\,]_{B_d(x)}}
\end{tikzcd}$$

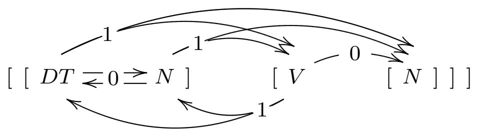
The following example shows the full common cover link set $R_\mathcal{C}$ for a simple bracketing $\mathcal{C}$:

$$\begin{tikzcd}
{[\;\; [\, w\,]} \arrow[r, bend right, "1"'] \arrow[rr, bend right, "1"'] \arrow[rrr, bend right, "1"'] & {[\, x} \arrow[l, bend right, "1"'] \arrow[r, bend left, "0"] \arrow[rr, bend left, "0"] & {[\, y} \arrow[r, shift left, "0"] & {z\,] \;\;] \;\;]} \arrow[l, shift left]
\end{tikzcd}$$

To see what sort of common cover link sets are created for the constituency structure of typical linguistic utterances, let us examine two simple (and typical) sequences of parts-of-speech with their bracketing. The part-of-speech sequence describing a typical simple verb phrase (in English) is $[V\ \ [N]\ \ [PP\ [DT\ N]]]$ (where V is a verb, N a noun, DT a determiner and PP a preposition). This has the following common cover links, all of which are of depth 0:



The simple sentence [$[DT\ N]$ $[V\ [N]]$] has both common cover links of depth 0 and of depth 1:



Given the set $R_{\mathcal{C}}$ it is simple to calculate the bracketing $\mathcal{C}$ from which it was derived by following the simple procedure outlined below.

2.3.3 (5.1.29). Algorithm. [simple bracket reconstruction from $R_{\mathcal{C}}$]

1. For every word x, construct the smallest bracket which covers x and all y such that $x \xrightarrow{0} y \in R_{\mathcal{C}}$ (this bracket is $B_0(x)$).

2. Having constructed $B_d(x)$ ($d \geq 0$) and if there are links $x \xrightarrow{d+1} y$ in $R_{\mathcal{C}}$, construct $B_{d+1}(x)$ by constructing the smallest bracket which covers $B_d(x)$ and all words y such that $x \xrightarrow{d+1} y \in R_{\mathcal{C}}$.

Notation I write $\mathcal{A}(L)$ for the result of applying the simple bracket reconstruction algorithm (Algorithm 2.3.3) to a set of common cover links L.

It is easy to see (Lemma 5.1.30) that, for any bracketing $\mathcal{C}$, this procedure reproduces $\mathcal{C}$ from $R_{\mathcal{C}}$, that is, $\mathcal{A}(R_{\mathcal{C}}) = \mathcal{C}$. The algorithm can, of course, be applied to any set of common cover links L to produce a set of brackets. In general, however, it is easy to verify that $\mathcal{A}(L)$ is not necessarily a bracketing (because brackets may cross).

As will be seen in the following sections, this algorithm is a theoretical rather than a practical tool. In practice, subsets of $R_{\mathcal{C}}$ are used and then other algorithms must be applied to reconstruct the bracketing. The simple reconstruction algorithm is, however, used in several proofs in chapter 5.

2.4 Representative Subsets

The process outlined in Algorithm 2.3.3 for the reconstruction of a bracketing $\mathcal{C}$ from $R_{\mathcal{C}}$ may construct a bracket multiple times. This happens when some bracket $X \in \mathcal{C}$ has more than one word of minimal depth d under X. When this is the case, the common cover links of depth d of any of the words of minimal depth d under X are sufficient to reconstruct the bracket X. It is then possible to discard the links of depth d of some (but not all) of these words without losing the ability to reconstruct the bracket.

At this point it is important to notice a subtle difference in the reconstruction process between links of depth 0 and other links. When a word x has no link $x \xrightarrow{d} y \in R_{\mathcal{C}}$ for $1 \leq d$ then no bracket $B_d(x)$ is constructed (because this bracket would be identical to $B_{d-1}(x)$, which has already been constructed). On the other hand, when a word x has no links $x \xrightarrow{0} y \in R_{\mathcal{C}}$, a bracket $B_0(x) = \langle x \rangle$ is constructed, because this bracket cannot be constructed in any other way. For this reason it is not possible to discard the links of depth 0 of a word x even if they result in a bracket being multiply constructed, because discarding the links would lead to the construction of an incorrect bracket $\langle x \rangle$. For links of non-zero depth there is no such problem and they may be discarded without losing the ability to reconstruct $\mathcal{C}$ by the algorithm. The process of discarding links will be referred to as *selecting representatives* for the bracket X. The selection of representatives results in a *representative subset* of $R_{\mathcal{C}}$, which is defined as follows:

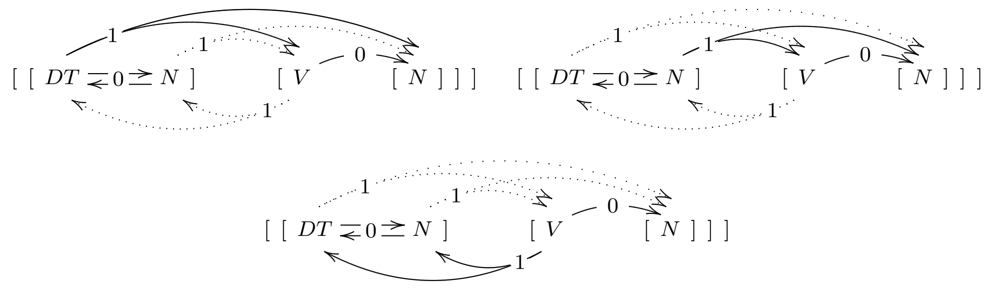
2.4.1 (5.1.18). Definition. [representative subset] Let $\mathcal{C}$ be a bracketing. A subset $R \subseteq R_{\mathcal{C}}$ is a representative subset of $R_{\mathcal{C}}$ iff:

1. For every $x \xrightarrow{0} y \in R_{\mathcal{C}}$ also $x \xrightarrow{0} y \in R$.
2. For every $X \in \mathcal{C}$, if d is the height of X then there is at least one word x of minimal depth d under X such that for every $x \xrightarrow{d} y \in R_{\mathcal{C}}$ also $x \xrightarrow{d} y \in R$. This word x is a *representative* for the bracket X.
3. For every word x and every depth d if $x \xrightarrow{d} y \in R$ and $x \xrightarrow{d} z \in R_{\mathcal{C}}$ then $x \xrightarrow{d} z \in R$.

The first part of the definition ensures that all links of depth 0 are in R. The second part ensures that for each bracket at least one word remains from which the bracket can be reconstructed. Finally, the last part of the definition ensures that either all or none of the links of depth d based at x are discarded (otherwise bogus brackets are created by the reconstruction algorithm).

The representative subsets for the examples given above are easily calculated. For the first example, $[V\ [N]\ [PP\ [DT\ N]]]$, the only representative subset of $R_{\mathcal{C}}$ is $R_{\mathcal{C}}$ itself, because $R_{\mathcal{C}}$ contains only links of depth 0. The second example,

[[*DT* *N*] [*V* [*N*]]] has three words (*DT*, *N* and *V*) which are of minimal depth 1 under the top bracket. Any combination of these words can be chosen as the representatives of this top bracket. The following diagram gives the three minimal representative subsets of $R_{\mathcal{C}}$ for this example, in which the top bracket has exactly one representative. The solid links are in the representative subset while the dotted links are in $R_{\mathcal{C}}$ but not in the representative subset:



2.5 Shortest Common Cover Link Sets

By looking at the examples given above, it is immediately clear that many of the links in the representative subsets of $R_{\mathcal{C}}$ are redundant, even when a minimal representative subset is selected. Formally, this can be seen from the following transitivity property which allows longer links in $R_{\mathcal{C}}$ to be deduced from shorter links (the proof is in chapter 5).

2.5.1 (5.1.16). Lemma (linear transitivity of $R_{\mathcal{C}}$). *Let $R_{\mathcal{C}}$ be the common cover link set of a bracketing $\mathcal{C}$. If $y \in (x,z)$, $x \overset{d_1}{\to} y \in R_{\mathcal{C}}$ and $y \overset{d_2}{\to} z \in R_{\mathcal{C}}$ then $x \overset{d}{\to} z \in R_{\mathcal{C}}$ with the following depth (See figure 2.1):*

1. *If $y \to x \in R_{\mathcal{C}}$ then $d = \max(d_1, d_2)$.*
2. *Otherwise $d = d_1$.*

Linear transitivity can be used to further reduce the set of links needed to reconstruct a bracketing by removing from a representative subset any links which

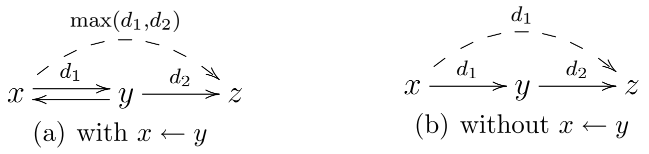
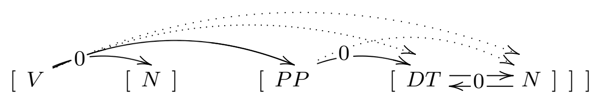


Figure 2.1: Linear transitivity of the common cover links in $R_{\mathcal{C}}$.

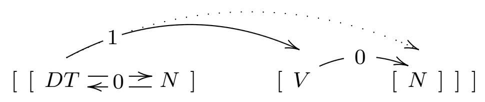
can be deduced from shorter links in the set. Such a reduced set is a *shortest common cover link set*:

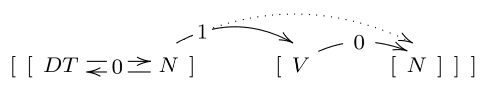
2.5.2 (5.1.19). Definition. [shortest common cover link set] Let $\mathcal{C}$ be a bracketing. A set $S \subseteq R_{\mathcal{C}}$ is a shortest common cover link set if there is a representative subset R of $R_{\mathcal{C}}$ such that $x \overset{d}{\to} z \in S$ iff $x \overset{d}{\to} z \in R$ and there is no word $y \in (x, z)$ such that $x \overset{d_1}{\to} y \in R$ and $y \overset{d_2}{\to} z \in R$.

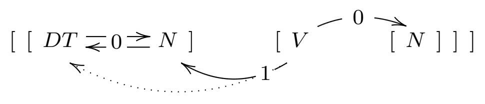
The shortest common cover link set is so named because it uses the shortest possible links to represent the structure. For every representative subset R of $R_{\mathcal{C}}$ the shortest common cover link set contained in that set is determined uniquely, so I write $S(R)$ for this set. A larger representative subset will usually (but not always) result in a larger shortest common cover link set. For the verb phrase example given above, for which $R_{\mathcal{C}}$ was the only representative subset, $S(R_{\mathcal{C}})$ is given in the next diagram, where solid links are in $S(R_{\mathcal{C}})$ and dotted links are in $R_{\mathcal{C}} \setminus S(R_{\mathcal{C}})$:



In the second example given (that of a sentence part-of-speech sequence) the three minimal representative subsets result in the following three shortest common cover link sets (solid links are in the shortest common cover link set and dotted links are in the representative subset):

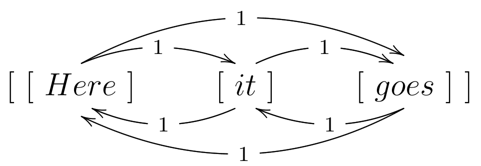


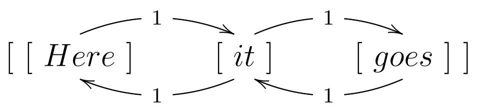




The shortest common cover links sets in these examples are already more similar to standard dependency structures than the full common cover link set of the bracketing. However, some differences are still evident, such as the back and forth links between the determiner (*DT*) and noun (*N*). The relation between the shortest common cover links and dependency structures is discussed in chapter 4.

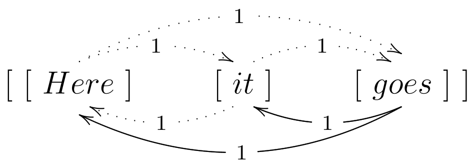
We can now see that there are good linguistic reasons for having allowed representatives to be chosen before calculating the shortest common cover link set. Consider the simple utterance [[*Here*] [*it*] [*goes*]]. Three words, *here*, *it* and *goes* are of minimal depth 1 under the top bracket. The utterance therefore has the following set $R_{\mathcal{C}}$ of common cover links (all of which are of depth 1) and the shortest common cover link set $S(R_{\mathcal{C}})$ based on $R_{\mathcal{C}}$:





$R_{\mathcal{C}}$ $S(R_{\mathcal{C}})$

This does not seem to reflect the fact that *goes* is the head of the utterance. However, since any of the three words can be selected as a representative of the top bracket, selecting the verb as the only representative results in the following representative subset, R_{goes} (dotted links are in $R_{\mathcal{C}} \setminus R_{\text{goes}}$). This set is also its own shortest common cover link set and agrees with the standard dependency analysis of this utterance.



The fact that not all shortest common cover link sets derived from a bracketing are linguistically acceptable should not be surprising. It only reflects the fact that, just like dependencies, shortest common cover link sets contain additional information not always available in the bracketing, namely, the identity of the heads.

2.6 Reconstructing the Bracketing

This section shows that any bracketing can be correctly reconstructed from any of its shortest common cover link sets. This means that any shortest common cover link set contains all the information which is available in the bracketing from which it was constructed and therefore that the shortest common cover set representation is at least as expressive as bracketing. This is theoretically interesting, but the practical implications of the reconstruction of the bracketing are also important. The parser described in chapter 3 produces a shortest common cover link set as the parse result. To produce a standard bracketing, the shortest common cover link set must be converted into a bracketing. The emphasis in this section will therefore be not only on showing that the bracketing can be reconstructed from a shortest common cover link set, but also to give an efficient algorithm for doing so. Moreover, because the parser of chapter 3 is incremental, I will also present here an incremental algorithm for converting the links into brackets. In this way, as the words of an utterance become available one by one, both a set of common cover links and a corresponding bracketing can be assigned to the prefix of the utterance already processed. In on-line processing situations it is often not clear when the end of the utterance is reached (a speaker may pause before continuing or terminate the utterance at any point) and it may be desirable to assign a syntactic analysis for every prefix of the utterance as it become available. In these situations, the incremental parser and bracketing algorithm can, at any moment, provide other processing modules with a parse (links and bracketing) of whatever part of the utterance was already processed. If and when the utterance is continued, this parse can be extended.

As mainly or only links of depth 0 and 1 seem to be needed to describe linguistic structures (section 4.4) the general claim that any bracketing can be reconstructed from any of its shortest common cover link sets is of mathematical rather than linguistic interest. Since the general reconstruction algorithm is also more complex than the algorithm when the links are all of depth 0 or 1, I will restrict the discussion in the present section to common cover links which satisfy this condition. The general case will be proved in the mathematical chapter of this work (section 5.2). An incremental bracket reconstruction algorithm is only given for the linguistically relevant case of links of depth 0 and 1.

2.6.1 Simple Reconstruction

I assume throughout this section that all links are of depth 0 or 1 (though some of the lemmas may also hold without this restriction). The reconstruction of the bracketing is conceptually performed in two steps. First, using linear transitivity (Lemma 2.5.1) a representative subset is constructed from the shortest common cover link set. Next, the brackets are deduced from the representative subset. In the present section I present a simple algorithm which works in exactly this way. In practice, the two steps can be combined to give more efficient algorithms, as in the incremental algorithm presented in section 2.6.3.

Let $\mathcal{C}$ be a bracketing and let S be a shortest common cover link set for this bracketing. Because S was created from some representative subset $R \subseteq R_{\mathcal{C}}$ it may be that not all links in $R_{\mathcal{C}}$ can be deduced from S by linear transitivity. At the same time, it may also be that links in $R_{\mathcal{C}} \setminus R$ can be deduced by linear transitivity from S. Therefore, simply taking the set of links deducible by linear transitivity from S does not necessarily produce a representative subset of $R_{\mathcal{C}}$. Since the bracketing $\mathcal{C}$ can only be correctly reconstructed from a representative subset of $R_{\mathcal{C}}$, we need a way to determine whether a given link $x \xrightarrow{d} y \in R_{\mathcal{C}}$ is in a representative subset $R \subset R_{\mathcal{C}}$. The following lemma shows that this is easy to determine for the *minimal* representative set $R(S)$ which contains S (the lemma also implies that this minimal set is unique, see Corollary 5.1.22).

2.6.1 (5.1.20). DEFINITION. [minimal representative subset] Let $\mathcal{C}$ be a bracketing and let S be a shortest common cover link set for this bracketing. A representative subset $R \subseteq R_{\mathcal{C}}$ is a *minimal representative subset* containing S if $S \subseteq R$ and there is no representative subset R' such that $S \subseteq R' \subset R$.

2.6.2 (5.1.21). LEMMA. *Let $\mathcal{C}$ be a bracketing, let S be a shortest common cover link set of $R_{\mathcal{C}}$ and let $R(S)$ be a minimal representative subset containing S. If $x \xrightarrow{d} y \in R_{\mathcal{C}}$ then $x \xrightarrow{d} y \in R(S)$ iff there is some z such that $x \xrightarrow{d} z \in S$.*

Having a way to determine whether a link is in $R(S)$, the reconstruction algorithm can now be described. Let $\mathcal{C}$ be a bracketing and let S be a shortest common cover link set for this bracketing. The reconstruction algorithm maintains a set L of links which is initialized to be equal to S. By applying Lemma 2.6.2, the algorithm makes sure that each link it adds to the set L is in $R(S)$, the unique minimal representative subset containing S. The algorithm looks at pairs of words x,z at increasing distance from each other. If there is $y \in (x, z)$ such that $x \xrightarrow{d_1} y \in L$ and $x \xrightarrow{d_2} z \in L$ then, by linear transitivity, there is a link $x \xrightarrow{d_3} z \in R_{\mathcal{C}}$. The algorithm needs to determine d_3 and then determine whether the link is in $R(S)$.

Since $d_1, d_2 \leq 1$ it follows from linear transitivity (Lemma 2.5.1) that $d_3 \leq \max(d_1, d_2) \leq 1$. Moreover, by linear transitivity, if $d_1 = 1$ then also $d_3 = 1$. If $d_1 = 0$ then there are two possibilities (again, by linear transitivity). If there is a link from y to x (in $R_{\mathcal{C}}$) then $d_3 = d_2$ and if there is no such link then $d_3 = d_1$. By Lemma 5.1.14, back and forth links between words must be of the same depth and, therefore, if a link from y to x is in $R_{\mathcal{C}}$, it must be of depth $d_1 = 0$. Therefore, this link is also in $R(S)$ (because all links of depth 0 are in a representative subset of $R_{\mathcal{C}}$). Since the algorithm works incrementally with words of increasing distance from each other, this link, if it exists, was already added to L, (that is, $y \xrightarrow{0} x \in R_{\mathcal{C}}$ iff $y \xrightarrow{0} x \in L$). The algorithm can therefore determine d_3. Having determined d_3, the algorithm uses Lemma 2.6.2 to decide whether $x \xrightarrow{d_3} y$ is in $R(S)$. If it is, it is added to L. The process continues until no more links can be added. This shows that when the process terminates, $L \subseteq R(S)$. From the definition of shortest common cover link sets it is then immediate that $L = R(S)$.

Having calculated $R(S)$ from S, the bracketing can be calculated from $R(S)$ using the simple bracket reconstruction algorithm (Algorithm 2.3.3). It is not difficult to verify (see Lemma 5.1.30) that if all links are of depth 0 and 1 then this algorithm correctly reconstructs the bracketing $\mathcal{C}$ from any representative subset of $R_{\mathcal{C}}$ (and in particular from $R(S)$). The requirement that the depth of all links is 0 or 1 is crucial here. The claim does not necessarily hold if this condition does not hold.

2.6.2 Paths

The main key to the reconstruction of the bracketing from a shortest common cover link set is linear transitivity, which allows a common cover link to be deduced from a sequence of shorter common cover links. Such sequences play a central role in the structure of shortest common cover link sets and will be referred to as *link paths* or simply as *paths.* These paths are defined as follows.

2.6.3 (5.1.26). Definition. [linear path] Let U be an utterance. A sequence $x_1, \ldots, x_m$ of words in U is a *linear path* from x_1 to x_m in U (written $x_1 - \ldots - x_m$) iff for each $1 < i < m$ $x_i \in (x_{i-1}, x_{i+1})$.

2.6.4 (5.1.27). Definition. [link path] Let L be a set of common cover links over an utterance U and let $x, y \in U$. An L-path from x to y is a set of links $\left\{x_i \overset{d_i}{\to} x_{i+1}\right\}_{i=1}^{m-1}$ in L such that $x_1 - \ldots - x_m$ is a linear path, $x_1 = x$ and $x_m = y$. In particular, for every $x \in U$, there is an empty L-path from x to x.

Notation I write $x \overset{L}{\to} y$ if there is an L-path from x to y and $x \overset{d,L}{\to} y$ if there is an L-path from x to y which begins with a link of depth d.

Let $R_{\mathcal{C}}$ be a common cover link set of a bracketing $\mathcal{C}$ and let S be a shortest common cover link set of $R_{\mathcal{C}}$. Linear transitivity implies that if there is an S-path from x to y ($x \overset{S}{\to} y$) and $x \neq y$ then there is a link $x \to y \in R_{\mathcal{C}}$. The depth of this link may depend on links which are not on the S-path from x to y. The fact that these links may be in $R_{\mathcal{C}}$ but not in S is the main source of complication in the reconstruction of $R_{\mathcal{C}}$ and the bracketing $\mathcal{C}$. But while the depth of a link cannot be deduced from the links in the link path connecting its two ends, it follows from the linear transitivity lemma (Lemma 2.5.1) that the depth of the first link in a link path is a lower bound on the depth of the combined link (Lemma 5.1.28). This is the reason for introducing the notation $x \overset{d,L}{\to} y$.

2.6.3 Incremental Reconstruction

The algorithm described in section 2.6.1 is simple but not very efficient because it first has to calculate all links in $R(S)$ (many of which are redundant) and only then deduces the bracketing. A simple (more efficient) algorithm can reconstruct $\mathcal{C}$ directly from S by assigning brackets incrementally to prefixes $[x_1, x_k]$ of U. As before, reconstruction is restricted to the case in which all links are of depth 0 or 1.

2.6.5 (5.4.1). ALGORITHM. [incremental reconstruction from S] Given is a set of links S of depth 0 or 1 over an utterance $\langle x_1, \ldots, x_n \rangle$. Let $S_k = \{x \xrightarrow{d} y \in S : x, y \in [x_1, x_k]\}$ be the restriction of S to $[x_1, x_k]$. The algorithm updates a bracketing $\mathcal{B}$.

- Initialize $\mathcal{B} = \{\langle x_1 \rangle\}$.
- For each $k = 2, \ldots, n$ perform the following modifications of $\mathcal{B}$, in the given order:
    1. For every link $x_i \xrightarrow{0} x_k \in S_k$, extend all brackets in $\mathcal{B}$ which cover x_i to cover x_k.
    2. For every link $x_i \xrightarrow{1} x_k \in S_k$:
        (a) Extend all brackets which cover $B_0^{\mathcal{B}}(x_i)$ to cover x_k.
        (b) If there is no $x \xrightarrow{d} y \in S_{k-1}$ such that $x \in B_0^{\mathcal{B}}(x_i)$ and $y \notin B_0^{\mathcal{B}}(x_i)$, add a bracket which covers x_k and $B_0^{\mathcal{B}}(x_i)$.
    3. If there is no x_i such that $x_k \xrightarrow{0} x_i \in S_k$ and $x_k \in B_0^{\mathcal{B}}(x_i)$ then add to $\mathcal{B}$ the smallest bracket which covers x_k and every x such that $x_k \xrightarrow{0,S_k} x$.
    4. If there is $x_k \xrightarrow{1} x_i \in S_k$ then add to $\mathcal{B}$ (if it is not already in $\mathcal{B}$) the smallest bracket which covers x_k and every x such that $x_k \xrightarrow{S_k} x$.
- Output $\mathcal{B}$.

This algorithm is much less intuitive than the algorithm given in the previous section. The following theorem shows that it does have the required property that when given as input a shortest common cover link set of any bracketing $\mathcal{C}$, the algorithm is guaranteed to reconstruct $\mathcal{C}$. The proof of this theorem is given in section 5.4 and uses the characterization of the shortest common cover link sets described in the next section.

2.6.6 (5.4.9). THEOREM (RECONSTRUCTION). *Let $\mathcal{C}$ be a bracketing over an utterance $U = \langle x_1, \ldots, x_n \rangle$ and let S be a shortest common cover link set for this bracketing. If S contains only links of depth 0 or 1 then applying Algorithm 2.6.5 to S outputs $\mathcal{C}$. The algorithm runs in time linear in the length of U.*

The next section shows (Lemma 2.7.4) that if S is a shortest common cover link over U then the sets S_k used by the incremental reconstruction algorithm are also shortest common cover link sets (on the prefixes $[x_1, x_k]$ of U). It follows that at each step the algorithm constructs a valid bracketing which corresponds to the structure assigned to the prefix by S_k.

2.7 Characterization of Shortest Common Cover Link Sets

In the previous sections the common cover link set and its subsets were defined based on bracketing. When parsing an utterance, the bracketing is not known and the parser needs to create a set of links which is a shortest common cover link set for a bracketing on that utterance. It is therefore necessary to determine the conditions on a set of links which make it a shortest common cover link set for some bracketing. Only sets which satisfy these conditions should be constructed by the parser. Since this restricts the possible sets which need to be considered, these conditions also have implications for learning, as will be discussed in chapter 6.

The following conditions characterize the shortest common cover link sets of bracketings over an utterance U. As in previous sections, the conditions are restricted to the case where the depth of the links is either 0 or 1. This seems enough to handle most or all linguistic cases (see section 4.4) and significantly simplifies the conditions.

2.7.1 (5.3.1). DEFINITION. [characterization] Let L be a set of common cover links of depth 0 or 1 over U. The set L is said to satisfy the *characterizing conditions* if for every $w, x, y, z \in U$ the following conditions hold:

1. *Monotonicity*: if $y \in (x, z]$, $x \overset{d_1}{\to} y \in L$ and $x \overset{d_2}{\to} z \in L$ then $d_1 \leq d_2$.
2. *Minimality*: if $x \overset{d_1}{\to} z \in L$ then there is no $y \in (x, z)$ such that $x \overset{L}{\to} y$ and $y \overset{L}{\to} z$.
3. *Connectedness*: if $x \overset{L}{\to} z$ and $y \in (x, z)$ then $x \overset{L}{\to} y$.
4. *Blocking*: if $w \overset{d_1}{\to} z \in L$ and, for some $y \in U$ and $x \in (w, z)$, $x \overset{L}{\to} w$ and $x \overset{d_2}{\to} y \in L$ then $d_1 = 1$ and $d_2 = 0$.

5. *Equality*: if $y \in [x, z)$, $x \xrightarrow{d_1} z \in L$, $z \xrightarrow{d_2} y \in L$ and $y \xrightarrow{L} x$ then $d_1 = d_2$.

6. *Resolution*: if $y \in (x, z)$, $x' \in (x, y]$, $x \xrightarrow{d_1} x' \in L$ and $x' \xrightarrow{L} y$ and if $z' \in (z, y]$, $z \xrightarrow{d_2} z' \in L$ and $z' \xrightarrow{L} y$ then there is $v \in [x, z]$ such that either $x \xrightarrow{d_1} v \in L$ and $v \xrightarrow{L} z$ or $z \xrightarrow{d_2} v \in L$ and $v \xrightarrow{L} x$.

The best way to understand these conditions is to look at the constructions which are forbidden by each condition.

1. *Monotonicity*: forbids $$\begin{tikzcd} x \arrow[r, "1"] \arrow[rr, bend left, "0"] & y & z \end{tikzcd}$$. This condition reflecting the fact that the depth of x under the smallest bracket covering x and y cannot be greater than its depth under the smallest bracket covering x and z.

2. *Minimality*: forbids link structures such as $$\begin{tikzcd} x \arrow[r, "0"] \arrow[rr, bend left, "0"] & y \arrow[r, "0"] & z \end{tikzcd}$$, because in such a structure the longer link can be deduced from the shorter links by linear transitivity, contrary to the definition of shortest common cover link sets.

3. *Connectedness*: forbids links structures such as $$\begin{tikzcd} x \arrow[rr, bend left, "0"] & y & z \end{tikzcd}$$, because if x is of minimal depth under the smallest bracket covering x and z then it must also be of minimal depth under the smallest bracket covering x and y.

4. *Blocking*: this condition is somewhat more complex than the previous ones. It forbids various different configurations, including the following typical examples:

$$\begin{tikzcd} w = y \arrow[r, shift left, "0"] \arrow[rr, bend left, "0"] & x \arrow[l, shift left] & z \end{tikzcd} \qquad \begin{tikzcd} w \arrow[r, shift left, "0"] \arrow[rrr, bend left, "1"] & x \arrow[l, shift left] \arrow[r, "1"] & y & z \end{tikzcd}$$

$$\begin{tikzcd} y & w \arrow[r, shift left, "0"] \arrow[rr, bend left, "1"] & x \arrow[l, shift left] \arrow[ll, bend right, "1"'] & z \end{tikzcd}$$

In all these configurations, the link from w to z implies that there should also be a link from x to z and, therefore, the link from w to z is not in the shortest common cover link set because it can be deduced by linear transitivity. Blocking does allow, however, the link configuration $$\begin{tikzcd} w = y \arrow[r, shift left, "0"] \arrow[rr, bend left, "1"] & x \arrow[l, shift left] & z \end{tikzcd}$$. The difference between this example and the previous ones has to do with the selection of representatives (Definition 2.4.1). In the last example, the link from y to z is of depth 1 and it is possible to select y and not x as the representative of the corresponding bracket. However, if the link from y to z is of depth 0 or if there is a link of depth 1 based at x (as in the previous examples) then x must be selected as a representative and the configuration is forbidden.

5. *Equality*: This condition requires that $d_1 = d_2$ in configurations such as the following:

$$\begin{tikzcd} x = y \arrow[r, shift left, "d_1"] & z \arrow[l, shift left, "d_2"] \end{tikzcd} \qquad \begin{tikzcd} x \arrow[r, shift left, "0"] \arrow[rr, bend left, "d_1"] & y \arrow[l, shift left] & z \arrow[l, "d_2"'] \end{tikzcd}$$

   This is because in these cases both x and z are of minimal depth under the smallest bracket covering them and their depth under this bracket must be the same.

6. *Resolution*: This is the most complicated condition. It applies to the following link structure, where it is possible that $x' = y$ or $z' = y$ (or both):

$$\begin{tikzcd} x \arrow[r, "d_1"] & x' \arrow[r, "L"] & y & z' \arrow[l, "L"'] & z \arrow[l, "d_2"'] \end{tikzcd}$$

   When such a configuration exists, there is a bracket B_x covering x and y and a bracket B_z covering z and y. Since both these brackets cover y and they may not cross, either $z \in B_x$ or $x \in B_z$. If $z \in B_x$ then, since x is of minimal depth under B_x, there must be a link path from x to z. Similarly, if $x \in B_z$ there must be a link path from z to x. The link path from x to z or from z to x *resolves* the question of whether it is B_x which covers z or B_z which covers x. Not only must one of these link paths exist, it must also begin with an appropriate depth. In the simplest cases (when $x' = y = z'$) the resolution condition forbids configurations such as:

$$\begin{tikzcd} x \arrow[r, "0"] \arrow[rr, bend left, "1"] & y & z \arrow[l, "0"'] \end{tikzcd} \qquad \begin{tikzcd} x \arrow[r, "0"] \arrow[rr, bend left, "1"] & y & z \arrow[l, "0"'] \arrow[ll, bend left, "1"] \end{tikzcd}$$

These examples clarify some of the reasons behind the definition of the characterizing condition, but do not prove that the conditions indeed characterize shortest common cover link sets. The correctness of the characterization is stated by the following theorem, (which is proved in chapter 5):

2.7.2 (5.3.2). THEOREM (CHARACTERIZATION). *Let L be a set of common cover links of depth 0 or 1 over U. There exists a bracketing $\mathcal{C}$ over U such that L is a shortest common cover link set of $\mathcal{C}$ iff L satisfies the characterizing conditions of Definition 2.7.1.*

The characterization is given by six simple properties, each of which is easy to check on a set of links. The following lemma shows that none of these conditions is redundant.

2.7.3 (5.3.3). LEMMA. *For each of the six conditions in Definition 2.7.1, there is a set of links L which violates that condition but satisfies all other conditions.*

One important and easily verified property of this characterization is that if it is satisfied by the full set of links L then it must also be satisfied by $L^{[x,y]} = \{u \xrightarrow{d} v \in L \ : \ u, v \in [x,y]\}$, the restriction of L to a segment $[x,y] \subseteq U$.

2.7.4 (5.3.4). LEMMA. *Let L be a set of common cover links over an utterance U and let $L^{[x,y]} = \{u \xrightarrow{d} v \in L \ : \ u, v \in [x,y]\}$ be the restriction of L to a segment $[x,y] \subseteq U$. If L satisfies the characterizing conditions then so does $L^{[x,y]}$.*

In particular, this holds for any prefix $[x,y]$ of the utterance U. This property of the characterization conditions means that when parsing incrementally the set of links assigned each prefix of an utterance must be a shortest common cover link set for that prefix. Each prefix is therefore always assigned a valid parse.

2.8 Conclusion

This chapter defined the shortest common cover link sets induced by a bracketing and showed that a bracketing can always be reconstructed from any of the shortest common cover link sets it induces. To make the shortest common cover link sets a syntactic representation in their own right, characterizing conditions for shortest common cover link sets were given. These conditions will from now on be seen as the definition of these link sets.

Chapter 3

Parsing

The common cover links of the previous chapter were designed with parsing in mind. It is the purpose of the present chapter to show that parsing can be performed using such links and that using common cover links indeed offers advantages to a parser. One advantage of the common cover link representation was already mentioned in the previous chapter: the restriction of the depth of links to 0 and 1 seems to capture the skewness of natural language parse trees. A parser which only outputs links with depth 0 and 1 thus avoids even having to consider many alternative parses which are not valid for natural languages. This considerably restricts the parser's search space.

An additional advantage of using common cover links in parsing is that this representation lends itself easily to incremental parsing. Incremental parsing is the ability to perform parsing as the words of an utterance are being received one by one. An incremental parser can assign a structure to whatever prefix of the utterance it has already received without having to wait for the end of the utterance. This is usually considered a useful property for the parser to have, for two reasons. First, in on-line tasks (such as speech processing) an analysis of the input is often required before the full utterance has been received. Second, psycholinguistic evidence suggests that humans process language incrementally (Crocker et al. 2000). Cognitive modeling of language processing would therefore require some sort of incremental parsing. In the present work, however, incrementality was adopted for a third reason - it restricts the possibilities the parser must consider and (no less importantly) the possibilities a learner has to consider when learning to parse incrementally (see chapter 6). This argument is based indirectly on the psycholinguistic evidence for incremental processing: if humans can parse incrementally then there is no need to consider the end of the utterance before making parsing decisions about a prefix of that utterance. Of course, for this to work, the incrementality of the parser has to be similar to that of the human processor. The version of incrementality introduced here is probably only a rough approximation of the incrementality of human language processing, but it does already seem to be beneficial in allowing simple learning and parsing with state-of-the-art accuracy.

This chapter begins with section 3.1, which defines what incremental parsing is. Having defined incremental parsing, I go on to investigate (in section 3.2) the conditions which must be imposed on an incremental parser so that it outputs a valid common cover link structure. The parser is required to output a shortest common cover link set and the characterizing conditions of the previous chapter can be combined with the requirements of incrementality to define the set of links the incremental parser may add at each step. This leads to the definition of the incremental parsing algorithm in section 3.3. It is shown that this algorithm only outputs shortest common cover link sets and that it can output any shortest common cover link set. This parsing algorithm is not deterministic. The choices left unspecified in the algorithm must be filled in by defining appropriate choice functions. These functions are defined in section 3.4 an are the subject of learning in chapter 6.

3.1 Incremental Parsing

For parsing to be incremental, the parser must construct the syntactic structure as the utterance is being read. This is no exact definition and leaves much room for variation. First, the unit of incrementality must be chosen, that is, how much input may be read at each step of processing before structure is assigned? What humans exactly do is not entirely clear so the most simple and natural choice is to assign structure with each additional word read. Of course, smaller phonetic units or larger combinations of words could also be considered as the unit of incrementality but it then becomes a non-trivial task just to determine what the size of each incremental step should be. For this reason, most incremental models (including the one I present here) take the single word (as defined by the written language) to be the unit of incrementality. This is probably a little simplistic and somewhat arbitrary (because of a certain arbitrariness of writing systems) but (as work with written text has often shown) is a very reasonable first approximation.

Having chosen the unit of incrementality, we must determine what parsing decisions the parser is required to take at each incremental step and whether these decisions may be undone later on. Of course, if all parsing decisions may be postponed or undone by subsequent steps, the parser cannot be considered incremental since it can effectively perform parsing after having read the full utterance. Most incremental parsers are therefore not allowed to change any decision already made and are not allowed to postpone decisions indefinitely. To avoid running into trouble by making hasty decisions, many incremental parsers are allowed to perform underspecified parsing decisions which are left for subsequent steps to specify. The fact that underspecification has to be used seems to be the direct result of the fact that most incremental parsers are based on non-incremental syntactic formalisms and parsers. As these formalisms were not designed for incrementality, their use without underspecification often requires the incremental parser to commit itself to decisions it cannot make.

The key to successful incremental parsing seems to be the ability to detect the structural relations which are fixed by each prefix of the utterance. Adding these relations to the syntactic structure (as they become fixed) but nothing more should result in the correct parse. These relations should be neither weaker than is required for constructing the syntactic structure nor stronger than the parser can commit itself to (see section 4.3 for an analysis of one linguistic aspect of this requirement). These relations are often weaker than the relations described by formalisms designed for non-incremental parsing and therefore must be described by underspecified versions of those formalisms. In this sense, common cover links represent a weaker relation than the standard dependency relation. While a dependency link from x to y indicates that y is the *head* of an argument of x, a common cover link from x to y only indicates that y is part of an argument of x. Common cover links represent (some sort of) a domination relation while a dependency link represents direct domination (a complete discussion of the relation between dependencies and common cover links appears in chapter 4). Common cover links can therefore be used directly for incremental parsing without having to use underspecification.

An incremental dependency parser is often defined as a dependency parser which assigns a single connected structure to every prefix of an utterance (Nivre 2004). This requirement is probably intended to stop the parser from leaving the utterance as a sequence of unconnected words until the end of the utterance is read and only then connect them all together in the last step. As Nivre notes, this requirement is too strong because it cannot handle constructions where a head has several arguments which appear to its left. In this work I therefore adopt a different definition of incrementality which at each incremental step allows the parser to add links only between the last word read and the words preceding it. This seems to me a much more natural definition of incrementality and I will show that under this definition any bracketing can be constructed by the common cover link incremental parser.

Given an input utterance U, the parser needs to construct a set S of common cover links over U. The set S should be a shortest common cover link set for some bracketing $\mathcal{C}$ of U. As in the previous chapter, because linguistic structures seem to use mainly or only links of depth 0 or 1 (see section 4.4) the discussion here assumes that this restriction on the depth of the links holds. I begin with some definitions to describe the incrementality of the process.

Notation Given an utterance U, I write $x_k(U)$ for the k'th word in U (when there is no risk of confusion, I simply write x_k). I also write U_k for the prefix $[x_1(U), x_k(U)]$ of length k of U. Similarly, if S is a shortest common cover link set for a bracketing over U, I write S_k for the set $\{x \xrightarrow{d} y \in S \ : \ x, y \in U_k\}$, the restriction of S to U_k.

The following definition defines incrementality for common cover link parsing. All it allows the parser to do as it reads the word $x_k(U)$ is to add links between $x_k(U)$ and words previously read (that is, words in U_{k-1}).

3.1.1 (5.5.1). DEFINITION. [incremental parsing]

1. A common cover link parser is a function $\mathcal{P}$ on the set of utterances such that, for any utterance U, $\mathcal{P}(U)$ is a shortest common cover link set over U.

2. The parser $\mathcal{P}$ is incremental if for every prefix U_k of U, $\mathcal{P}(U_k) = S_k$ is a shortest common cover link set over U_k and, for each $2 \leq k \leq n$, $S_{k-1} \subseteq S_k$ and $S_k \setminus S_{k-1}$ contains only links which have one end at $x_k(U)$.

What makes incremental parsing at all possible is the fact that if S is a shortest common cover link set over an utterance U then the restriction S_k of S to a prefix U_k of U is itself a shortest common cover link set for a bracketing of U_k (Lemma 2.7.4). This contrasts with many other formalisms (mainly phrase structure based) which often do not assign a well-formed representation to the prefixes of an utterance.

3.2 Conditions on Incremental Parsing

After the parser has calculated S_{k-1} based on U_{k-1}, it reads the next word, x_k, and needs to determine which links should be added to S_{k-1} to form S_k. The incrementality of the parser requires the links added in this step to have one end at x_k. To ensure that the set of links S_k assigned by the parser to each prefix U_k is a shortest common cover link set, the six characterizing conditions of Definition 2.7.1 must be satisfied.

The links are added one by one. The *monotonicity*, *minimality*, *blocking* and *equality* conditions must be satisfied with each link being added because once any of these conditions is violated, the violation cannot be repaired by adding additional links. The *connectedness* condition can be repaired after being violated but it is much simpler not to allow it to be violated in the first place. Therefore, the parser only adds links such that these five condition are satisfied with every link being added. For the last characterizing condition, *resolution*, it is not always possible to avoid its violation but it is always possible to repair such a violation (section 3.2.3). Therefore, once such a violation is created, the parser must continue to add links until the violation is repaired.

3.2.1 The Adjacency Properties

The first four characterizing properties (*monotonicity*, *minimality*, *connectedness* and *blocking*) are the *adjacency properties*. To ensure the preservation of the adjacency properties, a link should be added only from a word to another word *adjacent* to it according to the following definition. Adjacency also specifies the minimal depth of such a link.

3.2.1 (5.5.2). DEFINITION. [adjacency] Let L be a set of common cover links of depth 0 or 1 over an utterance U. A word y is adjacent to x with depth $d \leq 1$ relative to L (written $x \dashv^L_d y$) iff for every $z \in (x, y)$:

1. $x \overset{L}{\rightarrow} z$ (connectedness).
2. $z \rightarrow y \notin L$ (minimality).
3. There is no $w \in U$ such that $z \overset{1}{\rightarrow} w \in L$ and $z \overset{L}{\rightarrow} x$ (blocking).

The depth d is 1 iff there exists $z \in (x, y)$ such that $z \overset{L}{\rightarrow} x$ (blocking) or $x \overset{1}{\rightarrow} z \in L$ (monotonicity). Otherwise, $d = 0$.

An adjacency $x \dashv^L_d y$ is *unused* if $x \rightarrow y \notin L$.

Notation I write $x \dashv^L y$ if there exists $d \leq 1$ such that $x \dashv^L_d y$.

If a link is added from a word to another word which is not adjacent to it (or with a depth which is smaller than that of the adjacency) then at least one of the four adjacency properties is violated. The need to maintain the adjacency properties (for each link added) implies that adjacency is a necessary condition for adding a link between words. Therefore, the incremental parser only adds links where adjacency is satisfied.

Adjacency is a necessary condition for adding a link, but, as the next lemma shows, adjacency in itself is not a sufficient condition to ensure minimality and blocking. As is also shown by the lemma, to ensure these two properties hold, links may only be added between words which are not already *covered* by a link, as defined by the following definition.

3.2.2 (5.5.3). Definition. [covering links] A pair of words $\langle x, y\rangle$ in an utterance U is *covered* by a link $u \overset{d'}{\rightarrow} v$ over U if $x \in (u, v)$ and $y \in [u, v]$. The pair $\langle x, y\rangle$ is covered by a set L of common cover links over U if there exists a link $u \overset{d'}{\rightarrow} v \in L$ such that $u \overset{d'}{\rightarrow} v$ covers $\langle x, y\rangle$.

3.2.3 (5.5.4). Lemma. *Let U be an utterance. Let L be a set of common cover links over U such that all links in L are of depth 0 or 1.*

1. *If L satisfies the adjacency properties and if $x \dashv^L_d y$ and the pair $\langle x, y\rangle$ is not covered by L then, for any $d \leq d' \leq 1$, $L \cup \{x \overset{d'}{\rightarrow} y\}$ satisfies the adjacency properties.*

2. *There exists a set L and a link $x \overset{d'}{\rightarrow} y$ such that L satisfies all characterizing conditions of Definition 2.7.1, $x \dashv^L_d y$ for $d \leq d'$ but $L \cup \{x \overset{d'}{\rightarrow} y\}$ violates minimality or blocking.*

As the parser adds links to the parse, new adjacencies may be created while other adjacencies may be blocked. As long as the parser only adds links between adjacent words which are not covered, the adjacency properties are guaranteed to hold at every step. The following lemma describes the adjacencies of each word when these conditions are satisfied. It shows that consecutive words are always adjacent, that if $x \rightarrow y$ then y is adjacent to x and that every word x has at most one unused adjacency on each side, that is, a word which is adjacent to x but not attached to x by a link from x. The unused adjacency is always the adjacency furthest away from x.

3.2.4 (5.5.5). Lemma. *Let L be common cover link set over an utterance U such that all links in L are of depth 0 or 1 and such that L satisfies the adjacency properties.*

1. *If x and y are consecutive words in the utterance then $x \dashv^L_0 y$ and $y \dashv^L_0 x$.*

2. *If $x \overset{d'}{\rightarrow} y \in L$ then $x \dashv^L_d y$ for some $d \leq d'$.*

3. *If, for $y_1 \in (x, y_2)$, $x \dashv^L_{d_1} y_1$ and $x \dashv^L_{d_2} y_2$ then $x \overset{d'}{\rightarrow} y_1 \in L$ for some $d_1 \leq d'$ (in words: on each side of x there is at most one unused adjacency and this word is the adjacent word furthest away from x on that side).*

These properties of adjacency apply to any parser which constructs a set of common cover links by adding links one by one in such a way that the adjacency properties are preserved. In particular, this holds for incremental parsing. The next sections look at the remaining two properties: *equality* and *resolution*.

3.2.2 Equality

Equality can and must be preserved with every link added by the parser, because once equality is violated, this violation cannot be repaired by adding additional links. Equality imposes a simple restriction on the possible depth assigned to a link. When the parser wants to add a link $x \xrightarrow{d} y$ to a set of links L, it has to check two possibilities:

1. If $y \xrightarrow{L} x$ then d must be equal to the depth of the first link in the L-path from y to x.
2. If there is z such that $y \in (x, z)$, $z \xrightarrow{d'} x \in L$ and $y \xrightarrow{L} z$ then d must be equal to d'.

As long as L satisfies the adjacency properties, connectedness and minimality imply that these two cases cannot occur simultaneously. Directional uniqueness (Lemma 5.1.25) implies that, in the second case, z is determined uniquely. Therefore, equality imposes at most a single requirement for d. This value of d may, however, conflict with the minimal value for d imposed by adjacency.[1] In such cases, the link cannot be created. The following definition summarizes the conditions a link has to satisfy for the parser to be allowed to add it to the set of links the parser is constructing. The definition imposes incrementality by only allowing the link to be added if one of its ends is at the last word having been read by the parser.

3.2.5 (5.5.6). Definition. [incrementally addable link] Let L be a set of common cover links over a prefix U_k of an utterance U such that all links in L are of depth 0 or 1 and such that L satisfies the adjacency properties and equality. A link $x \xrightarrow{d'} y$ is incrementally addable to L over U_k iff:

1. (*incrementality*) $x_k \in \{x, y\} \subseteq U_k$ and $d' \leq 1$.
2. (*adjacency*) For some $d \leq d'$, $x \dashv^{L}_{d} y$ is an unused adjacency in L.
3. (*non-covered*) L does not cover the pair $\langle x, y \rangle$.
4. (*equality*) The set $L \cup \{x \xrightarrow{d'} y\}$ satisfies equality.
5. (*forcing*) If there is $u \xrightarrow{1} x \in L$ such that $y \in [u, x)$ then $d' = 1$.

[1]An example of such a conflict is give by the following diagram (solid links are in L) where adjacency requires that the depth of the link from x to y be 1 while equality requires this depth to be 0:

$$\begin{tikzcd}
x \arrow[r, shift left, "0"] \arrow[rr, bend left, dashed] & \bullet \arrow[l, shift left] & y \arrow[l, "0"']
\end{tikzcd}$$

3.2.3 Resolution

The last property the parser has to maintain is resolution. This property, however, cannot be maintained with every link added, because in many cases a resolution violation can be repaired only by a link which was not addable at the time the violation was created. A simple (and typical) example of this is shown in the following diagram, where the link $z \xrightarrow{0} x$ (which repairs the resolution violation) is not addable before the link $z \xrightarrow{0} y$ (which creates the violation) is added:

$$\begin{tikzcd}
x \arrow[r, "0"] & y & z \arrow[l, "0"'] \arrow[ll, bend right, "0"']
\end{tikzcd}$$

In this, resolution differs from all other characterizing properties.

A resolution violation is defined for a pair of words such that there is a word between them at which link paths from these two words meet. For a given pair of words there may be more than one word between them at which paths from the two words meet. What is important for the resolution property is the depth of the first link on the two paths which meet. As can be seen in the following example, these depths may depend on the word at which the two paths meet.

$$\begin{tikzcd}
x \arrow[r, "0"] \arrow[rr, bend left, "1"] & y_1 & y_2 & z \arrow[l, "0"'] \arrow[ll, bend right, "1"']
\end{tikzcd}$$

For this reason, the definition of a resolution violation must also specify the depths of the first links in the two meeting paths.

3.2.6 (5.5.7). Definition. [resolution violation] Let L be a set of links over an utterance U such that all links in L are of depth 0 or 1. A tuple $\langle x, z, d_1, d_2\rangle$ is a *resolution violation* in L if $x, z \in U$, $x \in [x_1(U), z)$ and there are $y \in (x, z)$, $x' \in (x, y]$ and $z' \in (z, y]$ such that $x \xrightarrow{d_1} x' \in L$, $x' \xrightarrow{L} y$, $z \xrightarrow{d_2} z' \in L$ and $z' \xrightarrow{L} y$ but there is no v such that either $x \xrightarrow{d_1} v \in L$ and $v \xrightarrow{L} z$ or $z \xrightarrow{d_2} v \in L$ and $v \xrightarrow{L} x$.

A tuple $\langle x, z, d_1, d_2\rangle$ is a *minimal resolution violation* in L if there is no resolution violation $\langle u, w, d_3, d_4\rangle$ in L such that $[u, w] \subset [x, z]$ or $[u, w] = [x, z]$ and $d_3 > d_1$.

3.3 The Parser Algorithm

Given this definition of resolution violation, the incremental parser algorithm can be stated.

3.3.1 (5.5.8). Algorithm. [incremental parser] Let U be an utterance of length n. The algorithm maintains a set L of common cover links.

- Initialize $L = \emptyset$ and $k = 2$.
- While $k \leq n$:
  1. If there is a minimal resolution violation $\langle x, z, d_1, d_2 \rangle$ in L, add an addable link $u \xrightarrow{d} v$ such that $u, v \in [x, z]$ and such that if $u = x$ then $d = d_1$ and if $u = z$ then $d = d_2$.
  2. Otherwise, increment k by 1 or add to L a link incrementally addable to L over U_k.

As long as there are no resolution violations, the algorithm may repeatedly choose whether to add a link (if an addable link exists) or to go on to the next input word. When a resolution violation is created, the parser must first resolve this violation before going on to the next word. To show that the algorithm is well defined, it must be shown that multiple minimal resolution violations never impose conflicting requirement on the link to be added and that as long as a minimal resolution violation exists, an addable link as required by the algorithm can be added. Showing that such a link exists also shows that the algorithm can repair any resolution violation created (because the number of possible links to be added is finite). Finally, it has to be shown that this algorithm can construct any shortest common cover link over U. This is summarized by the following theorem.

3.3.2 (5.5.17). THEOREM (INCREMENTAL PARSER CORRECTNESS). *The incremental parser, Algorithm 3.3.1, is well-defined and always outputs a shortest common cover link set. Moreover, it can output any common cover link set.*

3.4 Parsing Functions

The incremental parser described above is a nondeterministic algorithm. It specifies a set of links which may be added at each step but does not indicate which of these links should be added (if at all). A deterministic incremental parser is an algorithm which specifies how the nondeterministic algorithm should make this choice in every step. This is defined in terms of a *parsing function* which performs a single parse step - adding one link or none to a given set of links over a given prefix of an utterance.

3.4.1. DEFINITION. [parsing function] A function $\mathcal{P}$ which maps every utterance U and common cover link set L over that utterance to another common cover link set L' over that utterance is a parsing function iff:

1. $L \subseteq L'$ and $|L' \setminus L| \leq 1$.
2. If $\{l\} = L' \setminus L$ then l is incrementally addable to L over U.
3. If L has a minimal resolution violation $\langle x, z, d_1, d_2 \rangle$ then $L' \setminus L = \{u \overset{d}{\rightarrow} v\}$ such that $u, v \in [x, z]$ and if $u = x$ then $d = d_1$ and if $u = z$ then $d = d_2$.

The parsing function can be used to create a deterministic parser by applying the parsing function repeatedly to a prefix of the utterance. Every time the parsing function adds no links, the next word in the utterance may be read and the parsing function can be applied to the new (extended) utterance prefix.

3.4.2. ALGORITHM. [deterministic incremental parser] Let $\mathcal{P}$ be a parsing function and let U be an utterance of length n. The algorithm maintains a set L of common cover links.

- Initialize $L = \emptyset$ and $k = 2$.
- While $k \leq n$:
    1. Let $L = \mathcal{P}(L, U_k)$.
    2. If L remains unchanged, increment k.

Since the parsing function $\mathcal{P}$ completely determines the behavior of the parsing algorithm, I will usually simply refer to the parsing function as the *parser*. Chapter 6 presents a family of parse functions and a learning procedure to select one of these functions based on a sequence of unannotated example utterances.

3.5 Conclusion

This chapter defined an incremental common cover link parser. The incrementality of the parser was defined in terms of the links the parser may add at each step: these links are required to have one end at the last word received by the parser. This seems a more natural definition of incrementality than that often used for dependency parsing, where the parser is required to construct a connected dependency structure for each prefix of the utterance being parsed. In contrast to this connected structure based definition of incrementality, which has problems dealing with left branching structures, the definition of incrementality introduced in this chapter has no such limitation. The incremental parser is shown to be able to construct any bracketing.

The incremental parser adds links one by one to create a shortest common cover link set for the utterance it is parsing. Several properties were defined to describe the links the parser may add at every step. The most important of these properties is adjacency, which considerably restricts the set of links which may be added at each step. In particular, a link from x to y may be added only if there is already a link path from x to each of the words between x and y. Thus, a link can be added to y only if y appears directly next to an already constructed syntactic unit 'headed' by x. This property is a direct consequence of the definition of the shortest common cover link sets and has consequences for learning, as described in chapter 6.

Chapter 4

Linguistic Analysis

The previous chapters defined and described the formal properties of common cover links and an incremental parser based on these links. The present chapter discusses the linguistic properties of the common cover link representation and of incremental parsing. While chapter 2 concentrated on the relation between common cover links and bracketing (constituent structure), the link-based nature of the representation suggests that it may be more closely related to another standard linguistic formalism: dependency structures. The present chapter therefore begins, in section 4.1, by presenting some basic properties of dependency structures which are then used in section 4.2 to compare common cover links and dependencies. It is shown that probably every dependency link of a sentence is also a common cover link and that most (but not all) these dependency links are in a shortest common cover link set of the sentence. At the same time, there are many common cover links which are not dependency links. The three main differences between dependency structures and shortest common cover link sets are then discussed: exocentric structures, link depths and adjacency.

Because sentences with a syntactically ambiguous prefix pose a challenge for incremental parsers, they have been used extensively by psycholinguists to study the human incremental parser. It turns out that humans can easily handle some initial ambiguities while failing to process others correctly without performing conscious reanalysis. In section 4.3, I discuss this problem, known as *reanalysis*, and show how the common cover link incremental parser can successfully handle at least some of the initial ambiguities which humans can process without difficulty. I also indicate briefly some similarities between common cover link parsing and models put forward by psycholinguists to describe the difference between easy and difficult reanalysis.

An assumption made several times in the previous chapters is that common cover links of depth 0 and 1 are sufficient to describe the constituency structure of natural languages. In section 4.4, I look more closely at this hypothesis by examining the bracketing found in several annotated corpora. While no definite conclusions can be drawn from such a survey, the data seems to suggest that the depth of common cover links needed to describe constituency structure is indeed bounded by a small number. Where links of depth greater than 1 may be needed, it is shown that either the analysis is debatable, or that the problem is better handled by extending the common cover link representation in ways other than allowing links of depth greater than 1. Since the need for these extensions is independent of the link depth issue, this suggests that links of depth 0 and 1 are probably sufficient.

The final section of this chapter, section 4.5, describes certain linguistic structures which require the extension of the common cover link representation. These include non-projective dependency structures, coordination and relative clauses. The possible extensions are sketched but left to be fully specified in future work.

4.1 Dependency Structures

Dependencies describe the syntactic structure of utterances by a set of (possibly labeled) directed links between pairs of words. Every such link connects a *head* word to a word which is a *dependent* of that head. This dependent may itself be a head with its own dependents. The syntactic structure of a sentence is then given by a directed (possibly labeled) graph where each directed link points from a head to one of its dependents (see examples below). A basic relation induced by dependencies is the domination relation: a word x *dominates* a word y in a dependency structure if the structure contains a directed path from x to y.

Notation I write $D(U) \subseteq U \times U$ for the set of dependencies assigned to the utterance U. If $\langle h, d \rangle \in D(U)$ then there is a dependency from the head h to the dependent d. This is a directed graph whose vertices are the words of U and whose edges are the dependency links. For $x, y \in U$, I write $x >_{D(U)} y$ if x dominates y in $D(U)$ (in most cases this will indeed be a strict order).

Despite the long history of dependency based syntactic analysis, there continues to be considerable disagreement between dependency based theories as to which dependency relations hold for various linguistic utterances. For example, while all dependency theories agree that a determiner has a dependency relation with the noun it belongs to, there is no agreement as to which is the head and which is the dependent of this relation. Many, such as Mel'čuk (1988), analyze the noun as the head and the determiner as the dependent while others, such as Hudson (1990), prefer the determiner as the head. In some cases, theories even disagree as to whether dependency relations are sufficient to describe certain syntactic constructions, such as coordination (*John and Mary*). While Mel'čuk (1988) sees the first conjunct (*John*) as the head of this construction, Hudson (1990) claims that coordination and dependency are different relations altogether and must be represented differently in the syntactic theory. Even though these differences are significant, they are mostly irrelevant for the comparison with common cover link structures.

4.1.1 No-Loop Conditions on Dependencies

Despite the many differences among them, almost all dependency theories require that the dependency structure $D(U)$ be a directed acyclic graph (DAG). This means that there cannot be a directed path in the graph which leads from a word to itself (in particular, a dependency link may not connect a word to itself). This is equivalent to saying that the domination relation is a strict order. This requirement is strengthened in many theories, such as Mel'čuk (1988), to the requirement that the dependency structure must be a *rooted directed tree*. This means that the (undirected) graph of $D(U)$ is a tree and there is a *root word* (typically the main verb) which dominates all other words in the utterance. In particular, every word except the root word is a dependent of exactly one head word.

Regardless of the theory being used, rooted trees are by far the most common dependency structure. It seems that the main reason for relaxing the rooted tree requirement is the need to find a satisfactory representation for relative clauses. Theories which impose the rooted tree condition are forced to assign the following dependency analysis to a sentence containing the relative clause *he read*:

$$\begin{tikzcd}
\text{the} & \text{book} \arrow[l] \arrow[rr, bend left] & \text{he} & \text{read} \arrow[l] & \text{was} \arrow[lll, bend right] \arrow[r] & \text{lost}
\end{tikzcd}$$

In this analysis, *he read* is seen as a modifier of *the book*. Though this saves the rooted tree condition, it does so at the expense of hiding the important fact that *the book* is the object of the verb *read*. When the rooted tree condition is relaxed, the following dependency structure may be assigned to better describe the relation between *read* and *the book*. This analysis follows, for example, the annotation guidelines of the Corpus Gesproken Nederlands (Hoekstra et al. 2003).

$$\begin{tikzcd}
\text{the} & \text{book} \arrow[l] & \text{he} & \text{read} \arrow[l] \arrow[ll, bend right] & \text{was} \arrow[lll, bend right] \arrow[r] & \text{lost}
\end{tikzcd}$$

If we look at the dependency links assigned to the word *read* under the first analysis, they are very different from the dependency links assigned to this verb (by all theories) in the sentence *He read the book*:

$$\begin{tikzcd}
\text{he} & \text{read} \arrow[l] \arrow[rr, bend left] & \text{the} & \text{book} \arrow[l]
\end{tikzcd}$$

Under the second analysis, though, the links are exactly the same, except for the linear position of the object *the book*.

Such relative clause examples (and other similar examples) are the main source of violations of the rooted tree condition by dependency theories.[1] These examples not only satisfy the directed acyclic graph condition but actually also satisfy a stronger condition: the graphs have no *undirected* loops, that is, the undirected graph (in which the direction of the dependencies is ignored) is a tree. I will refer to this as the *directed tree* condition (as opposed to the *rooted directed tree* condition described above). It seems that most dependency theories which impose the directed acyclic graph condition also satisfy the directed tree condition. The two conditions are not equivalent, of course, as can be seen in the following example which is a directed acyclic graph, but not a directed tree: $\begin{tikzcd} x \arrow[r] \arrow[rr, bend left] & y & z \arrow[l] \end{tikzcd}$.

4.1.2 Projective Dependency Structures

An additional condition which may be imposed on dependency structures is that they be *projective*. A dependency structure $D(U)$ is projective if for any $\langle h, d \rangle \in D(U)$ and any $x \in (h, d)$ (that is, x between h and d in the linear order of U), $h >_{D(U)} x$. It was discovered in the early 60's by Lecerf (1960) and Hays (1964) that this condition holds for many sentences in many languages. Depending on the specific linguistic theory, the projectivity condition may fail in different constructions. All theories agree, however, that this condition fails for some sentences. One construction in which projectivity fails is a simple English question:

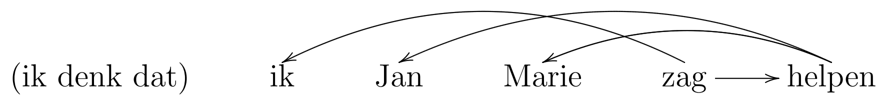
$$\begin{tikzcd} \text{what} & \text{are} \arrow[r] \arrow[rr, bend left] & \text{you} & \text{doing} \arrow[lll, bend right] \end{tikzcd}$$

[1]Hudson's Word Grammar (Hudson 1990) allows violations of the directed acyclic graph requirement but distinguishes (in a later version of the theory, Hudson 2003) between dependencies which are in the surface structure and other dependencies. It seems that the surface dependencies do not violate even the rooted tree requirement. For example, the relative clause example given above is assigned the following dependency structure, where only the links above the sentence and at the sentence level are part of the surface structure.

$$\begin{tikzcd} \text{the} & \text{book} \arrow[l] \arrow[rr, bend left] & \text{he} & \text{read} \arrow[l] \arrow[ll, bend left] & \text{was} \arrow[llll, bend right] \arrow[r] & \text{lost} \end{tikzcd}$$

When all dependencies are taken into account, many of Hudson's examples (not only relative clauses) violate the directed acyclic graph condition. It seems that Hudson's notion of dependency codes many relations which are not part of other dependency theories. It is, therefore, probably best to compare standard dependencies only to Hudson's surface dependencies.

A more complicated violation of projectivity can be found in Dutch:



No dependency based theory I am aware of has an analysis of these structures which is projective.

While it is not claimed by anyone that projectivity holds for all sentences, everybody agrees that it holds for many sentences (and for large parts of most sentences). As a result, projectivity can be used as a simplifying assumption about dependency structures without losing too much in the accuracy of the model. This assumption is adopted by many computational models papers (Klein and Manning 2004; Nivre and Scholz 2004; McDonald et al. 2005).

One reason why projectivity is often used as a simplifying assumption is that there is a strong connection between projectivity and bracketing structures. Projectivity holds if and only if, for every head, the head and all the words it dominates form a sequence of consecutive words in the utterance. This means that, given a projective dependency structure, it is possible to construct a bracketing by creating, for each head, a bracket covering the head and all words which it dominates. Given this bracketing, it is clearly easy to reconstruct the original dependency structure (the head of each bracket is the word of depth 0 under the bracket). This shows an equivalence between a subset of bracketing structures and projective dependency structures. In contrast, when a dependency structure is not projective it is not possible to construct an equivalent bracketing in the way outlined above because there is some head h such that a bracket covering h and all words it dominates also covers additional words. For example, in the English question *what are you doing* (whose non-projective dependency structure is given above) the bracket covering *doing* and all its dependents (*what*) must cover the whole sentence, just like the bracket covering *are* (the root of the dependency tree) and all its dependents. In the resulting bracketing, [[*what*] *are* [*you*] *doing*], it is no longer possible to determine whether it is *are* or *doing* which is the root of the dependency structure.

4.2 Dependencies and Common Cover Links

From the basic mathematical properties of dependency structures described in the previous section it is already clear that shortest common cover link sets are not dependency structures. This can be seen in the shortest common cover link set in figure 4.1(b), which contains both a directed loop (from *the* to *boy* and back) and an undirected loop (*know*, *sleeps*, *boy*, *the*, *know*). Despite this formal difference, it is apparent from figure 4.1 that there are also many similarities

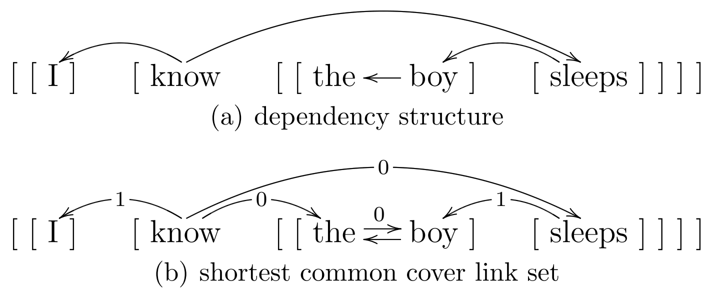

[[I] [know [[the boy] [sleeps]]]]
(a) dependency structure
0
1 0 0 1
[[I] [know [[the boy] [sleeps]]]]
(b) shortest common cover link set

Figure 4.1: A (possible) dependency structure and a shortest common cover link set of the sentence *I know the boy sleeps.*

between the two syntactic representations. In particular, every link in the dependency structure in figure 4.1(a) is also a link in the shortest common cover link set in figure 4.1(b). It is therefore worthwhile to look more closely at the similarities and differences between the two representations. Section 4.2.1 explains why dependencies are probably always common cover links and often (though not always) belong to a shortest common cover link set. The following three sections describe the three main differences (all seen in figure 4.1) between dependency structures and shortest common cover link sets:

1. Exocentric constructions (section 4.2.2): constructions with more than one head, such as the noun phrase *the boy* in figure 4.1(b), where there are links going back and forth between two words.

2. Link Depths (section 4.2.3): common cover links have a depth assigned to them.

3. Adjacency (section 4.2.4): the link from *know* to *sleeps* in figure 4.1 requires a common cover link path (but no dependency path) from *know* to each word between *know* and *sleeps.*

Of these three, adjacency is by far the most important and is the main property which enables incremental parsing and simplifies learning (as described in chapter 6).

Because the definition of common cover links is based on bracketing, common cover links can only describe projective dependency structures and the comparison in the present section will be restricted to such structures. While projectivity is a reasonable first approximation of linguistic structure, it is clearly inadequate for the description of many common constructions. The question of how the common cover link representation may be extended to handle such non-projective structures will be discussed (but not solved) in section 4.5.

4.2.1 Are Dependencies Common Cover Links?

In dependency structures, every linguistic unit consists of a head word and its dependents. Every such unit is a constituent and there is, therefore, a bracket which covers the head and all its dependents. I will call this bracket the *head's maximal bracket* (which may differ somewhat from a head's maximal projection as defined in Government and Binding theory, see Haegeman 1994). If the head is of minimal depth under its maximal bracket then there is a common cover link (in $R_{\mathcal{C}}$) from the head to every other word in the bracket and, in particular, every dependency link is also a common cover link.

To see whether every dependency link is also a common cover link, it remains to check whether the head is indeed always of minimal depth under its maximal bracket. Since every dependent is itself a head of a constituent, every dependent is inside its own bracket and cannot be of depth less than 1 under the head's maximal bracket. This means that as long as the head is of depth at most 1 under its maximal bracket, it is of minimal depth under it. For this to hold, there should be at most one bracket which contains the head but not all its dependents.

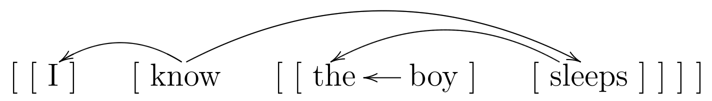
We can distinguish between *internal* dependents of a head, which are inside the smallest bracket covering the head, and *external* dependents which are not inside this bracket. As long as there is only one level of external dependents (that is, a bracket covering the head and some of its external dependents must cover *all* external dependents) the head remains of depth at most 1 under its maximal bracket and there are common cover links from the head to all its dependents. If, however, there are several levels of external arguments and brackets may cover the head together with only *some* of its external dependents, the head may be of depth 2 or higher under its maximal bracket. It may then be that one of the external dependents is of smaller depth than the head under the head's maximal bracket. In this case, there is no common cover link from the head to that dependent but, instead, a link from the dependent to the head. If such cases exist, not every dependency is also a common cover link.

The question remains whether heads may have external dependents at all and if yes, whether there may be more than one level of external dependents. The answer may be theory dependent, but at least in one case, that of the subject of a sentence, most theories seem to agree that the subject of a sentence is not inside the verb phrase, the smallest bracket covering the main verb of the sentence. It is also usually agreed that the subject of a sentence is a dependent of the main verb (or other predicate) of the sentence (but see section 4.2.2 for an alternative analysis). Under this standard analysis, the subject is an external dependent of the main verb (see Haegeman (1994) p. 72). Beyond this example, I am not aware of any other case where there is general agreement that a dependent is external or where more than one level of external dependents is allowed. One can, however, find annotated corpora where the annotation does actually distinguish between different levels of external dependents (that is, there are brackets covering the head with only *some* of its external dependents). A detailed analysis of several such examples will be carried out in section 4.4. All these examples seem to allow reasonable alternative analyses which require at most one level of external dependents.

To conclude this discussion, it seems that either most or all dependencies are also common cover links. The only exceptions may result from constructions where a head has more than one level of external dependents. If they exist at all, such constructions are relatively rare. The question whether they exist is strongly related to the question whether there are common cover links of depth greater than 1. This is discussed in section 4.4.

Are Dependencies in the Shortest Common Cover Link Set?

While dependency links are also common cover links (of various depths) the opposite is not true and it is easy to see that there are many common cover links which are not dependency links. It is, however, more interesting to compare dependency links not to the full set of common cover links (R_C) but to the shortest common cover link sets (which are actually used for parsing). Often, dependency links appear in a shortest common cover link set, but this is not necessarily always so. Sometimes this also depends on the dependency analysis chosen. For example, if we adopt an analysis in which the determiner and not the noun is the head of a noun phrase then the dependency structure for the example in figure 4.1 becomes:



The dependency link from *sleeps* to *the* is a common cover link in the set R_C of this sentence but does not appear in the shortest common cover link set given in figure 4.1(b) (it also does not appear in any other shortest common cover link set, because of the link from *sleeps* to *boy*).

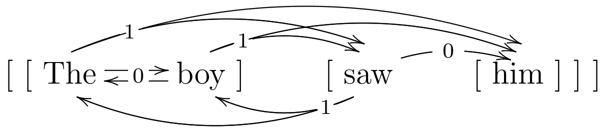
Despite such examples, the agreement between dependency structures and shortest common cover link sets is significant and most dependencies do appear in shortest common cover link sets. The following sections describe the most important differences between these two structures.

4.2.2 Exocentric Constructions

The first difference between the shortest common cover link sets and dependencies is that in some constructions there are back-and-forth common cover links between words or constituents (something which is not possible with dependencies). One place where this happens is in the structure of the smallest multi-word constituents, namely, noun phrases. In such a constituent, every word has a common cover link to every other word in the constituent and therefore the *shortest* common cover links connect every word to the word adjacent to it inside the constituent. This means that there are back-and-forth links between every two adjacent words in the constituent. In the simplest case, a noun phrase such as [DT N] has the shortest common cover link structure [DT $\rightleftarrows$ N] but its dependency structure is either [DT $\rightarrow$ N] or [DT $\leftarrow$ N] (depending on the linguistic theory). This can be seen in the structure of the noun phrase *the boy* in figure 4.1. This common cover link structure cannot be a dependency structure under any linguistic theory because it violates the directed acyclic graph condition.

The common cover link structure assigned to noun phrases agrees well with the difficulty linguists have in deciding whether it is the determiner or the noun which is the head of the noun phrase. It has even been recently suggested (Beavers 2003) that a noun phrase has no single head, that is, that noun phrases are exocentric constructions. The common cover link structure supports this point of view by giving both the determiner and the noun an equal status. If linguists find it difficult to agree which is the head of a noun phrase, wouldn't it also be easier for a learning algorithm if it doesn't have to decide which is the head? The common cover link representation allows a learner to avoid making this decision. This difference between the dependency and common cover link analysis of noun phrases may seem small, but because noun phrases are the most frequent (and probably also most important) building blocks of a sentence, this translates into numerous differences in the actual assignment of links.

A similar difference between dependencies and the shortest common cover link sets can be observed in the attachment of a subject to a verb phrase. In dependency structures, there usually is a dependency link from the verb to the subject. In the common cover link set ($R_{\mathcal{C}}$) there are common cover links of depth 1 both from the subject to the verb and from the verb to the subject:



Whether links in one or both directions are selected for the representative subset being used (and therefore also for the shortest common cover link set being used) is not specified by the representation itself but remains to be decided separately (by the linguistic theory or parser). This means that the common cover link representation can either have only a link from the verb to the subject or have links going back and forth between them. In the first case, this results in an analysis of the subject as an external argument of the verb, in agreement with dependency based theories as well as many other modern linguistic theories (e.g. Haegeman (1994) p. 71). In the second case, a more traditional linguistic analysis is adopted, in which the sentence is an exocentric construction resulting from the combination of a subject with a predicate as two elements of equal status, none of which dominates the other (see e.g. Potter (1950) p. 91 or Hudson (1987) p. 113).

As these two examples show, common cover links allow an exocentric representation of linguistic structures, something which is not possible in dependency structures. Some exocentric constructions in a shortest common cover link set are determined by the bracketing represented by the link set while others are optional. This has to do with the free choice involved in the definition of a representative subset (section 2.4): when the exocentric construction is of common cover links of depth 0, the construction is determined by the bracketing while exocentric constructions of links of non-zero depth are optional and depend on the choice of representatives. In this sense, bracketing lies between dependency structures and shortest common cover link sets: exocentric constructions of links of depth 0 can be represented by a bracketing while exocentric constructions of links of non-zero depth cannot.

To see how this is reflected in the bracketing, let us first consider a simple bracket covering two words. If both these words are of depth 0 under the bracket (as in $[x\ y]$) then there must be depth 0 back and forth links between the words, $[x \overset{0}{\rightleftarrows} y]$, and the structure is exocentric. If, on the other hand, there is an additional bracket around one of the two words, as in $[x\ [y]]$, then there is only a link from x to y and the structure is not exocentric ($[x \xrightarrow{0} [y]]$). In both of these cases the bracketing determines whether the structure is exocentric. If there are brackets covering each of the two words (as in $[[x]\ [y]]$) then the links connecting the two words are of depth 1 and either one or both words may be chosen as representatives of the bracket. If both words are chosen as representatives ($[[x] \overset{1}{\rightleftarrows} [y]]$) then the structure is exocentric while if only one word is chosen as the representative ($[[x] \xrightarrow{1} [y]]$) the structure is not exocentric. Here the bracketing does not indicate whether the construction is exocentric or not. This is one of the cases where shortest common cover link sets are more expressive than bracketing.

It should be stressed that the common cover link representation (as such) allows exocentric structures but does not define which structures actually are exocentric. This remains to be determined by the specific linguistic theory (or parser) which is used to assign the syntactic structure. It may be that exocentric structures do not exist at all and that dependencies are indeed sufficient for describing any syntactic structure. Such a conclusion must be the result of careful linguistic inquiry and not merely the by-product of a choice of constraints on the representation (however appealing their simplicity may seem). The fact that linguists have come up with exocentric analyses of syntactic structures suggests that this option should remain open in the formalism being used. Moreover, the difficulty linguists have in determining the head of structures such as noun phrases shows that the answer to this question is not simple. Indeed, it may be that there is no single answer and that different levels of linguistic analysis result in different heads (see Zwicky (1985) and Hudson (1987)). The learning algorithm of the next chapter is able to detect many head-dependent relations (such as those between a verb and its object or a preposition and its argument) but assigns an exocentric structure to noun phrases. This may be seen as empirical evidence suggesting an exocentric analysis of noun phrases at least at the basic level of syntactic analysis which is distributional rather than semantic.

4.2.3 Common Cover Link Depth

The second difference between common cover links and dependencies is that each common cover link has a depth. These depths impose restrictions on the possible link configurations through the characterizing properties of Definition 2.7.1. In contrast, most dependency based formalisms label dependency links by labels which represent different syntactic or semantic relations (such as *subject of*, *agent of* or *purpose of*, depending on the specific theory). There may be some correlation between the depth of common cover links and the labels assigned to dependencies in some theories, but I am not aware of any dependency based system in which the labels have the same implications as the depths of common cover links have for possible allowed structures.

While having links of different depths is certainly necessary if we want to be able to describe all possible bracketings, one may wonder whether the depth is also necessary when only bracketings describing syntactic structure are considered. The answer was already given in the previous section which describes the common cover links joining the subject of a sentence with the verb phrase. Whether one adopts an exocentric or dependency based analysis of this relation, the subject and the verb phrase are joined by links of depth 1. This is important when calculating the bracketing (constituency) structure associated with a certain link structure. As can be seen in the example of figure 4.1(a), the bracketing of the sentence *I know the boy sleeps* cannot be calculated from the unlabeled dependencies because there is no way of knowing that the subject *I* should be outside the minimal bracket covering the verb *know* while the object *the boy sleeps* should be inside it. Of course, if the dependency links from the verb *know* were labeled as *subject* and *object* then it would have been possible to add a rule which specifies that the *subject* is an external argument (and should not be inside the minimal bracket covering the verb *know*) while the object is and internal argument (and should be inside that bracket). This is equivalent to adding a depth property to every dependency link.

Apart from joining the subject and the verb phrase, are there additional constructions which require links which are of non-zero depth? This is not entirely clear. In some annotation schemes (such as the one used in the Wall Street Journal Corpus) one can find bracketings such as [[*the chairman*] [*of* [*the board*]]] of a noun phrase with a modifier. This sort of bracketing requires links of depth 1.

The bracket [*the chairman*] is not, however, usually accepted as a constituent because it fails most constituency tests (it cannot be replaced by a pronoun such as *it* or be moved). Therefore, if we assume (as I do) that the bracketing represents constituency structure then the correct bracketing should not mark *the chairman* as a constituent and links of depth 0 are sufficient to represent the structure. I return to this discussion in section 4.4.

4.2.4 Adjacency

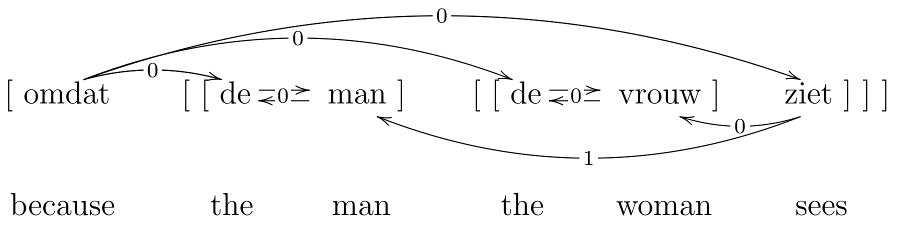
The most fundamental difference between dependency structures and shortest common cover link sets is the result of the *connectedness* characterizing condition in Definition 2.7.1. In dependency structures, when a head h dominates a constituent D, there is a dependency link from h to the head d of D and this is the only link from h to D. In contrast, the connectedness condition on shortest common cover link sets requires that if there is a link from h to d then there must also be a path of links from h to every word between h and d. In particular, there is a common cover link from h to the word in D which is closest to h. This word need not necessarily be the head of D. We can see this in figure 4.1, where the shortest common cover link set contains common cover links from *know* to both *the* (the closest word in the constituent to *know*) and *sleeps* (the head of the constituent). In contrast, there is a dependency link from the head *know* only to the dependent *sleeps*, which is the head of the constituent *the boy sleeps*.

Whether the shortest common cover link set contains links from h to D other than to the word in D closest to h depends on the structure of D. If, for example, D is a minimal bracket (a noun phrase), there is a single link from the head h to the word inside D closest to h. This means that while a verb has a common cover link to the *noun* in the subject constituent, it has a common cover link to the *determiner* in the object constituent:

$$\begin{tikzcd}[column sep=small]
{[\,[\,DT} \arrow[r, shift left, "0"] & {N\,]} \arrow[l, shift left] & {[\,V} \arrow[l, bend right, "1"'] \arrow[r, bend left, "0"] & {[\,DT} \arrow[r, shift left, "0"] & {N\,]\,]\,]} \arrow[l, shift left]
\end{tikzcd}$$

This is not surprising if one recalls that the common cover links cannot determine whether it is the determiner or the noun which is the head of the noun phrase. Therefore, any of the two words will do for attachment and the closest one to the verb is chosen. With dependency links, such an attachment is not possible. Once the theory we use has determined whether it is the determiner or the noun which is the head of the noun phrase, the dependency links from the verb to the subject and the object must both attach to the determiner or both attach to the noun.

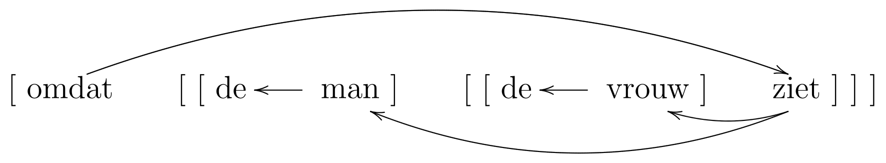
When the head d of the constituent D is the only word of minimal depth in D, the shortest common cover link set must contain a link from h to d. This then results in a structure similar to that in figure 4.1(b). An additional example of this form can be found in Dutch, where the phrasal complement of *omdat* (*because*) is to the right of *omdat* but has its own dependents to the left of the head (the verb *ziet*):



In this example, there is a link from the head *omdat* to the leftmost word of each dependent inside the constituent headed by *ziet* (the phrase *de man de vrouw ziet* which means *the man sees the woman*).

This structure of shortest common cover link sets is behind the adjacency condition (Definition 3.2.1) used by the incremental parser. The most important property this condition imposes on the parser is that the parser needs to consider adding a link from x to y only if there is already a path of links from x to all words between x and y. In the above Dutch example, this means that as the arguments of the verb *ziet* are read, they are attached to *omdat* even though they are not dependents of *omdat*. When the verb *ziet* is read, it is adjacent to *omdat* and a link from *omdat* to *ziet* can be created. This adjacency has not only a formal definition but also an intuitive meaning: the verb *ziet* is immediately next to a syntactic unit (already constructed by the incremental parser) which is "headed" by *omdat* and contains the two constituents *de man* and *de vrouw*. It can therefore be attached to this syntactic unit and this may be done by a link from the "head" of the unit. At the same time, a link from *ziet* to its closest dependent, *de vrouw*, can be created. Once this link is created, the next dependent, *de man* becomes adjacent and can be attached to the verb.

In this way, the attachment of the sentential complement of *omdat* is performed in several steps, with the words attached one by one under the head *omdat*. Even before the full complement is read, the parser already determines that whatever is to the right of *omdat* is part of its complement. This can be done without actually knowing what the head of the complement is. When a dependency representation is used, this is not possible because there is only one dependency link from the head *omdat*: to its dependent *ziet*.



When the words are read one by one, the parser can only add the dependency link from *omdat* to *ziet* when all words have been read. Therefore, before the verb *ziet* is read, the different elements of the phrase remain unattached to each other. This seems to be different from what we feel happening when hearing such a phrase. Consider a similar English example: *because the man saw the woman*. When the speaker of this phrase pauses after uttering only the first three words, *because the man*, we already feel that *the man* is part of the reason described by the clause, even though the head of this complement (the verb *saw*) has not yet been heard and therefore the dependency from *because* to the clause has not yet been created. In this sense it seems that incremental parsing with common cover links is more similar to what we feel happening when humans parse such phrases.

Such an argument, based on introspection, cannot be taken as evidence for anything, but may serve to motivate closer inspection of the issues involved. These were studied extensively by psycholinguists interested in sentence processing. One phenomenon studied in this context is known as *reanalysis*: the seeming ability of humans to switch from one syntactic analysis to another while processing a sentence incrementally. The next section shows how cases of easy reanalysis, where humans do not seem to have much trouble changing from one analysis to another, do not actually involve any change in the analysis when parsing with common cover links.

4.3 Reanalysis

In the psycholinguistic literature on sentence processing, *reanalysis* refers to situations where the human incremental parser has to modify decisions made in previous stages of the parse. In the following three examples (taken from Sturt and Crocker 1996), the syntactic analysis of the sentence remains ambiguous until the word in boldface is read. Psycholinguistic experiments show that before reaching the disambiguating word, people tend to adopt the globally incorrect reading and must therefore modify their analysis when the disambiguating word is reached:

(1) a. John knows the truth **hurts**.

b. While Philip was washing the dishes **crashed** onto the floor.

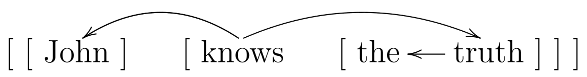
c. The boat floated down the river **sank**.

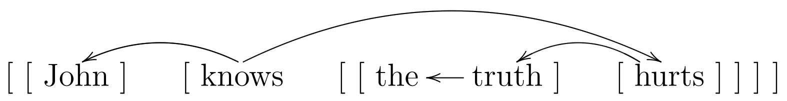
While the first sentence (1a) does not seem to cause any difficulty, sentences (1b) and (1c) cause considerable difficulty in reading. These two are *garden path* sentences, where the prefix of the sentence (before the disambiguating word) leads the reader down an incorrect analysis.[2] To correctly interpret the garden path

[2]Dick de Jongh suggested the following explanation for the term *garden path sentence*: *Alice walked down the garden path after path attracting her attention.*

sentences, the reader must *consciously* reanalyze them when the disambiguating word is reached and may have to restart processing from the beginning. The measurable difficulty people have processing garden path sentences serves as empirical evidence for the incrementality of human language processing.

While sentence (1a) is not a garden path sentence and does not seem to cause any processing difficulty, its prefix is also ambiguous until the word *hurts* is reached. This is because the verb *knows* can take either a noun phrase or a sentence as a complement. After the prefix *John knows the truth* has been read, it is not yet known whether *the truth* is the noun phrase complement of *knows* or merely the subject of the sentential complement *the truth hurts.* In terms of dependencies, these are two very different analyses, because the first requires a dependency to be created from *knows* to *the truth* while the other does not:





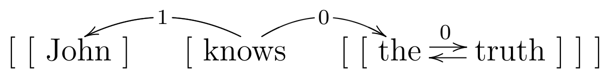
What, then, should an incremental parser do after reading the prefix *John knows the truth*? If it assigns a dependency link from *knows* to *the truth*, it may have to undo this link when the verb *hurts* is read. If it decides not to add this link, it fails to assign a correct parse to *John knows the truth* in case this is the complete sentence. One possibility is that the prefix *John knows the truth* is first parsed to have *the truth* as a noun phrase complement of *knows* and that this parse is reanalyzed if and when the verb *hurts* is read. Experiments in (Sturt et al. 1999) indeed detect a certain delay in reading when the disambiguating word (*hurts*) is reached in this sort of sentences. This is *unconscious* reanalysis which people can perform without difficulty (the measured delay is small) as opposed to the *conscious* reanalysis involved in processing garden path sentences.

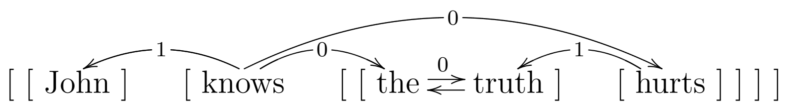
To explain the difference between the two types of reanalysis, conscious and unconscious, researchers have suggested different models which allow the modifications needed for unconscious reanalysis but exclude the modifications needed for conscious reanalysis (Abney 1989; Pritchett 1992; Lewis 1993). An alternative adopted by others is to look for underspecified representations of syntactic structure such that parsing decisions do not have to be undone in cases of unconscious reanalysis (but do have to be undone in cases of conscious reanalysis). This approach is attractive because it derives the restrictions on structural modification from the syntactic representation being used.

Many of the suggestions along this line are based on description theory (D-theory), first proposed in Marcus et al. (1983). D-theory is a framework for describing syntactic trees which takes *domination* rather than *direct domination* as the basic relation for specifying the tree structure. Several authors (Weinberg

1993; Weinberg 1995; Gorrell 1995a; Gorrell 1995b; Sturt and Crocker 1996), have used this basic framework to put forward their own models for sentence processing. The use of domination (rather than direct domination) to describe tree structures can explain the ease with which the unconscious reanalysis takes place in processing *John knows the truth hurts.* When parsing the prefix *John knows the truth*, the parser only determines that *knows* dominates *the truth* but does not determine whether this is also direct domination. Since *knows* dominates *the truth* both in *John knows the truth* and in *John knows the truth hurts*, the parser does not need to change this decision when the word *hurts* is reached. This explains why reanalysis in this case is easy. In the garden path sentence (1b), on the other hand, no such easy solution is available. After having read the prefix *while Philip was washing the dishes*, the parser must decide whether *washing* dominates *the dishes.* The initial parse preferred by humans (and which is often correct) has *the dishes* as an object of *washing.* Therefore, a domination relation must be created from *washing* to *the dishes.* However, when the disambiguating word *crashed* is reached, the analysis has to be altered so that *the dishes* are the subject of *crashed* and *while Philip was washing* is a modifier of *crashed.* In this analysis, *washing* does not dominate *the dishes* anymore. As a result, even when working with the weaker relation of domination, this example requires the parser to undo previous parsing decisions. This explains why this is a garden path sentence which requires conscious reanalysis.

Common cover link parsing has much in common with D-theory based models in that the relation represented by common cover links is closer to domination than to direct domination (but the relation is not actually the domination relation, as evidenced by the possibility of directed common cover link loops). This then leads to the same difference in reanalysis between the examples given above. Consider the common cover link structures of the sentences *John knows the truth* and *John knows the truth hurts*:





The first four words, *John knows the truth*, are connected by the same links in both structures. This means that when the word *hurts* is reached, only the two links connecting it to the prefix of the structure need to be added. Incrementality is preserved and reanalysis is easy (in terms of common cover links there is actually no reanalysis). The garden path sentence (1b), on the other hand, causes common cover link parsing the same problem it causes D-theory based approaches. At first, a common cover link is created from *washing* to *the dishes* to allow the analysis in which *the dishes* is the object of *washing.* Then, when *crashed* is reached, this link needs to be removed. This is not allowed in incremental parsing, in agreement with the difficulty experienced by humans when processing this sentence.

I have only discussed very few examples here. Reanalysis comes in many different forms (and languages) and its difficulty may depend not only on syntactic structure but also on meaning, priming, length of phrases and other factors. It was not my intention to present common cover links as a cognitive model of language processing. Rather, I wanted to show that incremental parsing with common cover links can handle (at least some) initial ambiguities in sentence structure when these do not seem to pose problems for humans. This means that the incrementality of the parser does not in itself form an obstacle to correct parsing. Of course, the parser may still fail on garden path sentences, but, as these are also difficult for humans, we may expect them to be rare.

4.4 Are Links of Depth 0 and 1 Sufficient?

In section 4.2.1 we saw that if every dependency head is of depth at most 1 under its maximal bracket (the smallest bracket covering it together with all its dependents) then every dependency link is also a common cover link. This holds if every dependency head has at most one level of external dependents, that is, there is at most one bracket which covers the head but does not cover all its dependents. If this assumption holds and if we accept the basic assumption behind dependency structures, that every syntactic constituent has a head word, then all common cover links are of depth 0 or 1. In section 4.2.2 we saw, however, that common cover links allow us to describe constituents which are exocentric and do not have a head word. While such exocentric constructions may, theoretically, be a source of common cover links of depth greater than 1, the examples of exocentric constructions given in section 4.2.2 do not require such links. This was because those exocentric constructions all had at least one component which had a head with no external dependents (and therefore had a head of depth 0). In fact, in the only case which definitely seems to require an external dependent (the subject of a sentence) the exocentric construction was an alternative to the external dependent construction.

It is, therefore, an appealing hypothesis that common cover links of depth 0 and 1 are sufficient to describe all syntactic structures. In terms of dependencies, this hypothesis is equivalent to assuming that no head has more than one level of external dependents (as defined above) and that exocentric constructions all have at least one component which has a head and no external arguments. Whether this holds is an empirical linguistic question: there are bracketings which require links of depth greater than 1 but it is not clear whether these bracketings are possible descriptions of linguistic structure. This also depends on what the brackets represent. Throughout this work I assume that brackets represent linguistic constituents.

The answer to this question is, of course, theory dependent, but, to give a first rough assessment of the possible constructions which need to be examined, I looked at the bracketings assigned to sentences in several syntactically annotated corpora. While these corpora represent only one form of annotation of specific domains of specific languages, they all contain complex sentences which are likely to reveal complex syntactic structures needing link depths greater than one, if such structures exist.

For these experiments I used the same corpora used in the experiments of chapter 7: the Wall Street Journal Corpus (version 2.0) for English, the Negra Treebank (version 2.0) for German and the Chinese Treebank (version 5.0) for Chinese. Each bracket in the annotation was considered equivalent to the set of words it covers (see details in section 7.2). I then calculated the height of each bracket (Definition 2.2.4). This height is equal to the depth of the links needed to represent the bracket.

Of the 730,024 brackets in the Wall Street Journal Corpus which cover more than one word, 1,693 were of height 2 and none were of height 3 or higher. Of the 141,977 brackets in the Negra Corpus which cover more than one word, 11 were of height 2 and none were of height 3 or more. Finally, of the 330,079 brackets in the Chinese Treebank which cover more than one word, 17,382 were of height 2 and 253 were of height 3. No brackets were of height 4 or more. Considering that many of the sentences in these corpora are long and contain complex constructions, these numbers seem to suggest that the height of brackets in syntactic structures is bounded by a small number. It is also clear that there are considerable differences between the different corpora. While the number of brackets of height 2 in the Negra Corpus is negligible and the number of brackets of height 2 in the Wall Street Journal Corpus is very small, around 5% of the brackets in the Chinese Treebank are of height 2. These differences may be due to real differences between the languages (and domains) or to differences between annotation schemes. The very low number of brackets of height 2 in the Negra Corpus is probably the result, among other things, of the fact that this corpus was originally annotated with dependency structures and only then converted into bracket annotation.

A closer look at the brackets of height 2 (or more) reveals that most of them fall into several distinct categories. I here describe these categories and discuss their analysis. For each category type I report below the number of brackets of height 2 or more which remain after removing brackets of that category.

Coordinator-less coordination This is a sequence of parallel phrases or sentences which could have been joined by a coordinator (such as *and*) but instead were only separated by punctuation (comma, semicolon or full stop). Only coordinations without a coordinator create a bracket of height 2 because when a coordinator is used, the coordinator itself is of depth 0 under the bracket. Coordinator-less coordinations account for the majority of brackets of depth greater than 1. Many of these are coordinations of full sentences. Ignoring the full sentence coordinations, there were only 922 brackets of height 2 in the Wall Street Journal Corpus, 5 brackets of height 2 in the Negra Corpus and 10944 brackets of height 2 in the Chinese Treebank. There were also 8 brackets of height 3 in the Chinese Treebank. Ignoring all coordinations without a coordinator (such as coordinations of two NPs separated by commas) there were 99 brackets of height 2 left in the Wall Street Journal Corpus, no brackets of height 2 in the Negra Corpus and 6542 brackets of height 2 in the Chinese Treebank. There were no brackets of height 3 or more left in the Chinese Treebank after this step.

As already mentioned in section 4.1, linguists differ as to the analysis of coordination. The parallel nature of the coordinates in a coordination suggests to some linguists (Hudson 1990; Goodall 1987; Muadz 1991; Moltmann 1992) that the relation between these coordinates is different from the subordination relation which holds between heads and their dependents. If this approach (which I believe to be correct) is adopted, there should also be no common cover links between coordinates but an alternative relation should be used instead. This is certainly true when the coordination is between full sentences. Therefore, while the common cover link representation should be extended to properly handle coordination, I do not think that adding common cover links of depth greater than 1 is the solution and I do not consider coordination structures to be sufficient evidence against the hypothesis that all common cover links are of depth 0 and 1. I will discuss the possible extension needed to handle coordination in some more detail in section 4.5.2.

Preverbal modifiers in the Chinese Treebank The majority of the 6542 brackets of height 2 in the Chinese Treebank which remain after the previous step are a result of the way preverbal modifiers are annotated in the Chinese Treebank. In Chinese, some dependents of the verb appear before the verb and others after the verb. In addition to the subject, various modifiers of the verb can appear between the subject and the verb (this can also be found in English, but to a lesser extent). The annotators of the Chinese Treebank have chosen to annotate the subject and the preverbal modifiers as two different levels of external dependents of the verbs. This means that they are annotated at different bracketing levels and are outside the smallest bracket containing the verb. The following example of such a structure is taken from the Chinese Treebank annotation guide (Xue and Xia 2000):

```
(IP (NP-SBJ (PN 他们))
    (VP (ADVP (AD 常常))
        (VP (VV 到)
            (NP-OBJ (NN 公园))))
```

Because the adverb (ADVP) is outside the smallest VP (verb phrase) bracket, an additional VP bracket is created. The subject (NP-SBJ) is then attached to this VP by creating an additional bracket (IP). This means that the annotation has three different brackets covering the verb (VV) and its dependents and as a result the verb is of depth 2 under its maximal bracket (IP). When the subject and the preverbal modifier are themselves complex structures, the IP bracket may be of height 2. The question is whether this is the best analysis of these structures. In the Wall Street Journal Corpus we can find a similar structure:

```
( (S
    (NP-SBJ (DT The) (NN company) )
    (ADVP (RB also) )
    (VP (VBD adopted)
      (NP (DT an) (JJ anti-takeover) (NN plan) ))
    (. .) ))
```

Here, the bracketing is different from that given in the Chinese Treebank. In the English annotation, both the subject and the adverb are external dependents of the verb at the same level. As a result, the verb is at depth 1 under its maximal bracket S. There is, in fact, also a third possible analysis: the adverb may be included inside the lowest VP while only the subject remains outside it. I think this should be the preferred analysis because when the adverb in English follows the verb, it is always put inside the lowest VP. It seems that whether the adverb appears before or after the verb should not cause any fundamental change to the constituency structure. A similar case is mentioned in the Chinese Treebank annotation guide, where (on p. 25) it is stated that a preverbal QP (quantifier phrase) should appear outside the lowest VP while a postverbal QP should appear inside it. This is supposed to distinguish between obligatory dependents (arguments) which appear inside the lowest VP and optional dependents (non-arguments) which appear outside this VP. To keep the annotation decisions simple (p. 8) it was decided by the annotators of the Chinese Treebank that preverbal dependents are always optional while postverbal dependents are always obligatory. Whether this is a good criterion for optionality, I cannot say, but as the annotation of the English corpus shows, there is no reason for keeping optional dependents out of the lowest VP. There is, therefore, no need to create an extra VP level and the two VP levels can be collapsed into one. After doing so (together with removing coordinator-less coordination) only 530 brackets of height 2 remained in the Chinese Treebank.

Chinese verb compounds The annotation guide for the Chinese Treebank states (on p. 85) that verb compounds (verbs composed of several individual verbs) should be treated in the same way as monolithic verbs (composed of one word). At the same time, compound verbs are bracketed together while a single verb is not bracketed. Clearly, the bracketing of the compound verbs together is meant to indicate that these form together a unit, but such a unit is not a constituent (just as a single verb is not a constituent without its arguments). I therefore removed all brackets grouping compound verbs together. After doing so (in addition to the modifications described above), 455 brackets of height 2 were left in the Chinese Treebank.

Nouns modified by a relative clause in English Of the 99 brackets of depth 2 which remain in the Wall Street Journal Corpus after coordinator-less coordination brackets are removed, the great majority (73) are the result of a noun which has both a modifier and a relative clause attached to it. One variant of such a construction is:

```
(NP-SBJ
  (NP
    (NP (DT A) (NN form) )
    (PP (IN of)
      (NP (NN asbestos) )))
  (RRC
    (ADVP-TMP (RB once) )
    (VP (VBN used)
      (NP (-NONE- *) )
      (S-CLR
        (NP-SBJ (-NONE- *) )
        (VP (TO to)
          (VP (VB make)
            (NP (NNP Kent) (NN cigarette) (NNS filters) )))))))
```

In this example, the noun phrase *a form* is first modified by a modifier *of asbestos* and then by a relative clause. Each modifier adds a level of bracketing, resulting in a bracket of height 2. This analysis assumes that *a form* is a constituent which needs to be bracketed separately. However, as already mentioned above, the pair *a form* fails most constituency tests (such as substitution by *it* or movement) and is not usually accepted as a constituent. Moreover, just as the modifiers of a verb are inside the minimal bracket covering the verb, the modifiers of a noun may be claimed to be inside the smallest bracket covering the noun. In this case, the bracket covering the lowest level NP is extended to also cover the modifiers and is merged with the higher level NP. In the example above, this means that the resulting bracketing is [a form [of asbestos]]. Under this analysis, there is no bracket of height 2.

Remaining brackets of height 2 After removing the brackets covered by the four cases mentioned above, I was left with 26 brackets of height 2 in the Wall Street Journal Corpus, no brackets of height 2 in the Negra Corpus and 455 brackets of height 2 in the Chinese Treebank. The brackets were removed automatically, so some brackets which should have been removed by the three conditions above may have slipped through. For example, of the 26 brackets of height 2 remaining in the Wall Street Journal Corpus, 6 seem to be cases of coordinator-less coordination which were not removed automatically because of mistakes in the annotation or in the removal algorithm. Similarly, in the Chinese Treebank, preverbal modifiers were put into the lowest verb phrase, but similar constructions with non-verbal predicates were not modified.

The number of brackets of height 2 remaining in the Wall Street Journal was small enough for them to be examined individually. Of the 26 brackets, 6 are coordinator-less coordinations which were not removed automatically, 4 were addresses (which should probably not be assigned a syntactic structure at all) and 6 were cases where two arguments of a verb were put together into a bracket (which I believe to be an incorrect analysis). The remaining 10 cases contain several obvious annotation mistakes as well as some cases which I could not figure out. I believe none of these can be considered as conclusive evidence for brackets of height 2 being necessary in any construction.

With Chinese being all Greek to me and 455 brackets of depth 2 remaining, I was unable to perform a similar analysis of the Chinese Treebank. Instead, I checked how many of the remaining brackets of height 2 were caused by a bracket covering a single word. Because brackets covering only a single word do not serve to group words together into constituents, they may seem redundant. This is not entirely true because these brackets serve to distinguish between a head (which is not inside a single-word bracket) and its single word dependents (which are inside such a bracket). For example, in the verb phrase [leave [it]], the verb *leave* is the head while *it* is the dependent. The brackets covering a single word in the corpus may therefore either cover heads (and therefore be redundant) or cover dependents (in which case they are not redundant). However, one would expect that if there is a construction in a language which requires brackets of height 2 then this construction does not appear only with single-word dependents. Therefore, removing the single-word brackets should leave us with some height 2 brackets if they exist in the language.

Having removed brackets covering a single word (on top of the previous steps), only 8 cases of brackets of height 2 remained. Some of these seem to be coordinator-less coordinations (which slipped through the previous steps) while others I could not explain. As with the Wall Street Journal Corpus, I believe none of these examples can be considered as conclusive evidence for brackets of height 2 being necessary in any construction.

While this study of the corpus data is only preliminary and covers only a small number of corpora, I believe it provides strong indications that the heights of brackets needed to describe constituency structures is bounded by a low number. Whether the maximal bracket height is indeed 1 (as hypothesized) cannot be conclusively determined, but it does seem that many of the brackets of height higher than 1 in the corpora fall into a small number of constructions all of which have an analysis which may be debated. It therefore seems to me that before attempting to extend the common cover link algorithms to handle links of depth greater than 1 it is better to consider other possible modifications and extensions. Some of these are mentioned in the following section.

4.5 Extensions of the Common Cover Link Representation

The definition of common cover links was based on bracketing and chosen in such a way that a bracketing could be reconstructed from the links. The main aim was not to improve on the representation of the final syntactic structure but rather to allow it to be constructed incrementally. As a result, common cover links can be used to represent any bracketing, but not much more, thus inheriting the basic limitations of bracketing. Solutions to some of these limitations can be found in various systems of dependency structures. The link-based nature of the common cover links should make it easy to adopt some of these solutions to extend the common cover link representation. This should be done in a way which preserves the incrementality of the parser.

In this section I describe several linguistic constructions for which I find an extension of the current system necessary: non-projective structures, coordination and relative clauses. While non-projective structures are no problem for almost all dependency based systems, coordination and relative clauses are handled in fundamentally different ways by different dependency based theories. This shows that these remain problematic linguistic structures and it is not my intention to solve these problems here. I would like, however, to point out those solutions found in the literature which are best suited for extending not only the representation as such but also its learnability.

4.5.1 Non-Projective Structures

Probably the most obvious limitation of bracketing as a representation of syntactic structure is that without additional descriptive mechanisms it can only describe linguistic structure in terms of continuous constituents. This is equivalent to the projectivity restriction on dependency structure which was described in section 4.1.2. While projectivity holds for many sentences (and for large parts of most sentences), we have seen that even the structure of simple English questions is not projective.

The remedy to this problem is to remove the projectivity restriction. When working with dependencies, this is usually done by simply removing any restrictions on the linear order of the words in the dependency structure (leaving only restrictions on its structure as a directed graph). Because the common cover links are a link-based representation, it may seem that the projectivity restriction can be removed in the same way. This is not so simple, however, because we would like to preserve those properties of the common cover links which make incremental parsing and learning possible. This means, for example, that if a head h has a dependent constituent D then there should be a link from h to the word in D closest to h. I believe that the best way to define this is to first define noncontinuous bracketings. A noncontinuous bracketing has all the properties of a bracketing except that brackets are not restricted to sequences of consecutive words but may be arbitrary subsets of the words in the utterance. The noncrossing bracket condition must still be preserved and common cover links can be defined as before. It then remains to select an appropriate form of transitivity to remove redundant links. Because brackets are no longer continuous, linear transitivity is no longer sufficient. Depending on the version of transitivity chosen, different variants of the representation may result.

I did not carry out the mathematical analysis for common cover links defined in this way but I believe that it involves no fundamental problems. The main question is whether completely removing the projectivity restriction does not allow far greater flexibility than is actually available in natural language syntax. Since introducing unnecessary freedom into the representation makes parsing and learning more difficult, it is worthwhile examining the linguistic data for possible conditions which may restrict the possible structures.

4.5.2 Coordination

Coordination involves two or more structures which play a parallel role. The simplest forms of coordination involve either two nouns (such as *John and Mary*) or two full sentences (such as *John walks and Mary sings*). In these two cases, which coordinate either the smallest or the largest syntactic units, the coordinates are two independent units of the same syntactic type which are joined by a coordinator (such as *and*) to form a new unit with syntactic properties similar to those of the original units. In a bracketing this is typically described by putting each of the coordinates as well as the coordination as a whole into a bracket, resulting in structures such as [[*John*] *and* [*Mary*]]. If the brackets are labeled, the coordination as a whole receives the same label as each of the coordinates. This seems (at first) to appropriately represent the symmetry between the two coordinates.

In dependency structures, even the simplest coordination *John and Mary* is problematic because of the inherent asymmetry of the dependency relation. One way to solve this problem is to claim, as Mel'čuk (1988) does, that the relation between the two coordinates is not entirely symmetric. In this way, Mel'čuk arrives at the structure $John \rightarrow and \rightarrow Mary$ for coordination. Many linguists, such as Hudson (2003) find this analysis unsatisfactory and claim that dependencies describe a subordination relation which is inherently different from the coordination relation. Such dependency based theories can then no longer use a single relation to describe syntactic structure but must use two different relations: dependency (subordination) and coordination.

While bracket based descriptions of syntactic structure do not seem to have the same problems that dependencies have in describing simple coordination structures, brackets also run into problems when arguments are shared between coordinated units. The simplest example of such shared arguments is a shared subject, as in *John walked and laughed.* Since the subject is an external argument and is not inside the verb phrase, this sentence can still be handled quite easily by a bracketing which puts the two verb phrases together in a single bracket: [[*John*] [*walked and laughed*]]. Things become more complicated when the object and not the subject is shared, as in *John bought and Mary ate the chocolate* (these structures are known as *right node raising* structures). Here, the object *the chocolate* has to be inside both verb phrases and it is no longer possible to claim that a simple bracketing adequately describes the coordination structure. Things are further complicated by gapping constructions such as *John bought chocolate and Mary, flowers* in which a verb is shared by two sets of coordinated arguments. Examples such as these (and many others) have motivated also researchers working with phrase structure as their main descriptive device to seek extensions of the basic representation. The main proposals (Goodall 1987; Muadz 1991; Moltmann 1992) suggested different ways of adding an additional dimension to phrase markers such that parallel phrases could be represented side by side. Essentially, this is not very different from dependency based theories which add a coordination relation to the dependency relation.

When allowing coordination to be represented by a different relation than subordination, the interaction between the two relations must be specified. As an example, the interaction between the two relations in Hudson's dependency based Word Grammar is given by the *dependency-in-coordination principle*:

> Any dependency between a word inside one conjunct of a coordination and a word outside the coordination must also be shared by one word in every other conjunct of the same coordination.

This means that units which are coordinated need to share their dependencies with the units outside the coordination. For example, in the sentence *John bought and Bill read the book*, both verbs must share a dependency link with the object *the book*:[3]

$$\text{and} \longleftarrow \begin{array}{c}\text{John} \longleftarrow \text{bought}\\ \text{Bill} \longleftarrow \text{read}\end{array} \longrightarrow \text{the book}$$

[3]The diagram shows that we must also find a place for the coordinator *and* in the structure. Hudson makes it part of the second coordinate but I decided to keep the symmetry and write it as a modifier of the coordination as a whole.

One can imagine a similar extension of the common cover link representation in which a coordination relation is added and coordinates are allowed to share links. Because coordinated units are parallel and share their common cover link structure, the common cover links remain the same as in structures which involve no coordination. This means that the incremental properties of the common cover links can be preserved and that the basic decision process as to where to add a common cover link (the parsing function) can remain similar to that used when no coordination is involved. Since most relations in a sentence are subordination and not coordination, also the learning of the subordination relation (as coded in the parsing function) can remain unchanged. The incremental parser will have to be extended to allow parallel units and a decision function will have to be added to decide when units should be put in parallel to each other.

4.5.3 Relative Clauses

Relative clauses are another linguistic structure whose analysis has been much debated. A relative clause modifies a noun phrase by specifying the role the noun phrase plays in the relative clause. For example, in the sentence *the book he read was lost*, the noun phrase *the book* is modified by the relative clause *he read* which specifies that the book in question is the one which is the object of the verb read in the relative clause. In a bracket based representation of syntactic structure, relative clauses can be bracketed together with the noun phrase they modify: [[*the book*] [[*he*] *read*]]. Most linguists find this insufficient because it does not represent the fact that *the book* is modified by its role as the *object* of the relative clause. When working with a phrase structure representation the only solution is to add a trace marker, which is essentially a dependency link from the verb *read* to the noun phrase *the book*.

Since some links seem to be necessary to properly describe relative clauses and because common cover links are already similar in many ways to dependency links, I will consider from now on only the dependency structures which may be assigned to noun phrases modified by relative clauses. I have already discussed this briefly in section 4.1.1, where the following two dependency structures are given as possible representations of relative clause constructions:

$$\begin{tikzcd}
\text{the} & \text{book} \arrow[l] \arrow[rr, bend left] & \text{he} & \text{read} \arrow[l] & \text{was} \arrow[r] \arrow[lll, bend right] & \text{lost}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{the} & \text{book} \arrow[l] & \text{he} & \text{read} \arrow[l] \arrow[ll, bend right] & \text{was} \arrow[r] \arrow[lll, bend right] & \text{lost}
\end{tikzcd}$$

The second structure seems more satisfactory because it clearly indicates that the noun phrase *the book* is modified by its role as the object of the relative clause *he read.* It is also attractive because it assigns each word in the phrase *the book he read* the same link structure as in the sentence *he read a book*:

$$\begin{tikzcd}
\text{he} & \text{read} \arrow[l] \arrow[rr, bend left] & \text{the} & \text{book} \arrow[l]
\end{tikzcd}$$

The only difference is in the order of the words, which means that the basic link structure which was learned for the simple sentence can also be used to parse the relative clause construction. This simplifies learning but requires some flexibility to be added to the common cover link representation. Specifically, a shortest common cover link set cannot have two links entering the same word from the same side (Lemma 5.1.25). This means that the definition of the shortest common cover link set must be modified in some way to allow a satisfactory parse of relative clause constructions.

4.5.4 Word Order

To properly handle relative clauses in the way sketched in the previous section would require the parser to be able to recognize that the arguments of a head do not always appear in the same order. This flexibility of word order is not required only for relative clauses but can appear in almost any construction, depending on the language and how free its word order is. While this is an important issue which has to be dealt with, allowing for free word order has nothing to do with the link representation itself but depends only on the parsing function being used. The parsing functions defined in chapter 6 allow for flexible word order only to the extent that they allow for ambiguous constructions. This is certainly not a satisfactory solution for languages with significant word order freedom, but proper treatment of free word order is beyond the scope of the current work and is left for the future.

4.6 Conclusion

Common cover links clearly have much in common with dependencies and can be seen as representing a somewhat weaker relation. While the relation remains strong enough to allow effective parsing, it is weak enough to allow exocentric constructions and, even more importantly, incremental parsing. The main challenge for an incremental parser is to avoid assigning an incorrect structure to the prefixes of sentences which are initially ambiguous. While even humans fail on some ambiguities (so-called *garden path sentences*) other ambiguities can be handled without difficulty. Because the relation represented by common cover links is weaker than dependency, the common cover link parser can successfully handle some such initial ambiguities by avoiding being committed to structures which must be modified later in the parse. This allows for successful incremental parsing in cases which are difficult for parsers working directly with dependencies.

Chapter 5

Mathematical Analysis and Proofs

This chapter contains the full formal mathematical analysis of the structures and algorithms defined and used in previous chapters. In addition to proving all claims which appear in those chapters it also describes some additional properties of common cover link sets which were not essential there. This chapter is technical in nature and does not contain any linguistic motivation or examples. For these, the reader is referred to the preceding chapters. This chapter has been written to be self-contained and therefore repeats all the definitions and claims which appear in other chapters. However, it contains very little beyond the technical details and it is assumed the reader is familiar with the previous chapters.

The order of presentation in this chapter is somewhat different from that of previous chapters. Specifically, the reconstruction of a bracketing from a shortest common cover link set is proved after the characterization of the shortest common cover links is proved. After repeating the definition given in chapter 2, section 5.1 proves some simple basic properties of the common cover link sets, including linear transitivity and some simple reconstruction properties. Section 5.2 then provides a general algorithm which can reconstruct any bracketing from any of its shortest common cover link sets. This is the last section which does not impose any restriction on the common cover link sets. The following sections deal only with common cover link sets where all links are of depth 0 or 1. Section 5.3 proves the characterization of shortest common cover link sets with links of depth 0 and 1. The next section, section 5.4, returns to the bracket reconstruction problem. It proves the correctness of the incremental reconstruction algorithm for links of depth 0 and 1. The last section, section 5.5, focuses on the properties relevant to parsing. It investigates the adjacency property used in parsing, defines the incremental parsing algorithm and proves the correctness of this algorithm.

5.1 Basic Definitions and Properties

For ease of reference, I begin by repeating the definitions of section 2.2. Let W be a finite set of *word types*. An *utterance* is a sequence of words $\langle x_1, \ldots, x_n \rangle$ in W. The order of the words in the sequence is the order in which the words are uttered and will be said to be ordered 'from left to right'. Seen as a function from $\{1, \ldots, n\}$ to W, the sequence $\langle x_1, \ldots, x_n \rangle$ is equal to the set $\{\langle i, x_i \rangle\}_{i=1}^n$. Each element $\langle i, x_i \rangle$ in this set is a *word token*. When no confusion can arise, I will refer to both word types and word tokens as words. Any sub-sequence $\langle x_i, \ldots, x_j \rangle$ of consecutive words $(1 \leq i \leq j \leq n)$ is a *bracket* over the utterance. I will use lower-case letters $(x,\ y,\ \ldots)$ to denote word tokens and upper-case letters $(X,\ Y,\ \ldots)$ to denote brackets. I will treat brackets as sets of word tokens. The bracket Y *covers* the bracket X if $X \subset Y$ (strict inclusion) as sets of word tokens. Similarly, the bracket X *covers* the word token x if $x \in X$. I will write $[x, y]$ for the bracket covering all word tokens between x and y including x and y. I will also write $[x, y)$ (or $(y, x]$) for $[x, y] \setminus \{y\}$ and (x, y) for $[x, y] \setminus \{x, y\}$. The order of x and y is not important, so $[x, y) = (y, x]$. Two brackets X and Y are *non-crossing brackets* if either $X \cap Y = \emptyset$, $X \subseteq Y$ or $Y \subseteq X$.

5.1.1 (2.2.1). Definition. [bracketing] A bracketing of an utterance is a set of non-crossing brackets over that utterance such that every word in the utterance is covered by at least one bracket.

5.1.2 (2.2.2). Definition. [depth in a bracketing] Let $\mathcal{C}$ be a bracketing over an utterance U. The word x is of *depth* d under B in $\mathcal{C}$ if $x \in B \in \mathcal{C}$ and d is the maximal number of brackets $X_1, \ldots, X_d \in \mathcal{C}$ such that $x \in X_1 \subset \ldots \subset X_d \subset B$. In particular, if $x \in B \in \mathcal{C}$ and there is no bracket $X \in \mathcal{C}$ such that $x \in X \subset B$ then the depth of x under B in $\mathcal{C}$ is zero.

Notation I write $d_B^{\mathcal{C}}(x)$ for the depth of x under B in $\mathcal{C}$. I will also write $B_d^{\mathcal{C}}(x)$ for the unique bracket $B \in \mathcal{C}$ such that x is of depth d under B in $\mathcal{C}$, if such a bracket exists. When the bracketing is fixed by the context, I will simply write $d_B(x)$ for $d_B^{\mathcal{C}}(x)$ and $B_d(x)$ instead of $B_d^{\mathcal{C}}(x)$.

5.1.3 (2.2.3). Definition. [word of minimal depth] Let $\mathcal{C}$ be a bracketing. A word x is of minimal depth under $B \in \mathcal{C}$ if $x \in B$ and for every $y \in B$, $d_B^{\mathcal{C}}(x) \leq d_B^{\mathcal{C}}(y)$.

5.1.4 (2.2.4). Definition. [bracket height] Let $\mathcal{C}$ be a bracketing and $B \in \mathcal{C}$. The *height* of B in $\mathcal{C}$ is $\min_{x \in B} d_B^{\mathcal{C}}(x)$.

5.1.5 (2.3.1). DEFINITION. [common cover link] A *common cover link* of depth d over an utterance U is a triple $(x, y, d) \in U^2 \times (\mathbb{N} \cup \{0\})$ where $x \neq y$. I write $x \xrightarrow{d} y$ for the link (x, y, d), and the link is said to be from x to y. The word x is the *base* of the link and y is its *head*.

5.1.6 (2.3.2). DEFINITION. [common cover links of a bracketing ($R_{\mathcal{C}}$)] Let $\mathcal{C}$ be a bracketing over an utterance U. The set $R_{\mathcal{C}}$ of common cover links for $\mathcal{C}$ is the set of common cover links over U such that $x \xrightarrow{d} y \in R_{\mathcal{C}}$ iff x is of minimal depth d under the smallest bracket $B \in \mathcal{C}$ such that $x, y \in B$.

5.1.1 Basic Properties of the Common Cover Link Set

This section begins by giving some very elementary (and simple) properties of bracketings and the minimal depth relation. In the second part of this section, some basic properties of the common cover link set are given, including several transitivity properties. The most important case of transitivity is that of linear transitivity, which is used in the definitions and algorithms of the following sections.

Elementary Properties of Bracketing

5.1.7. LEMMA. *Let $\mathcal{C}$ be a bracketing, $X, Z \in \mathcal{C}$ and $X \subset Z$. If $Y_1, \ldots, Y_d \in \mathcal{C}$ is a maximal subset of $\mathcal{C}$ such that $X \subset Y_1 \subset \ldots Y_d \subset Z$ then $\{Y_1, \ldots, Y_d\} = \{Y \in \mathcal{C} \ : \ X \subset Y \subset Z\}$.*

PROOF. By definition, $\{Y_1, \ldots, Y_d\} \subseteq \{Y \in \mathcal{C} \ : \ X \subset Y \subset Z\}$. Let $X \subset Y \subset Z$. For each i, $X \subset Y_i$ and therefore $Y \cap Y_i \neq \emptyset$. Because brackets in $\mathcal{C}$ do not cross, either $Y \subseteq Y_i$ or $Y_i \subset Y$. Because of the maximality of the chain $Y_1, \ldots, Y_d$ there is an i such that $Y = Y_i$ (otherwise Y can be added to create a longer chain). $\square$

5.1.8. LEMMA. *Let $\mathcal{C}$ be a bracketing and let $x \in X \in \mathcal{C}$. If $X_1, \ldots, X_d \in \mathcal{C}$ is a maximal subset of $\mathcal{C}$ such that $x \in X_1 \subset \ldots X_d \subset X$ then $\{X_1, \ldots, X_d\} = \{Y \in \mathcal{C} \ : \ x \in Y \subset X\}$.*

PROOF. The proof is identical to the proof of Lemma 5.1.7. □

5.1.9. LEMMA. *Let $\mathcal{C}$ be a bracketing. If $B, C \in \mathcal{C}$, $x, y \in C \subseteq B$ then $d_B(x) - d_C(x) = d_B(y) - d_C(y)$.*

PROOF. If $C = B$, the claim is trivial, so assume $C \subset B$. Let $X_1, \ldots, X_{d_x} \in \mathcal{C}$ and $Y_1, \ldots, Y_{d_y} \in \mathcal{C}$ be maximal chains such that $x \in X_1 \subset \ldots \subset X_{d_x} \subset B$ and $y \in Y_1 \subset \ldots \subset Y_{d_y} \subset B$. By Lemma 5.1.8 and the definition of depth, $C = X_{d_C(x)+1} = Y_{d_C(y)+1}$. By Lemma 5.1.7, $\{X_{d_C(x)+2}, \ldots X_{d_x}\} = \{Y_{d_C(y)+2}, \ldots Y_{d_y}\}$ and therefore $d_B(x) - d_C(x) = d_B(y) - d_C(y)$. □

5.1.10. LEMMA. *Let $\mathcal{C}$ be a bracketing. If $B, C \in \mathcal{C}$, $x \in C \subset B$ and x is of minimal depth under B then x is of minimal depth under C.*

PROOF. By definition, $C = B_{d_1}(x)$ and $B = B_{d_2}(x)$ for some $d_1 < d_2$. Assume by contradiction that there exists y such that $C = B_{d_3}(y)$ with $d_3 < d_1$. It follows from Lemma 5.1.9 that $B = B_{d_3+(d_2-d_1)}(y)$ and since $d_3 + (d_2 - d_1) < d_2$ this contradicts the assumption that x is of minimal depth under B. □

5.1.11. LEMMA. *Let $\mathcal{C}$ be a bracketing. If $x \xrightarrow{d} y \in R_{\mathcal{C}}$ then $B_d(x)$ exists and is the smallest bracket in $\mathcal{C}$ which covers both x and y.*

PROOF. By definition of $R_{\mathcal{C}}$, $x \xrightarrow{d} y \in R_{\mathcal{C}}$ implies that there is a minimal bracket $B \in R_{\mathcal{C}}$ which covers x and y and x is of depth d under B. By definition, $B = B_d(x)$. □

Basic Properties of Common Cover Link Sets

5.1.12. LEMMA (CONNECTEDNESS). *Let $\mathcal{C}$ be a bracketing. If $x \xrightarrow{d} y \in R_{\mathcal{C}}$ and $z \in (x, y)$ then there is a link $x \xrightarrow{d'} z \in R_{\mathcal{C}}$ for some $d' \leq d$.*

PROOF. By the assumptions of the lemma, $y \in B_d(x)$ and x is of minimal depth under this bracket. Since $z \in (x, y)$, there is a bracket covering x and z and the minimal such bracket is $B_{d'}(x)$ for some $d' \leq d$. By Lemma 5.1.10, x is of minimal depth under $B_{d'}(x)$ and therefore $x \xrightarrow{d'} z \in R_{\mathcal{C}}$. □

5.1.13. LEMMA. *Let $\mathcal{C}$ be a bracketing. If $B_d(x)$ exists and x is of minimal depth under $B_d(x)$ then $B_d(x) = \{x\} \cup \bigcup_{0 \leq d' \leq d} \left\{ y \ : \ x \xrightarrow{d'} y \in R_{\mathcal{C}} \right\}$.*

PROOF. Assume $B_d(x)$ exists and x is of minimal depth under $B_d(x)$. By Lemma 5.1.11, if $x \xrightarrow{d'} y \in R_{\mathcal{C}}$ for some $d' \leq d$ then $y \in B_{d'}(x) \subseteq B_d(x)$. At the same time, if $y \in B_d(x) \setminus \{x\}$ then there is $d' \leq d$ such that $B_{d'}(x)$ is the minimal bracket covering x and y. By Lemma 5.1.10, x is of minimal depth under $B_{d'}(x)$, so $x \xrightarrow{d'} y \in R_{\mathcal{C}}$. □

5.1.14. LEMMA (BACK AND FORTH EQUALITY). *Let $\mathcal{C}$ be a bracketing and assume $x \xrightarrow{d_1} y \in R_{\mathcal{C}}$ and $y \xrightarrow{d_2} x \in R_{\mathcal{C}}$ then $d_1 = d_2$.*

PROOF. By Lemma 5.1.11, since $x \xrightarrow{d_1} y \in R_{\mathcal{C}}$, x is of minimal depth d_1 under the smallest bracket B in $\mathcal{C}$ which covers both x and y. Because $y \xrightarrow{d_2} x \in R_{\mathcal{C}}$, y is of minimal depth d_2 under B. Since the minimal depth under a bracket is unique it follows that $d_1 = d_2$. □

5.1.15. LEMMA (TRANSITIVITY OF $R_{\mathcal{C}}$). *Let $\mathcal{C}$ be a bracketing. If $x \xrightarrow{d_1} y \in R_{\mathcal{C}}$ and $y \xrightarrow{d_2} z \in R_{\mathcal{C}}$ and $x \neq z$ then $x \xrightarrow{d} z \in R_{\mathcal{C}}$ for some $d \leq \max(d_1, d_2)$.*

PROOF. Assume the assumptions of the lemma hold. By definition, $x, y \in B_{d_1}(x)$ and $y, z \in B_{d_2}(y)$. Since $y \in B_{d_1}(x) \cap B_{d_2}(y)$ and brackets do not cross, either $B_{d_2}(y) \subseteq B_{d_1}(x)$ or $B_{d_1}(x) \subset B_{d_2}(y)$.

- *Case 1: $B_{d_2}(y) \subseteq B_{d_1}(x)$.* Since $z \in B_{d_2}(y)$, there is a bracket covering z and x and the minimal such bracket is $B = B_d(x)$ for some $d \leq d_1$. By Lemma 5.1.10, x is of minimal depth under $B_d(x)$ and therefore $x \xrightarrow{d} z \in R_{\mathcal{C}}$.
- *Case 2: $B_{d_1}(x) \subset B_{d_2}(y)$.* Since $y \in B_{d_1}(x)$ and is of minimal depth under $B_{d_2}(y)$, it follows from Lemma 5.1.10 that y is of minimal depth under $B_{d_1}(x)$. The depth of x and y under $B_{d_1}(x)$ must be the same and, therefore, by Lemma 5.1.9, the depth of x and y under $B_{d_2}(y)$ must also be the same. Therefore, $B_{d_2}(y) = B_{d_2}(x)$ and x is of minimal depth under this bracket. Since $y \xrightarrow{d_2} z \in R_{\mathcal{C}}$, $B_{d_2}(y)$ is the smallest bracket in $\mathcal{C}$ covering y and z. Since brackets do not cross, $B_{d_2}(x) = B_{d_2}(y)$ is also the smallest bracket covering x and z, so $x \xrightarrow{d_2} z \in R_{\mathcal{C}}$.

□

5.1.16 (2.5.1). LEMMA (LINEAR TRANSITIVITY OF $R_{\mathcal{C}}$). *Let $R_{\mathcal{C}}$ be the common cover link set of a bracketing $\mathcal{C}$. If $y \in (x, z)$, $x \xrightarrow{d_1} y \in R_{\mathcal{C}}$ and $y \xrightarrow{d_2} z \in R_{\mathcal{C}}$ then $x \xrightarrow{d} z \in R_{\mathcal{C}}$ with the following depth (See figure 2.1):*

1. *If $y \to x \in R_{\mathcal{C}}$ then $d = \max(d_1, d_2)$.*
2. *Otherwise $d = d_1$.*

PROOF. Assume that the assumptions of the lemma hold. This is a special case of transitivity (Lemma 5.1.15) and therefore $x \xrightarrow{d} z \in R_{\mathcal{C}}$ for some $d \leq \max(d_1, d_2)$. By connectedness (Lemma 5.1.12), $d \geq d_1$. We are either in case 1 or case 2 of the previous proof:

- *Case 1:* $B_{d_2}(y) \subseteq B_{d_1}(x)$. In this case $d \leq d_1$ which implies $d = d_1$. If $y \to x \in R_{\mathcal{C}}$ then $B_{d_1}(y) = B_{d_1}(x)$ which implies $d_2 \leq d_1$. This completes the proof in this case.
- *Case 2:* $B_{d_1}(x) \subset B_{d_2}(y)$. We saw that in this case y is of minimal depth under $B_{d_1}(x)$, so $y \xrightarrow{d_1} x \in R_{\mathcal{C}}$. We also saw that $d = d_2$. It remains to show that $d_1 \leq d_2$, which follows from $B_{d_2}(x) = B_{d_2}(y)$.

$\square$

5.1.17. LEMMA (CROSSING LINKS). *Let $\mathcal{C}$ be a bracketing. If, for $y \in (x, z)$, $y \xrightarrow{d_1} x \in R_{\mathcal{C}}$ and $x \xrightarrow{d_2} z \in R_{\mathcal{C}}$ then $d_1 \leq d_2$ and there is a link $x \xrightarrow{d_1} y \in R_{\mathcal{C}}$.*

PROOF. Assume the assumptions of the lemma hold. Because $y \xrightarrow{d_1} x \in R_{\mathcal{C}}$, $B_{d_1}(y)$ is the smallest bracket in $\mathcal{C}$ which covers both x and y. Similarly, $B_{d_2}(x)$ is the smallest bracket in $\mathcal{C}$ which covers x and z. Because $y \in (x, z)$, $y \in B_{d_2}(x)$ and therefore $B_{d_1}(y) \subseteq B_{d_2}(x)$. Since $x \xrightarrow{d_2} z \in R_{\mathcal{C}}$, x is of minimal depth under $B_{d_2}(x)$ and (by Lemma 5.1.10) must also be of minimal depth under $B_{d_1}(y)$. It follows that $d_1 \leq d_2$ and $x \xrightarrow{d_1} y \in R_{\mathcal{C}}$. $\square$

5.1.2 Representative and Shortest Common Cover Link Sets

This section examines some of the relations between the common cover link set, the representative subsets and the shortest common cover link sets.

5.1.18 (2.4.1). Definition. [representative subset] Let $\mathcal{C}$ be a bracketing. A subset $R \subseteq R_{\mathcal{C}}$ is a representative subset of $R_{\mathcal{C}}$ iff:

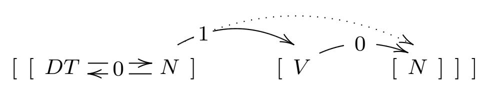
1. For every $x \xrightarrow{0} y \in R_{\mathcal{C}}$ also $x \xrightarrow{0} y \in R$.
2. For every $X \in \mathcal{C}$, if d is the height of X then there is at least one word x of minimal depth d under X such that for every $x \xrightarrow{d} y \in R_{\mathcal{C}}$ also $x \xrightarrow{d} y \in R$. This word x is a *representative* for the bracket X.
3. For every word x and every depth d if $x \xrightarrow{d} y \in R$ and $x \xrightarrow{d} z \in R_{\mathcal{C}}$ then $x \xrightarrow{d} z \in R$.

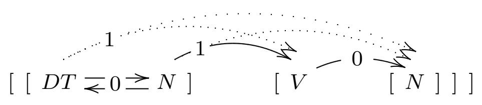
5.1.19 (2.5.2). Definition. [shortest common cover link set] Let $\mathcal{C}$ be a bracketing. A set $S \subseteq R_{\mathcal{C}}$ is a shortest common cover link set if there is a representative subset R of $R_{\mathcal{C}}$ such that $x \xrightarrow{d} z \in S$ iff $x \xrightarrow{d} z \in R$ and there is no word $y \in (x, z)$ such that $x \xrightarrow{d_1} y \in R$ and $y \xrightarrow{d_2} z \in R$.

It is clear from the definition that a common cover link set $R_{\mathcal{C}}$ may have more than one representative subset (an example of this was given in section 2.4). It is also clear from the definition that given a representative subset R of $R_{\mathcal{C}}$, there is a unique shortest common cover link set derived from it.

Notation I write $S(R)$ for the shortest common cover link set induced by the representative subset R.

While a representative subset determined a shortest common cover link set uniquely, The opposite is not true. Different representative subsets (of the same bracketing) may result in the same shortest common cover link set. An example of this can be based on the examples given in section 2.4. Consider two representative subsets for the bracketing [[DT N] [V [N]]]. The first, R_1, has the first N as the only representative of the top bracket while the second set, R_2, has both N and DT as representatives of the top bracket. The following diagram shows that both these representative subsets reduce to the same shortest common cover link set (solid links are in $S(R_i)$ and dotted links in $R_i \setminus S(R_i)$):


R_1


R_2

In this example one representative subset (R_1) is contained in the other (R_2). This suggests that if only minimal representative subsets are considered then a shortest common cover link set does determine a representative subset uniquely. This turns out to be indeed the case, as shown by Corollary 5.1.22 below. This corollary follows from a lemma which, given a link $x \xrightarrow{d} y \in R_{\mathcal{C}}$ and a shortest common cover link set S of $R_{\mathcal{C}}$ determines whether $x \xrightarrow{d} y$ is in the minimal representative subset containing S. The lemma states that a link $x \xrightarrow{d} y \in R_{\mathcal{C}}$ is in the minimal representative subset containing a shortest common cover link set S if and only if S contains a link with the same base and depth as $x \xrightarrow{d} y$. This lemma plays an important role in the reconstruction of the bracketing $\mathcal{C}$ from S in section 5.2.

5.1.20 (2.6.1). Definition. [minimal representative subset] Let $\mathcal{C}$ be a bracketing and let S be a shortest common cover link set for this bracketing. A representative subset $R \subseteq R_{\mathcal{C}}$ is a *minimal representative subset* containing S if $S \subseteq R$ and there is no representative subset R' such that $S \subseteq R' \subset R$.

5.1.21 (2.6.2). Lemma. *Let $\mathcal{C}$ be a bracketing, let S be a shortest common cover link set of $R_{\mathcal{C}}$ and let $R(S)$ be a minimal representative subset containing S. If $x \xrightarrow{d} y \in R_{\mathcal{C}}$ then $x \xrightarrow{d} y \in R(S)$ iff there is some z such that $x \xrightarrow{d} z \in S$.*

Proof. Assume that the assumptions of the lemma hold and $x \xrightarrow{d} y \in R_{\mathcal{C}}$. First assume that $x \xrightarrow{d} y \notin R(S)$. By the definition of a representative subset (Definition 5.1.18) it follows that there is no link $x \xrightarrow{d} z \in R(S)$ for any word z and in particular there is no such link in S.

To prove the other direction, assume that $x \xrightarrow{d} y \in R(S)$. Let z be the closest word to x in $[x, y]$ such that $x \xrightarrow{d} z \in R_{\mathcal{C}}$. I will show that $x \xrightarrow{d} z \in S$ (which proves the lemma). Assume, by contradiction, that $x \xrightarrow{d} z \notin S$. I will show that this implies that there is a representative subset R such that $S \subseteq R \subset R(S)$, contradicting the minimality assumption on $R(S)$.

Because $x \xrightarrow{d} y \in R(S)$, it follows by the definition of a representative subset that also $x \xrightarrow{d} z \in R(S)$. Since $x \xrightarrow{d} z \notin S$ there must be $w \in (x, z)$ such that $x \xrightarrow{d_1} w \in R(S)$ and $w \xrightarrow{d_2} z \in R(S)$. Since z is the closest word to x in $[x, y]$ with a link $x \xrightarrow{d} z \in R(S)$ it follows that $d_1 \neq d$. By linear transitivity (Lemma 5.1.16), it follows that $d = d_2 > d_1$ and (together with back and forth equality, Lemma 5.1.14) that $w \xrightarrow{d_1} x \in R_{\mathcal{C}}$. This means that both x and w are of minimal depth under $B_{d_1}(x) = B_{d_1}(w)$ which is the smallest bracket covering them both. Since $d > d_1$, it follows from $x \xrightarrow{d} y \in R_{\mathcal{C}}$ and Lemma 5.1.9 that both x and w are of minimal depth under $B_d(x) = B_d(w)$. Since $x \xrightarrow{d} z \in R(S)$ and $w \xrightarrow{d} z \in R(S)$, both x and w are representatives of $B_d(x)$ in $R(S)$. Therefore, the set R which is constructed from $R(S)$ by discarding all links $x \xrightarrow{d} v \in R(S)$ is a representative subset of $R_{\mathcal{C}}$. Since there is no link $x \xrightarrow{d} v \in S$, $S \subseteq R \subset R(S)$ in contradiction to the minimality assumption of $R(S)$. So $x \xrightarrow{d} z \in S$ after all. □

5.1.22. Corollary (uniqueness of $R(S)$). *Let $\mathcal{C}$ be a bracketing and let S be a shortest common cover link set of $R_{\mathcal{C}}$. There is a unique minimal representative subset $R(S)$ of $R_{\mathcal{C}}$ containing S.*

Proof. This follows immediately from Lemma 5.1.21. □

The next lemma shows that the different depths of links used in $R_{\mathcal{C}}$ must also all be used in every shortest common cover link set of $R_{\mathcal{C}}$. In particular, this means that when the depth of links is restricted to 0 and 1, it does not matter whether this restriction is defined on the common cover link set $R_{\mathcal{C}}$ or on the shortest common cover link set.

5.1.23. Lemma. *Let $\mathcal{C}$ be a bracketing and let S be a shortest common cover link set of $R_{\mathcal{C}}$. For every $0 \leq d$, there is a link of depth d in $R_{\mathcal{C}}$ iff there is a link of depth d in S.*

Proof. Since $S \subseteq R_{\mathcal{C}}$, it is enough to show that if there is a link of depth d in $R_{\mathcal{C}}$ then there is a link of depth d in S. Assume there is a link of depth d in $R_{\mathcal{C}}$ and let R be a representative subset of $R_{\mathcal{C}}$ such that $S = S(R)$. It is obvious from the definition of representative subsets that there is a link of depth d in R. If $x \xrightarrow{d} z \in R$ but $x \xrightarrow{d} z \notin S$ then, by definition, there is $y \in (x, z)$ such that $x \xrightarrow{d_1} y \in R$ and $y \xrightarrow{d_2} z \in R$. By linear transitivity, either $d_1 = d$ or $d_2 = d$. This shows that $x \xrightarrow{d} z \in R$ and $x \xrightarrow{d} z \notin S$ imply that there is a shorter link of depth d in R. This argument can be repeated for this shorter link. Since the length of links is bounded from below, this shows that there is a link of depth d in S. □

5.1.3 Directional Uniqueness

In the process of parsing it is useful to assume the *directional uniqueness* property: for every word z there is at most one link entering it from each side. When the depth of links is restricted to 0 and 1, this property follows directly from the characterization of the shortest common cover link set and there is no need to make explicit use of directional uniqueness. When there is no restriction on the depth of links, this property is no longer automatic and it may be useful to enforce it in some way. As long as linguistic considerations restrict links to depth 0 or 1, this remains of theoretical interest and is used in several of the proofs in this chapter.

5.1.24. Definition. [directional uniqueness] Let L be a set of common cover links over an utterance U. The set L has the *directional uniqueness* property if for any $x, y, z \in U$ such that $z \notin [x, y]$, $x \xrightarrow{d_1} z \in L$ and $y \xrightarrow{d_2} z \in L$ implies $x = y$.

The directional uniqueness property is useful because it restricts the space of link combinations the parser needs to consider: when the parser has two candidate links entering the same word from the same side, it must select one or the other. The following lemma shows that it is reasonable for the parser to assume the directional uniqueness property of the links because this property holds for at least some shortest common cover link sets of any bracketing. Moreover, when the depth of links is limited to 0 and 1 (see section 4.4), any shortest common cover link set has the directional uniqueness property.

5.1.25. Lemma (directional uniqueness). *Let $\mathcal{C}$ be a bracketing and let S be a shortest common cover link set for $R_{\mathcal{C}}$. There is a shortest common cover link set S' for $R_{\mathcal{C}}$ such that $S' \subseteq S$ and S' has directional uniqueness. Moreover, if the depth of links in S is 0 or 1, S has directional uniqueness.*

Proof. Assume the assumptions of the lemma hold. Assume $z \notin [x, y]$, $x \xrightarrow{d} z \in S$ and $y \xrightarrow{d_2} z \in S$. If $x = y$ then directional uniqueness is not violated. Therefore, assume without loss of generality that $y \in (x, z)$. Since $x \xrightarrow{d} z \in L$ it follows from connectedness (Lemma 5.1.12) that $x \xrightarrow{d_1} y \in R_{\mathcal{C}}$ for some $d_1 \leq d$.

Let $R(S)$ be the smallest representative subset of $R_{\mathcal{C}}$ such that $S \subseteq R(S)$. If $x \xrightarrow{d_1} y \in R(S)$ then $x \xrightarrow{d} z \in S$ and $y \xrightarrow{d_2} z \in S$ contradict the definition of the shortest common cover link set. Therefore, $x \xrightarrow{d_1} y \in R_{\mathcal{C}} \setminus R(S)$. If $d_1 = d$ then because $x \xrightarrow{d} z \in R(S)$ also $x \xrightarrow{d} y \in R(S)$, which is a contradiction. From linear transitivity (Lemma 5.1.16) the only remaining possibility is that $d_1 < d$ and $d = d_2$. If $d, d_2 \leq 1$ then $d_1 = 0$ and the link $x \xrightarrow{d_1} y \in R(S)$, which is a contradiction. This shows that if the depth of links in S is 0 or 1, then directional uniqueness holds. The only possibility left is that $1 < d = d_2$ and $x \xrightarrow{d_1} y \in R_{\mathcal{C}}$ which implies (by linear transitivity) that $y \xrightarrow{d_1} x \in R_{\mathcal{C}}$. Therefore, both x and y are representatives of $B_d(x) = B_{d_2}(y)$ in $R(S)$. There therefore exists a smaller representative subset $R' \subset R(S)$ in which only one of x and y is a representative of $B_d(x)$. At most one of the links $x \xrightarrow{d} z$ and $y \xrightarrow{d_2} z$ is in R' and therefore the smallest common cover link set $S' \subset S$ derived from R' does not contain this violation of directional uniqueness. This can be repeated until no violations of directional uniqueness remain, resulting in the set S' required by the lemma. □

5.1.4 Paths

Since linear transitivity can be apply repeatedly, a single link in the common cover link set $R_{\mathcal{C}}$ may be represented in a shortest common cover link set of $R_{\mathcal{C}}$ by a sequence of links. Such sequences of links are defined as *link paths*:

5.1.26 (2.6.3). Definition. [linear path] Let U be an utterance. A sequence $x_1, \ldots, x_m$ of words in U is a *linear path* from x_1 to x_m in U (written $x_1 - \ldots - x_m$) iff for each $1 < i < m$ $x_i \in (x_{i-1}, x_{i+1})$.

5.1.27 (2.6.4). Definition. [link path] Let L be a set of common cover links over an utterance U and let $x, y \in U$. An L-path from x to y is a set of links $\left\{x_i \xrightarrow{d_i} x_{i+1}\right\}_{i=1}^{m-1}$ in L such that $x_1 - \ldots - x_m$ is a linear path, $x_1 = x$ and $x_m = y$. In particular, for every $x \in U$, there is an empty L-path from x to x.

Notation I write $x \xrightarrow{L} y$ if there is an L-path from x to y and $x \xrightarrow{d,L} y$ if there is an L-path from x to y which begins with a link of depth d.

The next lemma gives two simple properties of link paths in a shortest common cover link set and their relation with links in the common cover link set and in the representative subset which induces the shortest common cover link set.

5.1.28. Lemma. *Let $\mathcal{C}$ be a bracketing and let R be a representative subset of $R_{\mathcal{C}}$. Let S be the shortest common cover link set induced by R.*

1. *If $x \to y \in R$ then $x \xrightarrow{S} y$.*

2. *If $x \xrightarrow{S,d} y$ then there is $d' \geq d$ such that $x \xrightarrow{d'} y \in R_{\mathcal{C}}$.*

Proof.

1. Assume the assumptions of the lemma hold and $x \xrightarrow{d} y \in R$. The proof is by induction on the distance between x and y. If the distance is 1 (x and y are consecutive words) then it follows directly from the definition of S that $x \xrightarrow{d} y \in S$. Assume now that the distance is greater than 1. If $x \xrightarrow{d} y \in S$ then the claim holds. Otherwise, by the definition of S, there is $w \in (x, y)$ such that $x \xrightarrow{d_1} w \in R$ and $w \xrightarrow{d_2} y \in R$. By the induction hypothesis, $x \xrightarrow{S} w$ and $w \xrightarrow{S} y$, which together imply $x \xrightarrow{S} y$, as required.

2. Assume that $\left\{x_i \xrightarrow{d_i} x_{i+1}\right\}_{i=1}^{m-1}$ is an S-path from x_1 to x_m. The proof of the first part of the lemma is by induction on the number of links in the S-path. If $m = 2$ then the claim is immediate from linear transitivity (Lemma 5.1.16). If $m > 2$ then, by the induction hypothesis, there is a link $x_2 \rightarrow x_m \in R_\mathcal{C}$. Again, using linear transitivity the claim follows.

$\square$

This lemma is not sufficient to reconstruct a representative subset from a shortest common cover link set S because an S-path from x to y does not specify the depth of the link from x to y in the representative subset. To overcome this problem (and others) a more sophisticated algorithm is need. This algorithm is given in section 5.2.

5.1.5 Simple Reconstruction Algorithm

The simple reconstruction algorithm can reconstruct a bracketing $\mathcal{C}$ from its common cover link set $R_\mathcal{C}$ and if all links in $R_\mathcal{C}$ are of depth 0 and 1, the algorithm can also reconstruct $\mathcal{C}$ from any representative subset of $R_\mathcal{C}$. While this algorithm is not of practical significance, it is used as a theoretical tool in some proofs.

5.1.29 (2.3.3). Algorithm. [simple bracket reconstruction from $R_\mathcal{C}$]

1. For every word x, construct the smallest bracket which covers x and all y such that $x \xrightarrow{0} y \in R_\mathcal{C}$ (this bracket is $B_0(x)$).

2. Having constructed $B_d(x)$ ($d \geq 0$) and if there are links $x \xrightarrow{d+1} y$ in $R_\mathcal{C}$, construct $B_{d+1}(x)$ by constructing the smallest bracket which covers $B_d(x)$ and all words y such that $x \xrightarrow{d+1} y \in R_\mathcal{C}$.

Notation I write $\mathcal{A}(L)$ for the result of applying the simple bracket reconstruction algorithm (Algorithm 5.1.29) to a set of common cover links L.

5.1.30. LEMMA (SIMPLE RECONSTRUCTION). *Let $\mathcal{C}$ be a bracketing.*

1. $\mathcal{A}(R_{\mathcal{C}}) = \mathcal{C}$.
2. *If all links in $R_{\mathcal{C}}$ are of depth 0 and 1 then for any representative subset R of $R_{\mathcal{C}}$, $\mathcal{A}(R) = \mathcal{C}$.*

PROOF. Let $A_0(x)$ be the bracket constructed for each x in the first step of the algorithm and let $A_d(x)$ be the bracket constructed by the algorithm from the links of depth d based at x.

1. Assume that the input to the algorithm is $R_{\mathcal{C}}$. I show that for every x and d, $A_d(x)$ is constructed iff $B^{\mathcal{C}}_d(x)$ exists and x is of minimal depth under $B^{\mathcal{C}}_d(x)$ and that in this case $A_d(x) = B^{\mathcal{C}}_d(x)$. This proves the claim because every bracket has a word which is of minimal depth under it. The proof is by induction on d.

   Let $d = 0$. The algorithm always constructs $A_0(x)$. Because, by definition, every word is covered by some bracket in $\mathcal{C}$) $B^{\mathcal{C}}_0(x)$ always exists. Because $d = 0$, x must be of minimal depth under $B^{\mathcal{C}}_0(x)$. That $B^{\mathcal{C}}_0(x) = A_0(x)$ follows directly from Lemma 5.1.13.

   Now let $d > 0$. The bracket $A_d(x)$ is constructed iff there is a link $x \xrightarrow{d} y \in R_{\mathcal{C}}$. By the definition of $R_{\mathcal{C}}$ and Lemma 5.1.11, there exists $x \xrightarrow{d} y \in R_{\mathcal{C}}$ iff $B^{\mathcal{C}}_d(x)$ exists, x is of minimal depth under $B^{\mathcal{C}}_d(x)$ and there is a y such that $B^{\mathcal{C}}_d(x)$ is the minimal bracket covering x and y. This holds iff $B^{\mathcal{C}}_d(x)$ exists and x is of minimal depth under $B^{\mathcal{C}}_d(x)$.

   It remains to show that in this case $A_d(x) = B^{\mathcal{C}}_d(x)$. By the induction hypothesis, $A_d(x)$ is the minimal bracket covering $B^{\mathcal{C}}_{d-1}(x)$ and $\{y : x \xrightarrow{d} y \in R_{\mathcal{C}}\}$. By Lemma 5.1.13, $A_d(x) = B^{\mathcal{C}}_d(x)$.

2. Assume that $R_{\mathcal{C}}$ only contains links of depth 0 and 1 and let R be a representative subset of $R_{\mathcal{C}}$. Since all links in $R_{\mathcal{C}}$ are of depth 0 and 1, every bracket in $\mathcal{C}$ is either of height 0 or 1. Also, the reconstruction algorithm clearly only constructs brackets $A_0(x)$ and $A_1(x)$.

   By definition, all links of depth 0 in $R_{\mathcal{C}}$ are also in R. Therefore, the case $d = 0$ of the previous part of the proof applies here and $A_0(x)$ is constructed iff $B^{\mathcal{C}}_0(x)$ exists and then $A_0(x) = B^{\mathcal{C}}_0(x)$. This covers the reconstruction of brackets of height 0.

By the definition of representative subsets, $B \in \mathcal{C}$ is of height 1 iff there is a representative x (for R) such that $B = B_1^{\mathcal{C}}(x)$. This holds iff there is some y such that $x \xrightarrow{1} yR_{\mathcal{C}}$ and, for every y, $x \xrightarrow{1} y \in R$ iff $x \xrightarrow{1} y \in R_{\mathcal{C}}$. This holds iff $A_1(x)$ is constructed. In this case the previous part of the proof applies and $A_1(x) = B_1^{\mathcal{C}}(x)$.

□

The second part of the simple reconstruction lemma cannot be extended to representative subsets without restriction of the depth of links because the induction argument assumes that $A_{d-1}(x)$ is constructed before $A_d(x)$. This can only be guaranteed for $d = 1$ because, for $d > 1$, it may be that x is a representative of $B_d^{\mathcal{C}}(x)$ but not of $B_{d-1}^{\mathcal{C}}(x)$. This problem is addressed by the general bracket reconstruction algorithm in section 5.2.

5.2 Bracket Reconstruction

This section shows that a bracketing $\mathcal{C}$ can always be reconstructed from a shortest common cover link set S of $R_{\mathcal{C}}$, regardless of the depth of the links in $R_{\mathcal{C}}$. Because linguistic structures do not seem to have links of depth greater than 1, the general case discussed in this section is of mathematical rather than linguistic interest. The reader who is interested only in the linguistic aspects of the method may safely skip this section.

When the depth of links is allowed to be greater than 1, the algorithm given in section 2.6.1 does not work anymore. In order to calculate $R(S)$ (the smallest representative subset containing S) the algorithm needs to apply linear transitivity to links in S. To deduce a link by linear transitivity, the algorithm needs to determine both the endpoints of the link and its depth. Given $y \in (x, z)$ and two links $x \xrightarrow{d_1} y \in R_{\mathcal{C}}$ and $y \xrightarrow{d_2} z \in R_{\mathcal{C}}$, linear transitivity (Lemma 5.1.16) implies that there is a link $x \xrightarrow{d_3} z \in R_{\mathcal{C}}$. If $d_1 \geq d_2$ then $d_3 = d_1$ and there is no problem. However, when $d_1 < d_2$, the algorithm must know whether there is a link $y \xrightarrow{d'} x \in R_{\mathcal{C}}$. When the depth of links is limited to 0 and 1, the depth of the link $y \xrightarrow{d'} x$ is always 0 (Lemma 5.1.14) and the link is in the representative subset $R(S)$. However, when the depth is not bounded, the link $y \xrightarrow{d'} x$ may be in $R_{\mathcal{C}}$ but not in $R(S)$. Since the algorithm given in section 2.6.1 reconstructs $R(S)$ (rather than $R_{\mathcal{C}}$), it cannot determine the depth d_3 in these situations.

Even if an algorithm has properly calculated the representative subset $R(S)$, it cannot apply the simple bracket reconstruction algorithm (Algorithm 5.1.29) to infer the bracket from $R(S)$. This is because the algorithm constructs a sequence of bracket $B_0(x), B_1(x), \ldots$ independently for each word x. To construct $B_{d+1}(x)$

the algorithm needs to know $B_d(x)$. When $1 \leq d$ it may be that x is not a representative word of $B_d(x)$ and therefore the bracket cannot be constructed from links based at x but needs to be constructed as $B_d(y)$ for some other word y. As a result, $B_{d+1}(x)$ can only be calculated after $B_d(y)$ has been created. The algorithm has to identify that $B_d(y)$ is indeed also $B_d(x)$.

Both these problems can be solved simultaneously. I will show below (Lemma 5.2.7) that to determine the depth of a link $x \xrightarrow{d_3} z$ inferred by linear transitivity from links $x \xrightarrow{d_1} y$ and $y \xrightarrow{d_2} z$ it is sufficient to know the bracketing reconstructed by Algorithm 5.2.1 from links in $R(S)$ which have their endpoints in $[x, z)$. At the same time, I will show Algorithm 5.2.1 can reconstruct $\mathcal{C}$ from any of its representative subsets. The combination of these two algorithms allows $\mathcal{C}$ to be reconstructed from S by calculating links in $R(S)$ incrementally for words further and further apart, each time calculating the bracketing implied by those links.

5.2.1 Reconstruction from Representative Subsets

Let $\mathcal{C}$ be a bracketing and let S be a shortest common cover link set of $R_{\mathcal{C}}$. Let $R(S)$ be the smallest representative subset of $R_{\mathcal{C}}$ such that $S \subseteq R(S)$ and let $R^{[x,z)}(S) = \{v \xrightarrow{d} w \in R(S) \ : \ v, w \in [x, z)\}$ be the restriction of $R(S)$ to $[x, z)$. The following algorithm can be used to calculate a bracketing of $[x, z)$ from $R^{[x,z)}(S)$.

5.2.1. Algorithm. [brackets from any links] Given a set L of common cover links over an utterance U, the algorithm creates a bracketing $\mathcal{B}$ over U. For every word $x \in U$ and $-1 \leq d$, the algorithm maintains sets of brackets $M_d(x) \subseteq \mathcal{B}$.

- Initialize $\mathcal{B} = \emptyset$ and for every $x \in U$, initialize $M_{-1}(x) = \{\langle x \rangle\}$ and $M_d(x) = \emptyset$ for $0 \leq d$.
- For each $d = 0, \ldots, \max\{d' \ : \ u \xrightarrow{d'} v \in L\}$ and for every $x \in U$, if $M_{d-1}(x)$ is not empty and either $d = 0$ or there exists a link $x \xrightarrow{d} y \in L$:
    - Add to $\mathcal{B}$ the smallest bracket B which covers all brackets in $M_{d-1}(x)$ and all $y \in U$ such that $x \xrightarrow{d} y \in L$.
    - Add B to $M_d(z)$ for each $z \in B$ such that $z = x$ or:
        1. There exists $z \xrightarrow{d'} w \in L$ for some $d \geq d'$.
        2. There are no $d < d'$ and $z \xrightarrow{d'} w \in L$ such that $w \in B$.
- Output $\mathcal{B}$ and $M_d(x)$ for all $x \in U$ and $0 \leq d$.

When this algorithm is applied to a representative subset R of some bracketing $\mathcal{C}$, the following theorem guarantees that the bracketing produced by the algorithm is equal to $\mathcal{C}$:

5.2.2. LEMMA. *Let $\mathcal{C}$ be a bracketing over an utterance U and let R be a representative subset of $R_{\mathcal{C}}$. Let $\mathcal{B}$ be the bracketing produced by Algorithm 5.2.1 from R, then $\mathcal{B} = \mathcal{C}$.*

The proof of this theorem makes use of two lemmas. For a set R of common cover links on U and for $[u,v] \subseteq U$, I write $R^{[u,v]} = \{x \xrightarrow{d} y \in R \ : \ x,y \in [u,v]\}$, which is the restriction of R to $[u,v]$.

5.2.3. LEMMA. *Let $\mathcal{C}$ be a bracketing over an utterance U and let R be a representative subset of $R_{\mathcal{C}}$. For $[u,v] \subseteq U$, let $\mathcal{B}$ and $M_d(x)$ ($x \in [u,v]$, $0 \leq d$) be the output of Algorithm 5.2.1 when applied to the links $R^{[u,v]}$ and the utterance $[u,v]$. For every $x \in [u,v]$ and $0 \leq d$, either $M_d(x)$ is empty or $M_d(x) = \{B^{\mathcal{C}}_d(x) \cap [u,v]\}$ and x is of minimal depth (in $\mathcal{C}$) under $B^{\mathcal{C}}_d(x)$.*

PROOF. I write $B_d(x)$ for $B^{\mathcal{C}}_d(x)$ and $\hat{B}_d(x)$ for $B^{\mathcal{C}}_d(x) \cap [u,v]$. First, I show that if $X \in M_d(x)$, $X = \hat{B}_d(y)$ and y is of minimal depth under $B_d(y)$ (in $\mathcal{C}$) then $X = \hat{B}_d(x)$ and x is of minimal depth under $B_d(x)$ (in $\mathcal{C}$). If $x = y$ then this is trivial. Assume, therefore, that $x \neq y$. Since $\hat{B}_d(y) \in M_d(x)$, $x \in B_d(y)$ and there is a link $x \xrightarrow{d'} z \in R$ for some $d < d'$ such that $z \notin B_d(y)$. This implies that $x \in B_d(y) \subset B_{d'}(x)$ and that x is of minimal depth under $B_{d'}(x)$. By Lemma 5.1.10, x is also of minimal depth under $B_d(y)$ which implies that $B_d(y) = B_d(x)$. Therefore, $X = \hat{B}_d(x)$, as required.

It remains to show that if $X \in M_d(x)$ then there exists y such that $X = \hat{B}_d(y)$ and y is of minimal depth under $B_d(y)$ (in $\mathcal{C}$). The proof is by induction on d.

For $d = 0$, if $X \in M_0(x)$ then, by definition and Lemma 5.1.13, there is $y \in [u,v]$ such that $X = \{y\} \cup \{z \ : \ y \xrightarrow{0} z \in R^{[u,v]}\} = (\{y\} \cup \{z \ : \ y \xrightarrow{0} z \in R\}) \cap [u,v]$. This bracket is $B_0(y) \cap [u,v] = \hat{B}_0(y)$. Trivially, y is of minimal depth under $B_0(y)$, which completes the proof of the induction basis.

Now assume that the lemma holds for $d-1$ and we prove it for d. Let $X \in M_d(x)$. By definition, there is a y such that $M_{d-1}(y)$ is not empty, there is a link $y \xrightarrow{d} z \in R^{[u,v]}$ and X is the smallest bracket covering $\bigcup M_{d-1}(y)$ and $\{z \ : \ y \xrightarrow{d} z \in R^{[u,v]}\}$. By the induction hypothesis, $M_{d-1}(y) = \{\hat{B}_{d-1}(y)\}$ and therefore, by Lemma 5.1.13 $X = (B_{d-1}(y) \cap [u,v]) \cup (\{z \ : \ y \xrightarrow{d} z \in R\} \cap [u,v])$ which is $\hat{B}_d(y)$. Because $y \xrightarrow{d} z \in R^{[u,v]}$, y is of minimal depth under $B_d(y)$. This completes the proof. □

5.2.4. LEMMA. *Let $\mathcal{C}$ be a bracketing over an utterance U and let R be a representative subset of $R_{\mathcal{C}}$. For $[u,v] \subseteq U$, let $\mathcal{B}$ and $M_d(x)$ ($x \in [u,v]$, $0 \leq d$) be the output of Algorithm 5.2.1 when applied to the links $R^{[u,v]}$ and the utterance $[u,v]$. The following holds:*

1. *For every $X \in \mathcal{B}$ there is a $Y \in \mathcal{C}$ such that $X = Y \cap [u,v]$.*
2. *If $X \in \mathcal{C}$ and $X \subseteq [u,v]$ then $X \in \mathcal{B}$.*

PROOF. I write $B_d(x)$ for $B_d^{\mathcal{C}}(x)$ and $\hat{B}_d(x)$ for $B_d^{\mathcal{C}}(x) \cap [u,v]$.

1. If $X \in \mathcal{B}$ then by the definition of the algorithm, there is $x \in [u,v]$ and $0 \leq d$ such that $X \in M_d(x)$. By Lemma 5.2.3, $X = B_d(x) \cap [u,v]$, as required.
2. If $X \in \mathcal{C}$ and $X \subseteq [u,v]$ then $X = B_d(x)$ for some representative $x \in [u,v]$ of X in R. This means that either $d = 0$ or there is a link $x \xrightarrow{d} y \in R^{[u,v]}$. In either case, the algorithm creates a bracket from x and places it in $\mathcal{B}$ and $M_d(x)$. By Lemma 5.2.3 this bracket is $B_d(x) \cap [u,v] = B_d(x)$, which completes the proof.

□

The proof of Lemma 5.2.2 is now trivial.

PROOF. This follows directly from Lemma 5.2.4 by taking $[u,v] = U$. □

A simple corollary of this lemma is that the brackets $\mathcal{B}$ produced by the algorithm on $[u,v]$ are non-crossing.

5.2.5. COROLLARY. *Let $\mathcal{C}$ be a bracketing over an utterance U and let R be a representative subset of $R_{\mathcal{C}}$. For $[u,v] \subseteq U$, let $\mathcal{B}$ be the brackets output by Algorithm 5.2.1 when applied to the links $R^{[u,v]}$ and the utterance $[u,v]$. The brackets in $\mathcal{B}$ are non-crossing.*

PROOF. This follows directly from Lemma 5.2.4. □

5.2.2 A Version of Linear Transitivity

This section presents a version of linear transitivity lemma which makes use of Algorithm 5.2.1. This form of linear transitivity is given in Lemma 5.2.7. Its proof makes use of the following corollary of the original linear transitivity lemma (Lemma 5.1.16).

5.2.6. Corollary. *Let $R_{\mathcal{C}}$ be the common cover link set of a bracketing $\mathcal{C}$. If $y \in (x, z)$, $x \xrightarrow{d_1} y \in R_{\mathcal{C}}$ and $y \xrightarrow{d_2} z \in R_{\mathcal{C}}$ then there is a common cover link from $x \xrightarrow{d} z \in R_{\mathcal{C}}$ and the following holds:*

1. *If $d_1 \geq d_2$ then $d = d_1$.*
2. *If $d_1 < d_2$ then if $x \notin B_{d_2-1}(y)$ then $d = d_1$ and otherwise $d = d_2$.*

Proof. Assume the assumptions of the corollary hold. That there is a link $x \xrightarrow{d} z \in R_{\mathcal{C}}$ and that if $d_1 \geq d_2$ then $d = d_1$ is immediate from Lemma 5.1.16.

Assume therefore that $d_1 < d_2$. Since $y \xrightarrow{d_2} z \in R_{\mathcal{C}}$, y is of minimal depth under $B_{d-2}(y)$ and (by Lemma 5.1.10) also under $B_{d'}(y)$ for any $d' \leq d_2$. If $x \in B_{d_2-1}(y)$ this implies that there is $y \xrightarrow{d'} x \in R_{\mathcal{C}}$ for some $d' < d_2$ and by linear transitivity, $d = \max d_1, d_2 = d_2$. If $x \notin B_{d_2-1}(y)$ then, since $d_1 < d_2$, $x \notin B_{d_1}(y)$ and therefore $y \xrightarrow{d_1} x \notin R_{\mathcal{C}}$. By Lemma 5.1.14, $y \to x \notin R_{\mathcal{C}}$ and, by linear transitivity, $d = d_1$. $\square$

It is now possible to state and prove the modified linear transitivity lemma. I continue to use the notation $R(S)^{[x,z)} = \{v \xrightarrow{d} w \in R(S) \ : \ v, w \in [x, z)\}$.

5.2.7. Lemma. *Let $\mathcal{C}$ be a bracketing over an utterance U and let S be a shortest common cover link set of $R_{\mathcal{C}}$. Let $R(S)$ be the minimal representative subset which contains S. Let $\mathcal{B}$ be the set of brackets output by Algorithm 5.2.1 when applied to the links $R(S)^{[x,z)}$ and the utterance $[x, z) \subseteq U$. If $y \in (x, z)$, $x \xrightarrow{d_1} y \in R_{\mathcal{C}}$ and $y \xrightarrow{d_2} z \in R_{\mathcal{C}}$ then there is a link $x \xrightarrow{d} z \in R_{\mathcal{C}}$ such that:*

1. *If $d_1 \geq d_2$ then $d = d_1$.*
2. *If $d_1 < d_2$ then if $B^{\mathcal{B}}_{d_2-1}(y)$ exists and $x \notin B^{\mathcal{B}}_{d_2-1}(y)$ then $d = d_1$. Otherwise $d = d_2$.*

Proof. That $B^{\mathcal{B}}_{d_2-1}(y)$ is well-defined follows from Corollary 5.2.5. The lemma follows from Corollary 5.2.6. All that needs to be shown is that under the conditions of the lemma $x \notin B^{\mathcal{C}}_{d_2-1}(y)$ iff $B^{\mathcal{B}}_{d_2-1}(y)$ exists and $x \notin B^{\mathcal{B}}_{d_2-1}(y)$.

Assume first that $x \notin B^{\mathcal{C}}_{d_2-1}(y)$. Since $y \xrightarrow{d_2} z \in R_{\mathcal{C}}$ and $y \in (x, z)$ it follows that $B^{\mathcal{C}}_{d_2-1}(y) \subseteq [x, z)$. By Lemma 5.2.4, $B^{\mathcal{C}}_{d'}(y) \in \mathcal{B}$ for every $d' \leq d_2 - 1$. This means that $B^{\mathcal{B}}_{d_2-1}(y)$ exists and $B^{\mathcal{B}}_{d_2-1}(y) \subseteq B^{\mathcal{C}}_{d_2-1}(y)$ and therefore also $x \notin B^{\mathcal{B}}_{d_2-1}(y)$, as required.

To prove the other direction, assume that $B^{\mathcal{B}}_{d_2-1}(y)$ exists and $x \notin B^{\mathcal{B}}_{d_2-1}(y)$. By Lemma 5.2.4 this means that there is a sequence of brackets $X_1 \subset X_2 \subset \ldots \subset X_{d_2-1}$ in $\mathcal{C}$ such that for each $1 \leq i \leq d_2 - 1$, $B^{\mathcal{B}}_i(y) = X_i \cap [x, z)$. Since these are all different brackets covering y it follows that $B^{\mathcal{C}}_{d_2-1}(y) \subseteq X_{d_2-1}$. Since $x \notin X_i$, also $x \notin B^{\mathcal{C}}_{d_2-1}(y)$, which completes the proof. $\square$

5.2.3 Putting It All Together

The version of the transitivity lemma given by Lemma 5.2.7 allows us to calculate $\mathcal{C}$ from any shortest common cover link set S of $R_{\mathcal{C}}$ by using Algorithm 5.2.1. Working incrementally with words further and further apart, the set $R(S)$ can be calculated as follows. For two words x, z we can assume that $R(S)^{[x,z)}$ and $R(S)^{[z,x)}$ have already been calculated. Whether there are links $x \overset{d}{\rightarrow} z$ and $z \overset{d}{\rightarrow} x$ in $R_{\mathcal{C}}$ and what their depth is can then be determined by the transitivity lemma given above (Lemma 5.2.7). Whether the link is in $R(S)$ can then be determined (just as in the limited depth case) by Lemma 5.1.21. The set $R(S)^{[z,x]}$ can then be calculated by Algorithm 5.2.1 and the process can be continued. In this way we find $R(S)$ and the bracketing $\mathcal{C}$ can be calculated using Algorithm 5.2.1.

5.3 Characterization of Shortest Common Cover Link Sets

In this section I give a characterization of shortest common cover link sets which contain only links of depth 0 and 1. I will refer to sets of common cover links which contain only link of depth 0 and 1 as 0,1-common cover link sets (and will refer more specifically to 0,1-shortest common cover link sets, 0,1-$R_{\mathcal{C}}$, etc.). Recall that by Lemma 5.1.23 it does not matter whether the 0,1 restriction is imposed on the common cover link set $R_{\mathcal{C}}$ or on the shortest common cover link set.

5.3.1 The Characterization

I begin with the definition of the characterizing properties and the theorem which shows that these properties indeed characterize the 0,1-shortest common cover link sets.

5.3.1 (2.7.1). Definition. [characterization] Let L be a set of common cover links of depth 0 or 1 over U. The set L is said to satisfy the *characterizing conditions* if for every $w, x, y, z \in U$ the following conditions hold:

1. *Monotonicity*: if $y \in (x, z]$, $x \xrightarrow{d_1} y \in L$ and $x \xrightarrow{d_2} z \in L$ then $d_1 \leq d_2$.
2. *Minimality*: if $x \xrightarrow{d_1} z \in L$ then there is no $y \in (x, z)$ such that $x \xrightarrow{L} y$ and $y \xrightarrow{L} z$.
3. *Connectedness*: if $x \xrightarrow{L} z$ and $y \in (x, z)$ then $x \xrightarrow{L} y$.
4. *Blocking*: if $w \xrightarrow{d_1} z \in L$ and, for some $y \in U$ and $x \in (w, z)$, $x \xrightarrow{L} w$ and $x \xrightarrow{d_2} y \in L$ then $d_1 = 1$ and $d_2 = 0$.
5. *Equality*: if $y \in [x, z)$, $x \xrightarrow{d_1} z \in L$, $z \xrightarrow{d_2} y \in L$ and $y \xrightarrow{L} x$ then $d_1 = d_2$.
6. *Resolution*: if $y \in (x, z)$, $x' \in (x, y]$, $x \xrightarrow{d_1} x' \in L$ and $x' \xrightarrow{L} y$ and if $z' \in (z, y]$, $z \xrightarrow{d_2} z' \in L$ and $z' \xrightarrow{L} y$ then there is $v \in [x, z]$ such that either $x \xrightarrow{d_1} v \in L$ and $v \xrightarrow{L} z$ or $z \xrightarrow{d_2} v \in L$ and $v \xrightarrow{L} x$.

5.3.2 (2.7.2). Theorem (characterization). *Let L be a set of common cover links of depth 0 or 1 over U. There exists a bracketing $\mathcal{C}$ over U such that L is a shortest common cover link set of $\mathcal{C}$ iff L satisfies the characterizing conditions of Definition 5.3.1.*

Because of the many different conditions involved, the proof of this theorem is quite long. I therefore split the proof into two parts. Section 5.3.2 shows that the characterization conditions are necessary and section 5.3.3 shows that they are sufficient. Before I begin with this proof, it is best to first prove a few simple properties of the characterization. First, I show that the characterization contains not redundant properties.

5.3.3 (2.7.3). Lemma. *For each of the six conditions in Definition 5.3.1, there is a set of links L which violates that condition but satisfies all other conditions.*

Proof. For each condition, I give below an example which violates that condition and satisfies all other conditions:

1. Monotonicity: $$\begin{tikzcd} x \arrow[r, "1"'] \arrow[rr, bend left, "0"] & y & z \end{tikzcd}$$
2. Minimality: $$\begin{tikzcd} x \arrow[r, "0"'] \arrow[rr, bend left, "0"] & y \arrow[r, "0"'] & z \end{tikzcd}$$

3. Connectedness: $\begin{tikzcd} x \arrow[rr, bend left, "0"] & y & z \end{tikzcd}$

4. Blocking: Several different violation examples are given here (these are not needed for the proof but are interesting in themselves):

$$\begin{tikzcd} w = y \arrow[r, shift left, "0"] \arrow[rr, bend left, "0"] & x \arrow[l, shift left] & z \end{tikzcd} \qquad \begin{tikzcd} w \arrow[r, shift left, "0"] \arrow[rrr, bend left, "1"] & x \arrow[l, shift left] \arrow[r, "1"] & y & z \end{tikzcd}$$

$$\begin{tikzcd} y & w \arrow[r, shift left, "0"] \arrow[rr, bend left, "1"] & x \arrow[l, shift left] \arrow[ll, bend right, "1"'] & z \end{tikzcd}$$

5. Equality: Two examples are given here, one for the case $x = y$ and one for the case $x \neq y$:

$$\begin{tikzcd} x = y \arrow[r, shift left, "0"] & z \arrow[l, shift left, "1"] \end{tikzcd} \qquad \begin{tikzcd} x \arrow[r, shift left, "0"] \arrow[rr, bend left, "1"] & y \arrow[l, shift left] & z \arrow[l, "0"'] \end{tikzcd}$$

6. Resolution: Here, too, several examples are given:

$$\begin{tikzcd} x \arrow[r, "0"] & y & z \arrow[l, "0"'] \end{tikzcd} \qquad \begin{tikzcd} x \arrow[r, "0"] \arrow[rr, bend left, "1"] & y & z \arrow[l, "0"'] \end{tikzcd}$$

$$\begin{tikzcd} x \arrow[r, "0"] \arrow[rr, bend left, "1"] & y & z \arrow[l, "0"'] \arrow[ll, bend left, "1"] \end{tikzcd}$$

□

The next important property of the characterization is that if it is satisfied by the full set of links L then it must also be satisfied by $L^{[x,y]} = \{u \overset{d}{\rightarrow} v \in L \ : \ u, v \in [x, y]\}$, the restriction of L to a segment $[x, y] \subseteq U$.

5.3.4 (2.7.4). LEMMA. *Let L be a set of common cover links over an utterance U and let $L^{[x,y]} = \{u \overset{d}{\rightarrow} v \in L \ : \ u, v \in [x, y]\}$ be the restriction of L to a segment $[x, y] \subseteq U$. If L satisfies the characterizing conditions then so does $L^{[x,y]}$.*

PROOF. The only two conditions in which a certain configuration requires additional links are connectedness and resolution. In both these cases, the link is inside the segment of the utterance which caused the requirement. Therefore, if the conditions are satisfied on the full utterance, they are also satisfied on a segment of the utterance. □

5.3.2 Characterization Conditions are Necessary

5.3.5. Lemma (Characterization Conditions are Necessary). *Let $\mathcal{C}$ be a bracketing such that $R_\mathcal{C}$ contains links of depth 0 or 1. If S is a shortest common cover link set for $R_\mathcal{C}$, then the conditions in Definition 5.3.1 hold for S.*

Proof. Assume that S is as in the assumptions of the lemma and let $R(S)$ be the minimal representative subset of $R_\mathcal{C}$ such that $S \subseteq R(S)$.

1. *Monotonicity*: By the definition of $R_\mathcal{C}$, there cannot be two links from x to y with different depths. For $y = z$ this proves the property immediately and for $y \in (x, z)$ it proves the property using Lemma 5.1.12.

2. *Minimality*: Assume $y \in (x, z)$, $x \xrightarrow{d_1} z \in S$, $x \xrightarrow{S} y$ and $y \xrightarrow{S} z$. There is $y' \in (x, z)$ such that $x \xrightarrow{S} y'$ and $y' \xrightarrow{d_2} z \in S$ (for some d_2). By Lemma 5.1.28, there exists d_3 such that $x \xrightarrow{d_3} y' \in R_\mathcal{C}$. By monotonicity, $d_3 \leq d_1$ and since $d_1 \leq 1$ either $d_3 = 0$ or $d_3 = d_1 = 1$. In either case, since $x \xrightarrow{d_1} z \in S$ it follows that $x \xrightarrow{d_3} y' \in R(S)$. Together, $x \xrightarrow{d_3} y' \in R(S)$, $y' \xrightarrow{d_2} z \in S$ and $x \xrightarrow{d_1} z \in S$ which is in contradiction to the definition of a shortest common cover link set.

3. *Connectedness*: Assume first that $x \xrightarrow{d_1} z \in S$ and $y \in (x, z)$. By Lemma 5.1.12, there exists $d_2 \leq d_1$ such that $x \xrightarrow{d_2} y \in R_\mathcal{C}$. Since $d_1 \leq 1$, either $d_2 = 0$ or $d_1 = d_2 = 1$. In either case $x \xrightarrow{d_2} y \in R(S)$ which implies, by Lemma 5.1.28, that $x \xrightarrow{S} y$, as required.

   Assume now that $x \xrightarrow{S} z$ and $y \in (x, z)$. Let $x' \xrightarrow{d_1} z' \in S$ be a link in an S-path from x to z such that $y \in [x', z')$. By definition, $x \xrightarrow{S} x'$. If $y = x'$, we are done. Otherwise, by what has been shown above, $x' \xrightarrow{S} y$ and together with $x \xrightarrow{S} x'$ we get $x \xrightarrow{S} y$.

4. *Blocking*: Assume that the assumptions of the property hold. I begin by showing that $d_1 = 1$. Assume, by contradiction, that $d_1 = 0$. By Lemma 5.1.28, $x \xrightarrow{S} w$ implies $x \xrightarrow{d} w \in R_\mathcal{C}$. This together with $d_1 = 0$ implies, by Lemma 5.1.17, that $d = 0$ and, therefore, $x \xrightarrow{0} w \in R(S)$. By transitivity (Lemma 5.1.15), $x \xrightarrow{0} z \in R_\mathcal{C}$ and because the depth of the link is 0, $x \xrightarrow{0} z \in R(S)$. By connectedness, $w \xrightarrow{S} x$, which implies that $w \xrightarrow{d'} x \in R_\mathcal{C}$. Since $x \xrightarrow{0} w \in R_\mathcal{C}$, it follows from Lemma 5.1.14 that $d' = 0$ and therefore $w \xrightarrow{0} x \in R(S)$. This together with $x \xrightarrow{0} z \in R(S)$ contradicts the assumption that $w \xrightarrow{0} z \in S$. This completes the proof that $d_1 = 1$.

Next, I show that $d_2 = 0$. Assume, by contradiction, that $d_2 = 1$. Since $x \xrightarrow{S} w$, $x \xrightarrow{d} w \in R_{\mathcal{C}}$. Applying transitivity (Lemma 5.1.15) to this link together with $w \xrightarrow{1} z \in S$ implies that $x \xrightarrow{d'} z \in R_{\mathcal{C}}$. Since, by assumption, $x \xrightarrow{1} y \in S$, also $x \xrightarrow{d'} z \in R(S)$ (whether $d' = 0$ or $d' = 1$). By connectedness, $w \xrightarrow{S} x$ and therefore $w \xrightarrow{d''} x \in R_{\mathcal{C}}$. Since it was shown that $d_1 = 1$, $w \xrightarrow{1} z \in S$ and therefore $w \xrightarrow{d''} x \in R(S)$. Together, $w \xrightarrow{d''} x \in R(S)$, $x \xrightarrow{d'} z \in R(S)$ and $w \xrightarrow{1} z \in S$ contradict the definition of S as a shortest common cover link set.

5. *Equality*: Assume $y \in [x, z)$, $x \xrightarrow{d_1} z \in S$ and $z \xrightarrow{d_2} y \in S$. If $y = x$ then $d_1 = d_2$ by Lemma 5.1.14. Assume now that $y \neq x$ and $y \xrightarrow{S} x$. Since $y \xrightarrow{S} x$, it follows from Lemma 5.1.28 that $y \xrightarrow{d_3} x \in R_{\mathcal{C}}$. By linear transitivity (Lemma 5.1.16) and Lemma 5.1.14, $z \xrightarrow{d_1} x \in R_{\mathcal{C}}$. By connectedness, $x \xrightarrow{S} y$. If $y \xrightarrow{0} z \in R_{\mathcal{C}}$ then $y \xrightarrow{0} z \in R(S)$ and (by Lemma 5.1.28) $y \xrightarrow{S} z$, which together with $x \xrightarrow{S} y$ and $x \xrightarrow{d_1} z \in S$ contradicts minimality. Therefore, either $y \xrightarrow{1} z \in R_{\mathcal{C}}$ or there is no link from y to z in $R_{\mathcal{C}}$. In either case linear transitivity (applied to $z \xrightarrow{d_1} x \in R_{\mathcal{C}}$, $z \xrightarrow{d_2} y \in S$ and $y \xrightarrow{d_3} x \in R_{\mathcal{C}}$) implies that $d_1 = d_2$.

6. *Resolution*: Assume that $y \in (x, z)$, $x' \in (x, y]$, $x \xrightarrow{d_1} x' \in L$, $x' \xrightarrow{L} y$ and $z' \in (z, y]$, $z \xrightarrow{d_2} z' \in L$, $z' \xrightarrow{L} y$. Since $x \xrightarrow{S} y$ and $z \xrightarrow{S} y$, it follows from Lemma 5.1.28 that there are links $x \xrightarrow{d_{xy}} y \in R_{\mathcal{C}}$ and $z \xrightarrow{d_{zy}} y \in R_{\mathcal{C}}$. Therefore, $y \in B_{d_{xy}}(x) \cap B_{d_{zy}}(z)$ and, because brackets do not cross, either $B_{d_{xy}}(x) \subseteq B_{d_{zy}}(z)$ or $B_{d_{zy}}(z) \subseteq B_{d_{xy}}(x)$. This means that either $x \xrightarrow{d_{xy}} z \in R_{\mathcal{C}}$ or $z \xrightarrow{d_{zy}} x \in R_{\mathcal{C}}$. Assume (without loss of generality) that $x \xrightarrow{d_{xy}} z \in R_{\mathcal{C}}$.

   If $d_{xy} = 0$ then $x \xrightarrow{0} y \in R(S)$, $x \xrightarrow{0} z \in R(S)$ and (by Lemma 5.1.28) $x \xrightarrow{S} z$. The link $x \xrightarrow{d_1} x' \in S$ is the first link on an S-path from x to y. Let v be such that $x \xrightarrow{d} v \in S$ is the first link on the S-path from x to z. By Lemma 5.1.28, $x \xrightarrow{0} y \in R(S)$ implies that $d_1 = 0$ and $x \xrightarrow{0} z \in R(S)$ implies that $d = 0$ and therefore the resolution property holds.

   Assume now that $d_{xy} = 1$. Let $w_1 \xrightarrow{1} w_2 \in S$ be the first link of depth 1 on an S-path from x to y (by linear transitivity such a link exists). Since $w_1 \xrightarrow{1} w_2 \in S$ and $w_2 \xrightarrow{S} y$, it follows that $w_1 \xrightarrow{1} y \in R(S)$. If $w_1 = x$, then $x \xrightarrow{1} y \in R(S)$ and therefore also $x \xrightarrow{1} z \in R(S)$, which implies (by Lemma 5.1.28) that $x \xrightarrow{S} z$. Let $x \xrightarrow{d} v \in S$ be the first link in an S-path from x to z. If $v \in (x, w_2)$ then, by connectedness, $v \xrightarrow{S} w_2$, which together with $x \xrightarrow{d} v \in S$ and $x \xrightarrow{1} w_2 \in S$ contradicts minimality. Therefore, $v \in [w_2, z]$

and monotonicity (from $x \xrightarrow{1} w_2 \in S$) implies that $d = 1$. This completes the proof in this case.

It remains to consider the case where $w_1 \neq x$. By the choice of w_1 and w_2, $x \xrightarrow{0} w_1 \in R(S)$ and this together with $w_1 \xrightarrow{1} y \in R(S)$ (shown above) and $x \xrightarrow{1} y \in R_{\mathcal{C}}$ implies (by linear transitivity, Lemma 5.1.16) that $w_1 \xrightarrow{0} x \in R(S)$. Because $x \xrightarrow{1} z \in R_{\mathcal{C}}$ it follows from transitivity (Lemma 5.1.15) that $w_1 \xrightarrow{d'} z \in R_{\mathcal{C}}$ and because $y \in (w_1, z)$ and $w_1 \xrightarrow{1} y \in R(S)$ it follows that $w_1 \xrightarrow{1} z \in R(S)$. Therefore, by Lemma 5.1.28, $w_1 \xrightarrow{S} z$. Similarly, because $x \xrightarrow{0} w_1 \in R(S)$, $x \xrightarrow{S} w_1$. Together with $w_1 \xrightarrow{S} z$ this means that $x \xrightarrow{S} z$. Let $x \xrightarrow{d} v \in S$ be the first link in the S-path from x to z thus constructed. The link $x \xrightarrow{d} v \in S$ is also the first link on an S-path from x to w_1 and, since $x \xrightarrow{0} w_1 \in R(S)$, it follows (by linear transitivity) that $d = 0$ ($x \xrightarrow{0} v \in S$). It remains to show that $d_1 = 0$ (that is, $x \xrightarrow{0} x' \in S$). By monotonicity, it is enough to show that $x' \in (x, v]$. Assume, by contradiction, that $v \in (x, x')$. By connectedness (from $v \xrightarrow{S} z$), $v \xrightarrow{S} x'$ which together with $x \xrightarrow{0} v \in S$ and $x \xrightarrow{d_1} x' \in S$ contradicts minimality. Therefore, $x \xrightarrow{0} x' \in L$ and this completes the proof.

□

5.3.3 Characterization Conditions are Sufficient

To prove that the characterizing conditions in Definition 5.3.1 are sufficient for a set of links L to be a shortest common cover link set for some bracketing, I first define a set $P(L)$ of links based on the link paths in L. I then use the simple reconstruction algorithm (Algorithm 5.1.29) to create a set of brackets from this set of links. After showing that this set of brackets is a bracketing, I prove that L is a shortest common cover link set for this bracketing.

The Link Sets $P^*(L)$ and $P(L)$

This section defines the sets of links $P^*(L)$ and $P(L)$ which are based on L-paths. It then proves a series of properties of these sets of links. These properties are of two types. First, variants of the characterizing conditions are shown to hold for $P^*(L)$ and $P(L)$. Next, different transitivity properties of $P^*(L)$ and $P(L)$ are proved.

5.3.6. Definition. [path links] Let L be a set of common cover links over an utterance U. The following sets of common cover links are induced by L:

1. The set $P^*(L)$ of *candidate path links* induced by L is a set of common cover links such that $x \xrightarrow{d} y \in P^*(L)$ iff $x \neq y$ and $x \xrightarrow{L} y$. The depth d is 1 if the following condition holds and 0 otherwise:

   there are $u \in [x, y)$ and $v \in (u, y]$ such that $u \xrightarrow{1} v \in L$, $x \xrightarrow{L} u$, $u \xrightarrow{L} x$ and $v \xrightarrow{L} y$.

2. The set $P(L)$ of *path links* is the subset of $P^*(L)$ such that

   (a) $x \xrightarrow{0} y \in P(L) \iff x \xrightarrow{0} y \in P^*(L)$.

   (b) $x \xrightarrow{1} y \in P(L) \iff x \xrightarrow{1} y \in P^*(L)$ and there is some z such that $x \xrightarrow{1} z \in L$.

5.3.7. Lemma (unique link path). *Let L be a 0,1-common cover link set over an utterance U. If L satisfies the monotonicity, minimality and connectedness characterizing properties of Definition 5.3.1 then for every $x, y \in U$ there is at most one L-path from x to y.*

Proof. Assume the assumptions of the lemma hold and assume, by contradiction, that there are two different L-paths from x to y. By monotonicity, for every two words x and y there is at most one link from x to y. Therefore, a link path is identified by the linear path it is based on. If the link paths are different then they should also be based on different linear paths, P_1 and P_2. If $P_1 \subset P_2$ then the minimality property is violated. Therefore, there is $z \in P_1 \setminus P_2$. There are two consecutive words u, v in the path P_2 such that $z \in (u, v)$. Since u, v are consecutive in P_2, there is d such that $u \xrightarrow{d} v \in L$. By connectedness, $u \xrightarrow{L} z$ and $z \xrightarrow{L} v$. These links together with the link $u \xrightarrow{d} v \in L$ violate minimality and therefore contradict the assumptions. $\square$

This uniqueness property will allow me, for now on, to simply refer to "the path" in L from x to y. As mentioned in the proof, the monotonicity property implies that given a 0,1-set of links L which satisfies the characterizing properties of Definition 5.3.1, an L-path can be identified with the linear path it is based on. This will be used in the following proofs.

5.3.8. Corollary. *Let L be a 0,1-common cover link set over an utterance U. If L satisfies the characterizing properties of Definition 5.3.1 then, for every $x, y \in U$, $P(L)$ contains at most one link from x to y.*

5.3.9. Corollary. *Let L be a 0,1-common cover link set over an utterance U. If L satisfies the characterizing properties of Definition 5.3.1, $x \xrightarrow{d} y \in P^*(L)$ and P is an L-path from x to y then $d = 1$ iff:*

there is $x_1 \xrightarrow{1} x_2 \in P$ such that $x_1 \xrightarrow{L} x$.

Proof. By Definition 5.3.6, if $x \xrightarrow{d} y \in P^*(L)$ then $d = 1$ iff there are $u \in [x, y)$ and $v \in (u, y]$ such that $u \xrightarrow{1} v \in L$, $x \xrightarrow{L} u$, $u \xrightarrow{L} x$ and $v \xrightarrow{L} y$. This means that u and v are on an L-path from x to y and, by Lemma 5.3.7, this is the path P. □

5.3.10. Lemma (monotonicity of $P^*(L)$). *Let L be a 0,1-common cover link set over an utterance U. Assume that L satisfies the characterizing properties in Definition 5.3.1. If $y \in (x, z]$, $x \xrightarrow{d_1} y \in P^*(L)$ and $x \xrightarrow{d_2} z \in P^*(L)$ then $d_1 \leq d_2$.*

Proof. Assume the assumptions of the lemma hold. If $d_1 = 0$, there is nothing to prove. If $y = z$ then the claim follows directly from Corollary 5.3.8. Assume, therefore, that $y \neq z$ and $d_1 = 1$. Let P_1 be the L-path from x to y and let P_2 be the L-path from x to z. By Corollary 5.3.9, there is $x_1 \xrightarrow{1} x_2 \in P_1 \subseteq L$ such that $x_1 \xrightarrow{L} x$. Let y_1 and y_2 be consecutive words in P_2 such that y_2 is the first word in P_2 which is not in $[x, y]$. By connectedness, $x \xrightarrow{L} y_1$ and $y_1 \xrightarrow{L} y$, which implies, by path uniqueness (Lemma 5.3.7), that y_1 is on P_1. If $y_1 \in [x, x_1)$ then $x_1 \xrightarrow{L} x$ implies $x_1 \xrightarrow{L} y_1$ and blocking is violated. Therefore, $y_1 \in [x_1, y]$. If $y_1 = x_1$ then monotonicity implies that $y_1 \xrightarrow{1} y_2 \in L$ and since $x_1 \xrightarrow{L} x$ this implies $x \xrightarrow{1} z \in P^*(L)$, as required. If $y_1 \in (x_1, y]$ then, since y_1 is on P_1, $x_2 \in (x_1, y_1]$ and it follows from path uniqueness that $x_1 \xrightarrow{1} x_2 \in P_2$. therefore, by the definition of $P^*(L)$, $x \xrightarrow{1} z \in P^*(L)$, as required. □

5.3.11. Lemma (connectedness of $P(L)$). *Let L be a 0,1-common cover link set over an utterance U. Assume that L satisfies the characterizing properties in Definition 5.3.1. If $y \in (x, z)$ and $x \xrightarrow{d} z \in P(L)$ then there is d' such that $x \xrightarrow{d'} y \in P(L)$.*

Proof. This follows directly from the connectedness of L, link monotonicity (Lemma 5.3.10) and the definition of $P(L)$. □

5.3.12. Lemma. *Let L be a 0,1-common cover link set over an utterance U. Assume that L satisfies the characterizing properties in Definition 5.3.1. If $x \overset{L}{\to} y$, $z \overset{L}{\to} x$ and $w_1 \overset{0}{\to} w_2 \in L$ is a link on the L-path from x to y then there is no word $u \in (w_1, w_2)$ on the L-path from z to x.*

Proof. Assume the assumptions of the lemma hold. If $w_1 \notin [x, z)$ then the lemma is obvious. Assume, therefore, that $w_1 \in [x, z)$. Assume, by contradiction, that there is $u \in (w_1, w_2)$ on the L-path from z to x. Since u is on the path from z to x, $u \overset{L}{\to} x$ and by connectedness it follows that $u \overset{L}{\to} w_1$. This together with the link $w_1 \overset{0}{\to} w_2$ contradicts blocking. □

5.3.13. Lemma. *Let L be a 0,1-common cover link set over an utterance U. Assume that L satisfies the characterizing properties of Definition 5.3.1. If $x \overset{L}{\to} y$, $z \overset{L}{\to} x$, $w_1 \in [x, z]$ and $w_1 \overset{1}{\to} w_2 \in L$ is a link on the L-path from x to y, then w_1 is on the L-path from z to x.*

Proof. Assume the assumptions of the lemma hold. If $w_1 = z$ then the lemma is obvious. Assume, therefore, that $w_1 \in [x, z)$. Let u_1 and u_2 be two consecutive words on the L-path from z to x such that u_2 is the first word in this path which is in $[x, w_1]$. We need to show that $u_2 = w_1$. Assume, by contradiction, that $u_2 \in [x, w_1)$. By assumption, $u_1 \in (w_1, z]$. Since w_1 is on the path from x to y, $w_1 \overset{L}{\to} y$ and by connectedness $w_1 \overset{L}{\to} u_1$. This link together with $w_1 \overset{1}{\to} w_2 \in L$ and $u_1 \overset{d}{\to} u_2 \in L$ contradict blocking. This shows that $u_2 = w_1$ and completes the proof. □

The following example shows that when $x \overset{L}{\to} y$ and $y \overset{L}{\to} x$ it is possible to have a link $w_1 \overset{0}{\to} w_2 \in L$ on the L-path from x to y such that neither w_1 nor w_2 is on the L-path from y to x:

$$\begin{tikzcd} x \arrow[r, "1"] & w_1 \arrow[r, shift left, "0"] & w_2 \arrow[l, shift left, "0"] \arrow[r, shift left, "0"] & y \arrow[l, shift left, "0"] \arrow[lll, bend right, "1"'] \end{tikzcd}$$

5.3.14. Corollary. *Let L be a 0,1-common cover link set over an utterance U. Assume that L satisfies the characterizing properties of Definition 5.3.1. If $x \overset{0}{\to} y \in P(L)$ and $y \overset{0}{\to} x \in P(L)$ then there is $z \in (x, y]$ such that $x \overset{0}{\to} z \in L$ and z is on the L-paths from x to y and from y to x.*

Proof. Because $x \overset{0}{\to} y \in P(L)$, the first link in the L-path from x to y is $x \overset{0}{\to} z \in L$ with $z \in (x, y]$. Let $u_1 \overset{d}{\to} u_2 \in L$ be the unique link in the L-path from y to x such that $z \in [u_1, u_2)$. If $d = 0$ then, by Lemma 5.3.12, $z = u_1$, which proves the claim. To complete the proof, I show that $d = 1$ leads to a contradiction. Assume $d = 1$. By Lemma 5.3.13, u_1 is on the path from x to y and, since $u_2 \in [x, z)$, it follows from Lemma 5.3.12 that $u_2 = x$. By equality, $d = 0$, contrary to the assumption. $\square$

5.3.15. Lemma (equality in $P^*(L)$). *Let L be a 0,1-common cover link set over an utterance U. Assume that L satisfies the characterizing properties of Definition 5.3.1. If $x \xrightarrow{d_1} y \in P^*(L)$ and $y \xrightarrow{d_2} x \in P^*(L)$ then $d_1 = d_2$.*

Proof. Assume the assumptions of the lemma hold. Because of the symmetry in the conditions on x and y it is sufficient to prove that if $d_1 = 1$ then $d_2 = 1$. Assume, therefore, that $d_1 = 1$. This means that there are two consecutive words w_1 and w_2 on the L-path from x to y such that $w_1 \xrightarrow{1} w_2 \in L$ and $w_1 \xrightarrow{L} x$. By Lemma 5.3.13, w_1 is on the L-path from y to x. Let u_1 and u_2 be two consecutive words on the L-path from y to x such that u_2 is the first word on this path which is in $[w_1, w_2)$. By connectedness (since w_2 is on the L-path from x to y and therefore $w_2 \xrightarrow{L} y$), $w_2 \xrightarrow{L} u_1$. Similarly by connectedness (since u_2 is on the L-path from y to x and therefore $u_2 \xrightarrow{L} x$), $u_2 \xrightarrow{L} w_1$. If $u_1 \in (w_2, y]$ then it follows by blocking from $w_2 \xrightarrow{L} u_1$ that $u_1 \xrightarrow{1} u_2 \in L$. If $u_1 = w_2$ then it follows by equality from $w_1 \xrightarrow{1} w_2 \in L$ and $u_2 \xrightarrow{L} w_1$ that $u_1 \xrightarrow{1} u_2 \in L$. By Lemma 5.3.13, u_1 is on the L-path from x to y and therefore $u_1 \xrightarrow{L} y$. Together with $u_1 \xrightarrow{1} u_2 \in L$ this shows that $d_2 = 1$. $\square$

The following lemma shows that linear transitivity holds for links in $P^*(L)$:

5.3.16. Lemma (linear transitivity of $P^*(L)$). *Assume L is a 0,1-common cover link set over an utterance U. Assume that L satisfies the characterizing properties of Definition 5.3.1. If $y \in (x, z)$, $x \xrightarrow{d_1} y \in P^*(L)$ and $y \xrightarrow{d_2} z \in P^*(L)$ then $x \xrightarrow{d} z \in P^*(L)$ where:*

1. *If there is a link $y \xrightarrow{d_3} x \in P^*(L)$ then $d = \max(d_1, d_2)$.*
2. *Otherwise, $d = d_1$.*

Proof. Assume the assumptions of the lemma hold. Since $x \xrightarrow{L} y$ and $y \xrightarrow{L} z$, it follows that $x \xrightarrow{L} z$ and therefore $x \xrightarrow{d} z \in P^*(L)$. If $d_1 = 1$ then there are two consecutive words w_1 and w_2 on the L-path from x to y such that $w_1 \xrightarrow{1} w_2 \in L$

and $w_1 \xrightarrow{L} x$. Since w_1 and w_2 are also on the path from x to z, it follows that $d = 1$, as required.

Assume now that $d_1 = 0$. To complete the proof we need to show that if $d = 1$ then $d_2 = 1$ and $y \xrightarrow{L} x$. Assume that $d = 1$. There are two consecutive words u_1 and u_2 on the path from x to z such that $u_1 \xrightarrow{1} u_2 \in L$ and $u_1 \xrightarrow{L} x$. If $u_1 \in [x, y)$ then (because the path from x to y is a prefix of the path from x to z) $d_1 = 1$, contrary to the assumption. Therefore, $u_1 \in [y, z)$ and since the path from y to z is a suffix of the path from x to z, u_1 and u_2 are consecutive words on the path from y to z. Because $y \in [u_1, x)$ and $u_1 \xrightarrow{L} x$, it follows from connectivity that $u_1 \xrightarrow{L} y$, which implies that $d_2 = 1$. □

The next lemma gives an additional transitivity property for $P^*(L)$. The general transitivity property (Lemma 5.1.15) holds for $R_{\mathcal{C}}$ but not necessarily for any of its representative subsets or shortest common cover link sets. This is due to the optionality of some of the depth 1 links. This is reflected in the following lemma in an extra condition (the link $y \xrightarrow{1} w \in L$) which is needed to ensure the existence of the link $y \xrightarrow{d'} z$ not only in $P(L)$ but even in $P^*(L)$.

5.3.17. Lemma (crossing transitivity of $P^*(L)$). *Let L be a 0,1-common cover link set over an utterance U. Assume that L satisfies the characterizing properties of Definition 5.3.1. If $y \in (x, z)$, $y \xrightarrow{L} x$ and $x \xrightarrow{d} z \in P^*(L)$ then:*

1. *If $d = 0$ then $y \xrightarrow{0} z \in P^*(L)$.*
2. *If $d = 1$ and there exists a link $y \xrightarrow{1} w \in L$ then $y \xrightarrow{d'} z \in P^*(L)$ for some $d' \leq 1$.*

Proof. Assume the assumptions of the lemma hold. The cases $d = 0$ and $d = 1$ are proved separately. First assume that $d = 0$. Let w_1 and w_2 be consecutive words on the L-path from x to z such that w_2 is the first word on the path such that $w_2 \notin [x, y]$. I will show that $w_1 = y$. Assume, by contradiction, that $w_1 \neq y$. This means that $w_1 \in [x, y)$. Because $y \xrightarrow{L} x$ it follows from connectedness that $y \xrightarrow{L} w_1$ and blocking implies that $w_1 \xrightarrow{1} w_2 \in L$. If $w_1 = x$ then $d = 1$, contradicting the assumptions. Therefore $w_1 \in (x, y)$. Now let v_1 and v_2 be two consecutive words on the L-path from y to x such that v_2 is the first word in the path such that $v_2 \notin [w_1, y]$. If $v_1 \neq w_1$ then because $w_1 \xrightarrow{L} z$ it follows from connectedness that $w_1 \xrightarrow{L} v_1$. But then $w_1 \xrightarrow{1} w_2 \in L$ and $v_1 \xrightarrow{d''} v_2 \in L$ contradict blocking. Therefore, $v_1 = w_1$ and it follows that $w_1 \xrightarrow{L} x$. But now, because $w_1 \xrightarrow{1} w_2 \in L$ is on the path from x to z, it follows from the definition of $P(L)$ that $d = 1$, which contradicts the assumption. This shows that $w_1 \neq y$ leads to a contradiction and therefore $w_1 = y$. Since $w_1 \xrightarrow{L} z$, this implies that $y \xrightarrow{d'} z \in P^*(L)$ for some d'. If $d' = 1$ then linear transitivity of $P^*(L)$ (Lemma 5.3.16) implies that $d = 1$, which contradicts the assumption. Therefore, $d' = 0$ and $y \xrightarrow{0} z \in P^*(L)$.

Now assume that $d = 1$ and there exists a link $y \xrightarrow{1} w \in L$. As before, let w_1 and w_2 be two consecutive words on the L-path from x to z such that w_2 is the first word on the path such that $w_2 \notin [x, y]$. It must be that $w_1 = y$ because otherwise the link from w_1 to w_2 together with $y \xrightarrow{L} w_1$ (from connectedness) and $y \xrightarrow{1} w \in L$ contradict blocking. Therefore, $y \xrightarrow{L} z$ and $y \xrightarrow{d'} z \in P^*(L)$ for some d'. □

5.3.18. Lemma (resolution in $P(L)$). *Let L be 0,1-common cover link set over an utterance U. Assume that L satisfies the characterizing properties of Definition 5.3.1. If $y \in (x, z)$, $x \xrightarrow{d_1} y \in P(L)$ and $z \xrightarrow{d_2} y \in P(L)$ then either $x \xrightarrow{d_1} z \in P(L)$ or $z \xrightarrow{d_2} x \in P(L)$.*

Proof. Assume the assumptions of the lemma hold and that $y \in (x, z)$, $x \xrightarrow{d_1} y \in P(L)$ and $z \xrightarrow{d_2} y \in P(L)$. The proof is by induction on $n = n_1 + n_2$ where n_1 and n_2 are the number of links in the L-paths from x to y and from z to y (respectively).

Basis: The basis of the induction is $n_1 = n_2 = 1$. This means that $x \xrightarrow{d_1} y \in L$ and $z \xrightarrow{d_2} y \in L$. By the resolution property, there exists $v \in (x, z)$ such that either $x \xrightarrow{d_1} v \in L$ and $v \xrightarrow{L} z$ or $z \xrightarrow{d_2} v \in L$ and $v \xrightarrow{L} x$. Assume, without loss of generality, that $x \xrightarrow{d_1} v \in L$ and $v \xrightarrow{L} z$. If there is no L-path from v to x then, by linear transitivity (Lemma 5.3.16), $x \xrightarrow{d_1} z \in P^*(L)$. Because $x \xrightarrow{d_1} y \in L$, it follows from the definition of $P(L)$ that $x \xrightarrow{d_1} z \in P(L)$, as required.

Assume, therefore, that $v \xrightarrow{L} x$. If $v \in (x, y)$ then, by connectedness (from $v \xrightarrow{L} z$), $v \xrightarrow{L} y$, which together with $x \xrightarrow{d_1} v \in L$ and $x \xrightarrow{d_1} y \in L$ contradicts minimality. Similarly, if $v \in (y, z)$ then, by connectedness (from $z \xrightarrow{d_2} y \in L$ and $v \xrightarrow{L} x$), $z \xrightarrow{L} v$ and $v \xrightarrow{L} y$ which together with $z \xrightarrow{d_2} y \in L$ contradicts minimality. Therefore, either $v = y$ or $v = z$. If $v = z$ then $x \xrightarrow{d_1} z \in L \subseteq P(L)$ and the claim holds. If $v = y$ then, by linear transitivity, if $d_1 = 1$ or if $y \xrightarrow{0} z \in P^*(L)$ then $x \xrightarrow{d_1} z \in P(L)$, as required. Assume, therefore, that $d_1 = 0$ and $y \xrightarrow{1} z \in P^*(L)$.

By Lemma 5.3.15, $z \xrightarrow{1} y \in L$ (that is, $d_2 = 1$). Because $v \xrightarrow{L} x$ and $y = v$, by linear transitivity, $z \xrightarrow{1} x \in P(L)$, which completes the proof of the induction basis.

Induction Step: Assume the claim holds up to $n-1$ and assume that $n_1+n_2 = n$. By definition, $x \xrightarrow{L} y$ and $z \xrightarrow{L} y$. Let $x \xrightarrow{d_x} x'$ and $z \xrightarrow{d_z} z'$ be the first links in the L-paths from x to y and from z to y (respectively). By the resolution property, there is $v \in (x, z)$ such that either $x \xrightarrow{d_x} v \in L$ and $v \xrightarrow{L} z$ or $z \xrightarrow{d_z} v \in L$ and $v \xrightarrow{L} x$. Assume, without loss of generality, that $x \xrightarrow{d_x} v \in L$ and $v \xrightarrow{L} z$. This implies that there exists d such that $x \xrightarrow{d} z \in P^*(L)$. If $d_1 = 1$ then, by monotonicity in $P^*(L)$ (Lemma 5.3.10), $d = 1$ and (by definition of $P(L)$) $x \xrightarrow{1} z \in P(L)$, which proves the claim.

Assume, therefore, that $d_1 = 0$. If $d = 0$ then $x \xrightarrow{0} z \in P(L)$ and the claim holds. Assume, therefore, that $d = 1$. Let $w_1 \xrightarrow{1} w_2 \in L$ be the first link of depth 1 on the L-path from x to z (by linear transitivity such a link exists). That $d_1 = 0$ implies (by linear transitivity) that $d_x = 0$ and therefore $w_1 \neq x$. Therefore, $x \xrightarrow{0} w_1 \in P(L)$. By linear transitivity, because $d = 1$, this means that there is a link $w_1 \xrightarrow{0} x \in P(L)$. There remain two cases: $w_1 \in (x, y)$ and $w_1 \in [y, z)$.

- *Case 1:* $w_1 \in (x, y)$. Let $v_1 \xrightarrow{d'} v_2 \in L$ be the link in the L-path from x to y such that $w_1 \in (v_1, v_2]$. If $w_1 \in (v_1, v_2)$ then, by connectedness $v_1 \xrightarrow{L} w_1$ and $w_1 \xrightarrow{L} v_2$, contradicting minimality. Therefore, w_1 is on the L-path from x to y. By linear transitivity (using $x \overset{0}{\rightleftarrows} w_1 \in P(L)$), $w_1 \xrightarrow{d_1} y \in P(L)$. Since the L-path from w_1 to y is shorter than that from x to y, it follows from the induction hypothesis applied to the links $w_1 \xrightarrow{d_1} y$ and $z \xrightarrow{d_2} y$, that either $w_1 \xrightarrow{d_1} z \in P(L)$ or $z \xrightarrow{d_2} w_1 \in P(L)$. Since $x \overset{0}{\rightleftarrows} w_1 \in P(L)$, this shows that either $x \xrightarrow{d_1} z \in P(L)$ or $z \xrightarrow{d_2} x \in P(L)$, as required.

- *Case 2:* $w_1 \in [y, z)$. Since $w_1 \xrightarrow{1} w_2 \in L$ and $w_2 \xrightarrow{L} z$, it follows that $w_1 \xrightarrow{1} z \in P(L)$. By connectedness (from $z \xrightarrow{d_2} y \in P(L)$) and the equality of depth of opposite links, $z \xrightarrow{1} w_1 \in P(L)$ and, by monotonicity, $z \xrightarrow{1} y \in P(L)$ (that is, $d_2 = 1$). By linear transitivity, since $w_1 \xrightarrow{0} x \in P(L)$, $z \xrightarrow{1} x \in P^*(L)$. Since $z \xrightarrow{1} y \in P(L)$, also $z \xrightarrow{1} x \in P(L)$, which proves the claim.

□

Constructing a Bracketing From $P(L)$

The next lemma shows that if L satisfies the characterizing properties of Definition 5.3.1 then applying the simple reconstruction algorithm (Algorithm 5.1.29) to $P(L)$ results in a bracketing of the utterance. Subsequent lemmas will show that L is a shortest common cover link set for this bracketing.

5.3.19. Lemma (bracketing from $P(L)$). *Let L be a 0,1-common cover link set over an utterance U. Assume that L satisfies the characterizing properties of Definition 5.3.1. If $\mathcal{A}$ is the result of applying the simple reconstruction algorithm (Algorithm 5.1.29) to $P(L)$ then $\mathcal{A}$ is a bracketing.*

Proof. Assume the assumptions of the lemma hold. It is obvious from the definition of the simple reconstruction algorithm that every word is covered by at least one bracket. It therefore remains to show that no brackets cross. Let $B_1, B_2 \in \mathcal{A}$ be such that $B_1 \cap B_2 \neq \emptyset$. By the definition of the algorithm, there are $x, y \in U$ such that B_1 was created from links based at x and B_2 was created from links based at y. By monotonicity and connectedness of $P(L)$ (Lemma 5.3.10 and Lemma 5.3.11), for every $z \in B_1 \setminus \{x\}$ there is a link $x \xrightarrow{d} z \in P(L)$ (for some d) and for every $z \in B_2 \setminus \{y\}$ there is a link $x \xrightarrow{d} z \in P(L)$ (for some d). If $x = y$, it follows directly from Lemma 5.3.10 that B_1 and B_2 do not cross.

To prove the lemma it is sufficient to show that if there exists $z \in B_1 \setminus B_2$ then $u \in B_2$ implies $u \in B_1$. Assume, therefore, that $z \in B_1 \setminus B_2$. Without loss of generality we can assume that $z \notin (x, y]$ (because if $z \in (x, y]$ then also $x \in B_1 \setminus B_2$ and we can take $z = x$). Let $u \in B_2$. It is enough to prove the claim for all $u \notin [x, y)$ because $y \in B_2$ and therefore by proving the claim for all $u \notin [x, y)$ we also prove that $y \in B_1$, which implies that, for any $u \in [x, y)$, $u \in B_1$.

Let $u \in B_2$ such that $u \notin [x, y)$. If $u \in [z, x]$ then $u \in B_1$, as required. Assume, therefore, that $u \notin [z, x]$. If $z \in [y, u]$ then $z \in B_2$, contrary to the assumption. Because $u \notin [x, y)$ and $z \notin (x, y]$, u and z are outside (x, y) and, because $z \notin [y, u]$ and $u \notin [z, x]$, u and z cannot be on the same side of (x, y), implying that $x, y \in [z, u]$. There are two case:

- *Case 1:* $x \in [z, y)$. There are three cases here, $y \xrightarrow{0} x \in P(L)$, $y \xrightarrow{1} x \in P(L)$ and no link from y to x in $P(L)$.

  If $y \xrightarrow{0} x \in P(L)$ then, because $y \xrightarrow{0} z \notin P(L)$ (otherwise $z \in B_2$), we have that $z \neq x$ and (by linear transitivity for $P^*(L)$, Lemma 5.3.16) $x \xrightarrow{1} z \in P(L)$ and $x \xrightarrow{0} y \in P(L)$. If $u = y$, we are done and otherwise, by assumption, $y \xrightarrow{d'} u \in P(L)$. By linear transitivity for $P^*(L)$, it follows that $x \xrightarrow{d'} u \in P^*(L)$ and since $z \in B_1$ and $x \xrightarrow{1} z \in P(L)$ we have that $u \in B_1$.

If $y \xrightarrow{1} x \in P(L)$ then (by linear transitivity for $P^*(L)$) $y \xrightarrow{1} z \in P(L)$. Since $z \notin B_2$ it follows that B_2 is constructed only from links of depth 0 based at y. This means that if $v \in B_1 \cap B_2 \cap [x, y]$ then $v \neq x$ and either $v = y$ or $y \xrightarrow{0} v \in P(L)$. Since B_1 and B_2 intersect, such a v exists. If $v = y$ then $x \xrightarrow{d'} y \in P(L)$ and, by equality of opposite links (Lemma 5.3.15), $d' = 1$. If $v \in (y, x)$ then, by resolution (Lemma 5.3.18), $x \xrightarrow{1} v \in P(L)$ and $x \xrightarrow{1} y \in P(L)$. By linear transitivity $x \xrightarrow{1} u \in P(L)$ and since $v \in B_1$ this implies that $u \in B_1$, as required.

Finally, assume that there is no link from y to x. Because the brackets B_1 and B_2 intersect, there is $v \in [x, y]$ such that $v \in B_1 \cap B_2$. If $v = x$ then $y \xrightarrow{d} x \in P(L)$, which contradicts the assumptions. Therefore, $v \in (x, y]$. If $v = y$ then $x \xrightarrow{d} y \in P(L)$ directly from $v \in B_1$. If $v \in (x, y)$ then $x \xrightarrow{d_1} v \in P(L)$ and $y \xrightarrow{d_2} v \in P(L)$ and by resolution for $P(L)$ (Lemma 5.3.18), either $x \xrightarrow{d_1} y \in P(L)$ or $y \xrightarrow{d_2} x \in P(L)$. By the assumption, $x \xrightarrow{d_1} y \in P(L)$. Since $v \in B_1$, also $y \in B_1$. Moreover, since there is no link from y to x, it follows from linear transitivity that $x \xrightarrow{d_1} u \in P(L)$. This shows that $u \in B_1$ and completes the proof for this case.

- *Case 2:* $x \in [u, y)$. Because $x \in [u, y)$, $x \neq z$ and $y \neq u$. Therefore, there are links $x \xrightarrow{d_1} z \in P(L)$ and $y \xrightarrow{d_2} u \in P(L)$. By connectedness (Lemma 5.3.11) and equality of opposite links (Lemma 5.3.15), there are links $x \xrightarrow{d} y \in P(L)$ and $y \xrightarrow{d} x \in P(L)$ (for $d \leq d_1, d_2$). If $d_1 = 0$ then by crossing transitivity (Lemma 5.3.17) there is a link $y \xrightarrow{0} z \in P(L)$, which contradicts the assumption that $z \notin B_2$. Therefore, $d_1 = 1$ ($x \xrightarrow{1} z \in P(L)$). By the definition of $P(L)$, this means that there is a word x' such that $x \xrightarrow{1} x' \in L$. Crossing transitivity and the definition of $P(L)$ then imply that $x \xrightarrow{d'} u \in P(L)$. Since $z \in B_1$ and $d' \leq d_1$ we have that $u \in B_1$, as required.

□

The set of links $P(L)$ is a representative subset of $R_{\mathcal{A}}$ for the bracketing $\mathcal{A}$ created from $P(L)$ by applying the simple reconstruction algorithm. To show this, we need the following lemma.

5.3.20. LEMMA. *Let L be a 0,1-common cover link set over an utterance U. Assume that L satisfies the characterizing properties of Definition 5.3.1. Let $\mathcal{A}$ be the result of applying the simple reconstruction algorithm (Algorithm 5.1.29) to $P(L)$. If, for $x \in U$, $A_d(x)$ is the bracket generated by the algorithm from* *links based at x of depth at most d then x is of minimal depth (in $\mathcal{A}$) under these brackets and $A_0(x) = B_0^{\mathcal{A}}(x)$. If $A_0(x) \neq A_1(x)$ then $A_1(x) = B_1^{\mathcal{A}}(x)$.*

Proof. Assume the assumptions of the lemma hold. By definition and the connectedness of $P(L)$, $A_0(x) = \{x\} \cup \{y : x \xrightarrow{0} y \in P(L)\}$. Since $\mathcal{A}$ is a bracketing (Lemma 5.3.19), to show that $A_0(x) = B_0^{\mathcal{A}}(x)$ it is sufficient to show that if there is a bracket B such that $x \in B \subseteq A_0(x)$ then $B = A_0(x)$. Assume that B is such a bracket. There are some y and $d \leq 1$ such that $B = A_d(y)$. Because $A_0(x) \subseteq A_1(x)$, we can assume that $y \neq x$. Because $y \in A_0(x)$, $x \xrightarrow{0} y \in P(L)$ and because $x \in A_d(y)$ there exists d' such that $y \xrightarrow{d'} x \in P(L)$. By equality of opposite links in $P(L)$ (Lemma 5.3.15), $d' = 0$ ($y \xrightarrow{0} x \in P(L)$). Let $z \in A_0(x)$ ($z \neq x$). There is a link $x \xrightarrow{0} z \in P(L)$ and by either linear transitivity (Lemma 5.3.16) or crossing transitivity (Lemma 5.3.17) it follows that $y \xrightarrow{0} z \in P(L)$. Therefore, $A_0(x) \subseteq A_d(y)$, which shows that $B = A_0(x)$, as required. Clearly, x is of minimal depth (0) under $A_0(x)$.

Now assume that $A_1(x) \neq A_0(x)$. By definition, $A_0(x) \subset A_1(x)$. Because $\mathcal{A}$ is a bracketing, to show that $A_1(x) = B_1^{\mathcal{A}}(x)$, it is sufficient to show that if there is a bracket $A_0(x) \subset B \subseteq A_1(x)$ then $B = A_1(x)$. Assume that B is such a bracket. There is a y such that $B = A_d(y)$. As before, there exists $d' \leq 1$ such that $x \xrightarrow{d'} y \in P(L)$ and $y \xrightarrow{d'} x \in P(L)$. If $d = 0$ then, by the first part of the lemma, $y \notin A_0(x)$ and therefore $d' = 1$. At the same time, $x \in A_0(y)$ and therefore $d' = 0$. This contradiction shows that $d = 1$ and $d' = 1$. By the definition of $P(L)$ this means that there exists w such that $y \xrightarrow{1} w \in L$. Let $u \in A_1(x)$, $u \neq x$. There is a link $x \xrightarrow{d_1} u \in P(L)$. Since $y \xrightarrow{1} x \in P(L)$, by either linear transitivity or crossing transitivity (and using $y \xrightarrow{1} w \in L$), it follows that there is $d_2 \leq 1$ such that $y \xrightarrow{d_2} u \in P(L)$. Therefore, $u \in A_1(y)$, showing that $A_1(x) = B$, as required. The proof also showed that $B = A_1(y)$ which proves that x is of minimal depth under $A_1(x)$. □

Characterization Is Sufficient

It is now possible to show that the properties given in the characterization (Definition 5.3.1) are sufficient to ensure that a set of links is a shortest common cover link set for some bracketing. The following lemma shows that a set of links L which satisfies the characterization properties is a shortest common cover link set for the bracketing produced from $P(L)$ by the simple bracket reconstruction algorithm.

5.3.21. Lemma. *Let L be a 0,1-common cover link set over an utterance U. If L satisfies the characterizing properties of Definition 5.3.1 then there exists a* *bracketing $\mathcal{C}$ such that L is a shortest common cover set for $\mathcal{C}$. Moreover, $P(L)$ is the smallest representative subset of $R_{\mathcal{C}}$ such that $L \subseteq P(L)$ and the application of the simple reconstruction algorithm (Algorithm 5.1.29) to $P(L)$ outputs $\mathcal{C}$.*

Proof. Assume the assumptions of the lemma hold. Let $\mathcal{C}$ be the result of applying the simple reconstruction algorithm (Algorithm 5.1.29) to $P(L)$. By Lemma 5.3.19, $\mathcal{C}$ is a bracketing. By lemma Lemma 5.3.20, every bracket in $\mathcal{C}$ is of height 0 or 1. By the same lemma (and using its notation) if $B \in \mathcal{C}$ is of height 0 then for every $x \in B$ of depth 0 under B, $B = A_0(x)$. If $B \in \mathcal{C}$ is of height 1 then $B = A_1(x)$ for some $x \in B$. This shows that $P(L)$ is a representative subset of $R_{\mathcal{C}}$.

Let S be the shortest common cover link set induced by $P(L)$. I will show that $S = L$. If $x \xrightarrow{d} y \in S$ then $x \xrightarrow{d} y \in P(L)$ and, by definition, there is no $z \in (x, y)$ such that $x \xrightarrow{d_1} z \in P(L)$ and $z \xrightarrow{d_2} y \in P(L)$. If the L-path from x to y consists of more than one link in L then there does exists such a z (if there is a link $w_1 \xrightarrow{1} w_2 \in L$ on the path from x to y take z to be the first w_1 for which this holds and otherwise take an arbitrary z on the path). This shows that $x \xrightarrow{d} y \in L$. Therefore $S \subseteq L$.

Now let $x \xrightarrow{d} y \in L$. If $x \xrightarrow{d} y \notin S$ then (because $x \xrightarrow{d} y \in P(L)$) there is $z \in (x, y)$ such that $x \xrightarrow{d_1} z \in P(L)$ and $z \xrightarrow{d_2} y \in P(L)$. It then follows that $x \xrightarrow{L} z$ and $z \xrightarrow{L} y$, which contradicts the minimality of L. This shows that $x \xrightarrow{d} y \in S$ and therefore, together with what has been shown above, that $L = S$.

It remains to show that $P(L)$ is the smallest representative subset of $R_{\mathcal{C}}$ such that $L \subseteq P(L)$. By the definition of $P(L)$, $x \xrightarrow{1} y \in P(L)$ implies that there is some z such that $x \xrightarrow{1} z \in L$. By Lemma 5.1.21 this is exactly the condition for the inclusion of the link $x \xrightarrow{d} y$ in the smallest representative subset of $R_{\mathcal{C}}$ which contains L. This shows that $P(L)$ is indeed equal to this set. $\square$

5.4 Incremental Reconstruction

Given a 0,1-shortest common cover link set S, there is an efficient incremental algorithm which reconstructs the bracketing represented by S.

5.4.1 (2.6.5). Algorithm. [incremental reconstruction from S] Given is a set of links S of depth 0 or 1 over an utterance $\langle x_1, \ldots, x_n \rangle$. Let $S_k = \{x \xrightarrow{d} y \in S : x, y \in [x_1, x_k]\}$ be the restriction of S to $[x_1, x_k]$. The algorithm updates a bracketing $\mathcal{B}$.

- Initialize $\mathcal{B} = \{\langle x_1 \rangle\}$.

- For each $k = 2, \dots, n$ perform the following modifications of $\mathcal{B}$, in the given order:

  1. For every link $x_i \xrightarrow{0} x_k \in S_k$, extend all brackets in $\mathcal{B}$ which cover x_i to cover x_k.
  2. For every link $x_i \xrightarrow{1} x_k \in S_k$:
     (a) Extend all brackets which cover $B_0^{\mathcal{B}}(x_i)$ to cover x_k.
     (b) If there is no $x \xrightarrow{d} y \in S_{k-1}$ such that $x \in B_0^{\mathcal{B}}(x_i)$ and $y \notin B_0^{\mathcal{B}}(x_i)$, add a bracket which covers x_k and $B_0^{\mathcal{B}}(x_i)$.
  3. If there is no x_i such that $x_k \xrightarrow{0} x_i \in S_k$ and $x_k \in B_0^{\mathcal{B}}(x_i)$ then add to $\mathcal{B}$ the smallest bracket which covers x_k and every x such that $x_k \xrightarrow{0,S_k} x$.
  4. If there is $x_k \xrightarrow{1} x_i \in S_k$ then add to $\mathcal{B}$ (if it is not already in $\mathcal{B}$) the smallest bracket which covers x_k and every x such that $x_k \xrightarrow{S_k} x$.

- Output $\mathcal{B}$.

The rest of this section show that this incremental reconstruction algorithm correctly reconstructs any bracketing $\mathcal{C}$ from any of its shortest common cover link sets S (as long as $R_{\mathcal{C}}$ contains only links of depth 0 or 1). To do so, I use the properties of the set of links $P(S)$ as described in the previous section. It is assumed throughout this section that the depth of all links is either 0 or 1.

Let $\mathcal{C}$ be a bracketing over an utterance $U = \langle x_1, \dots, x_n \rangle$ and let S be a shortest common cover link set for $\mathcal{C}$. I refer to the steps carried out by the incremental reconstruction algorithm for a given value of k as *loop* k of the algorithm. I write $\mathcal{I}_k(S)$ for the set of brackets constructed by the incremental algorithm by the end of loop k, where $\mathcal{I}_1(S)$ is defined to be the initial bracketing $\{\langle x_1 \rangle\}$. I also write $S_k = \{x \xrightarrow{d} y \in S \ : \ x, y \in [x_1, x_k]\}$ for the restriction of the set S to $[x_1, x_k]$ and $\mathcal{A}(P(S_k))$ for the set of brackets produced from $P(S_k)$ on $[x_1, x_k]$ by Algorithm 5.1.29. By Lemma 5.3.21, to prove the correctness of the incremental reconstruction algorithm, it is enough to show that $\mathcal{A}(P(S_n)) = \mathcal{I}_n(S)$. I will show this by showing by induction that $\mathcal{A}(P(S_k)) = \mathcal{I}_k(S)$ for every k.

Since S is a shortest common cover link set, it satisfies all characterizing conditions of Definition 5.3.1. By Lemma 5.3.4, any restriction S_k of S must also satisfy these characterizing conditions. Therefore, S_k is a shortest common cover link set for a bracketing on $[x_1, x_k]$. As shown in Lemma 5.3.21, this bracketing is $\mathcal{A}(P(S_k))$ and $P(S_k)$ is the minimal representative subset of $R_{\mathcal{A}(P(S_k))}$ which contains S_k. This also means that any of the properties of $P(S)$ proved in the previous section hold for $P(S_k)$.

5.4.1 Generators

Brackets in $\mathcal{A}(P(S_k))$ are most easily studied by referring to the word for which they were constructed by the reconstruction algorithm. Such a word will be referred to as a *generator* of the bracket. A single bracket may have more than one generator and we are interested in the first such generator to appear in the utterance. Formally, this is defined as follows:

5.4.2. Definition. [bracket generator] Let $\mathcal{C}$ be a bracketing over an utterance $U = \langle x_1, \ldots, x_n \rangle$ such that all links in $R_\mathcal{C}$ are of depth 0 or 1 and let S be a shortest common cover link set for $\mathcal{C}$.

1. For $x \in [x_1, x_k]$, $G_d^k(x) = \{x\} \cup \{y \ : \ \exists d' \leq d \ s.t. \ x \xrightarrow{d'} y \in P(S_k)\}$ is the bracket in $\mathcal{A}(P(S_k))$ generated by x and depth d.
2. A word $x \in [x_1, x_k]$ is a *generator* of B in $\mathcal{A}(P(S_k))$ iff there exists $d \leq 1$ such that $B = G_d^k(x)$.
3. A generator x of B (in $\mathcal{A}(P(S_k))$) is the *leftmost generator* of B (in $\mathcal{A}(P(S_k))$) if for every other generator x' of B (in $\mathcal{A}(P(S_k))$), $x' \in (x, x_k]$.

The following lemma describes conditions which imply that two words are generators of the same bracket.

5.4.3. Lemma. *Let $\mathcal{C}$ be a bracketing over an utterance $U = \langle x_1, \ldots, x_n \rangle$ such that all links in $R_\mathcal{C}$ are of depth 0 or 1 and let S be a shortest common cover link set for $\mathcal{C}$.*

1. *If $x, y \in [x_1, x_k]$ are such that $x \xrightarrow{0} y \in P(S_k)$ and $y \xrightarrow{0} x \in P(S_k)$ then:*
   (a) *$G_0^k(x) = G_0^k(y)$.*
   (b) *If there are u, w such that $x \xrightarrow{1} u \in S_k$ and $y \xrightarrow{1} w \in S_k$ then for $d \leq 1$, $G_d^k(x) = G_d^k(y)$.*
2. *If $x, y \in [x_1, x_k]$ are such that $x \xrightarrow{1} y \in P(S_k)$ and $y \xrightarrow{1} x \in P(S_k)$ then $G_1^k(x) = G_1^k(y)$.*

Proof. Assume the assumptions of the lemma hold.

1. Assume that $x, y \in [x_1, x_k]$ are such that $x \xrightarrow{0} y \in P(S_k)$ and $y \xrightarrow{0} x \in P(S_k)$.
   (a) Since $x \xrightarrow{0} y \in P(S_k)$, $y \in G_0^k(x)$ and since $y \xrightarrow{0} x \in P(S_k)$, $x \in G_0^k(y)$. If $z \neq x, y$ then by crossing transitivity (Lemma 5.3.17) and linear transitivity (Lemma 5.3.16), $x \xrightarrow{0} z \in P(S_k) \iff y \xrightarrow{0} z \in P(S_k)$. Therefore, $G_0^k(x) = G_0^k(y)$.

(b) Assume that there are u, w such that $x \xrightarrow{1} u \in S_k$ and $y \xrightarrow{1} w \in S_k$. That $G_0^k(x) = G_0^k(y)$ has been shown already. By crossing and linear transitivity and by the existence of the links $x \xrightarrow{1} u \in S_k$ and $y \xrightarrow{1} w \in S_k$, it follows that, for $z \neq x, y$, $x \xrightarrow{1} z \in P(S_k) \iff y \xrightarrow{1} z \in P(S_k)$. Therefore, $G_1^k(x) = G_1^k(y)$.

2. Assume that $x, y \in [x_1, x_k]$ are such that $x \xrightarrow{1} y \in P(S_k)$ and $y \xrightarrow{1} x \in P(S_k)$. Since $x \xrightarrow{1} y \in P(S_k)$, $y \in G_1^k(x)$ and, similarly, $x \in G_1^k(y)$. Because $x \xrightarrow{1} y \in P(S_k)$, by definition there is u such that $x \xrightarrow{1} u \in S_k$. Similarly, there is w such that $y \xrightarrow{1} w \in S_k$. It therefore follows from crossing and linear transitivity that, for $z \neq x, y$ and $d \leq 1$, $x \xrightarrow{d} z \in P(S_k)$ iff there is $d' \leq 1$ such that $y \xrightarrow{d'} z \in P(S_k)$. This shows that $G_1^k(x) = G_1^k(y)$.

□

5.4.4. Lemma (bracket generator). *Let $\mathcal{C}$ be a bracketing over an utterance $U = \langle x_1, \ldots, x_n \rangle$ such that all links in $R_{\mathcal{C}}$ are of depth 0 or 1 and let S be a shortest common cover link set for $\mathcal{C}$.*

1. *If $B \in \mathcal{A}(P(S_k))$ then there exist $x \in B$ and $d \leq 1$ such that $B = G_d^k(x)$.*

2. *If x is the leftmost generator of $G_d^k(x)$ and there are $z \in G_d^k(x)$ and d' such that $z \xrightarrow{d'} x_{k+1} \in S_{k+1}$ then $z \in [x, x_k]$.*

3. *If there exists $z \in G_d^k(x) \backslash G_d^{k-1}(x)$ such that $z \neq x_k$ then $d = 1$, $x \xrightarrow{1} x_k \in S_k$ and there is $u \in [x_1, x)$ such that $x \xrightarrow{0} u \in P(S_{k-1})$, $u \xrightarrow{0} x \in P(S_{k-1})$ and $u \xrightarrow{1} v \in S_{k-1}$ for some v.*

4. *If $x \neq x_k$ is the leftmost generator of $G_d^k(x)$ then either $G_d^k(x) = G_d^{k-1}(x)$ or $G_d^k(x) = G_d^{k-1}(x) \cup \{x_k\}$.*

5. *If x is the leftmost generator of $G_d^k(x)$ then either $G_d^{k+1}(x) = G_d^k(x)$ or $G_d^{k+1}(x) = G_d^k(x) \cup \{x_{k+1}\}$.*

6. *If $x \neq x_{k+1}$ then x is the leftmost generator of $G_d^k(x)$ iff x is the leftmost generator of $G_d^{k+1}(x)$.*

7. *If, for $B \in \mathcal{A}(P(S_k))$, there is a link $x \xrightarrow{d} y \in S_k$ such that $x \in B$ and $y \notin B$, then $d = 1$ and $B = G_0^k(x)$. Moreover, if z is a generator of B then x is a generator of $G_d^k(z)$ for $d \leq 1$.*

Proof. Assume the assumptions of the lemma hold.

1. That there exists a generator x of B is immediate from the construction of $\mathcal{A}(P(S_k))$ by the simple reconstruction algorithm (Algorithm 5.1.29).

2. Assume that x is the leftmost generator of $G_d^k(x)$ and that there are $z \in G_d^k(x)$ and d' such that $z \xrightarrow{d'} x_{k+1} \in S_{k+1}$. If $z = x$ then $z \in [x, x_k]$, so assume from now that $x \neq z$. Since $z \in G_d^k(x)$, there is a link $x \xrightarrow{d''} z \in P(S_k)$. If $z \xrightarrow{0} x_{k+1} \in S_{k+1}$ then it follows immediately from blocking that $z \in (x, x_k]$.

   Assume, therefore, that $z \xrightarrow{1} x_{k+1} \in S_{k+1}$ and assume that $z \notin [x, x_k]$. I will show that this leads to a contradiction. It follows from $z \in G_d^k(x)$ and blocking that $x \xrightarrow{0} z \in P(S_k)$. From $z \xrightarrow{1} x_{k+1} \in S_{k+1}$ it follows from connectedness for $P(L)$ (Lemma 5.3.11) that $z \xrightarrow{d'''} x \in P(S_{k+1})$. Since $x \xrightarrow{0} z \in P(S_k)$, $x \xrightarrow{0} z \in P(S_{k+1})$. By the equal depth of opposite links, $z \xrightarrow{0} x \in P(S_{k+1})$ and since $x, z \in [x_1, x_k]$, $z \xrightarrow{0} x \in P(S_k)$. If $d = 0$ then, by Lemma 5.4.3, $G_d^k(x) = G_d^k(z)$, in contradiction to the assumption that x is the leftmost generator $G_d^k(x)$. Assume, therefore, that $d = 1$ and $G_1^k(x) \neq G_0^k(x)$. It follows that there is a link $x \xrightarrow{1} w \in P(S_k)$ and therefore there is a link $x \xrightarrow{1} u \in S_k$. By crossing transitivity this means that there is a link $x \xrightarrow{d'''} x_{k+1} \in P(S_{k+1})$ which together with $z \xrightarrow{0} x \in P(S_{k+1})$ and $z \xrightarrow{1} x_{k+1} \in S_{k+1}$ contradicts the minimality property of S_{k+1}.

3. Assume that $z \in G_d^k(x) \setminus G_d^{k-1}(x)$ and $z \neq x_k$. There is a link $x \xrightarrow{d'} z \in P(S_k) \setminus P(S_{k-1})$ such that $d' \leq d$. Since $z \neq x_k$, the S_k-path from x to z is also an S_{k-1}-path and therefore $x \xrightarrow{d'} z \in P^*(S_{k-1})$. By definition, since $x \xrightarrow{d'} z \notin P(S_{k-1})$, the only possibility is that $d' = 1$ (and therefore $d = 1$) and there is no link $x \xrightarrow{1} w \in S_{k-1}$ but there is a link $x \xrightarrow{1} x_k \in S_k$. Since $x \xrightarrow{1} z \in P(S_k)$ and there is no $w \in [x_1, x_{k-1}]$ such that $x \xrightarrow{1} w \in S_{k-1}$, it follows that there are two consecutive words u and v on the S_{k-1}-path from x to z such that $u \xrightarrow{1} v \in S_{k-1}$, $x \xrightarrow{0} u \in P(S_{k-1})$ and $u \xrightarrow{0} x \in P(S_{k-1})$. If $u \in (x, x_k)$ then it follows from transitivity that $u \xrightarrow{d''} x_k \in P(S_k)$ which together with $x \xrightarrow{0} u \in P(S_{k-1})$ and $x \xrightarrow{1} x_k \in S_k$ contradicts the minimality property of S_k. Therefore, $u \in [x_1, x)$, which completes the proof.

4. Assume that $x \neq x_k$ is the leftmost generator of $G_d^k(x)$. Assume, by contradiction, that $z \in G_d^k(x) \setminus G_d^{k-1}(x)$ and $z \neq x_k$. By the previous part of the lemma, $d = 1$, $x \xrightarrow{1} x_k \in S_k$ and there is $u \in [x_1, x)$ such that $x \xrightarrow{0} u \in P(S_{k-1})$, $u \xrightarrow{0} x \in P(S_{k-1})$ and $u \xrightarrow{1} v \in S_{k-1}$. By Lemma 5.4.3, u is a generator of $G_d^k(x)$ in contradiction to the assumption that x is the leftmost generator of $G_d^k(x)$.

5. Assume that x is the leftmost generator of $G^k_d(x)$. Assume, by contradiction, that $z \neq x_{k+1}$ and $z \in G^{k+1}_d(x) \setminus G^k_d(x)$. As above, this implies that $d = 1$, $x \xrightarrow{1} x_{k+1} \in S_{k+1}$ and there is $u \in [x_1, x)$ such that $x \xrightarrow{0} u \in P(S_k)$, $u \xrightarrow{0} x \in P(S_k)$ and $u \xrightarrow{1} v \in S_k$. If $G^k_d(x) = G^k_0(x)$ then, by Lemma 5.4.3, u is a generator of $G^k_d(x)$. If $G^k_d(x) \neq G^k_0(x)$ then there is $w \in [x_1, x_k]$ such that $x \xrightarrow{1} w \in S_k$. It then again follows from Lemma 5.4.3 that u is a generator of $G^k_d(x)$. In either case u is a generator of $G^k_d(x)$, in contradiction to the assumption that x is the leftmost generator of $G^k_d(x)$.

6. Assume x is the leftmost generator of $G^k_d(x)$ and y is the leftmost generator of $G^{k+1}_d(x)$ (that is, $G^{k+1}_{d'}(y) = G^{k+1}_d(x)$). By definition, $y \in [x_1, x]$. Specifically, $y \neq x_{k+1}$ and, therefore, by a previous part of the lemma, $G^k_{d'}(y) = G^{k+1}_{d'}(y) \setminus \{x_{k+1}\}$. Similarly, $G^k_d(x) = G^{k+1}_d(x) \setminus \{x_{k+1}\}$. Since $G^{k+1}_{d'}(y) = G^{k+1}_d(x)$, this shows that $G^k_{d'}(y) = G^k_d(x)$ and since x is the leftmost generator of this bracket and $y \in [x_1, x]$ it follows that $x = y$, as required.

7. Assume that $B \in \mathcal{A}(P(S_k))$ and there is $x \xrightarrow{d} y \in S_k$ such that $x \in B$ and $y \notin B$. Let z be a generator of B and let d_B be such that $B = G^k_{d_B}(z)$. If $z = x$ then the claim is obvious. Assume, therefore, that $x \neq z$. Because $x \in B$, there is a link $z \xrightarrow{d'} x \in P(S_k)$ for $d' \leq d_B$.

   If $z \in (x, y)$ then, by blocking, $d = 1$ and $z \xrightarrow{0} x \in P(S_k)$. By Lemma 5.1.17, also $x \xrightarrow{0} z \in P(S_k)$. If $x \in (z, y)$ then $y \notin B$ implies (by linear transitivity) that $x \xrightarrow{0} z \in P(S_k)$, $z \xrightarrow{0} x \in P(S_k)$ and $d = 1$. In either case, this means that $G^k_0(z) = G^k_0(x)$. If $G^k_1(z) \neq G^k_0(z)$ then there is a link $z \xrightarrow{1} w \in S_k$ and, by Lemma 5.4.3, $G^k_1(z) = G^k_1(x)$. Moreover, by crossing or linear transitivity, $y \in G^k_1(z)$. Since $y \notin B$, $B = G^k_0(z) = G^k_0(x)$.

□

5.4.2 Correctness of Incremental Reconstruction

Because several different cases need to be considered in the proof that $\mathcal{A}(P(S_k)) = \mathcal{I}_k(S)$, I split the proof into several lemmas.

5.4.5. Lemma. *Let $\mathcal{C}$ be a bracketing over an utterance $U = \langle x_1, \ldots, x_n \rangle$ such that all links in $R_{\mathcal{C}}$ are of depth 0 or 1 and let S be a shortest common cover link set for $\mathcal{C}$. If, for some $1 < k \leq n$, $\mathcal{A}(P(S_{k-1})) = \mathcal{I}_{k-1}(S)$ and $B \in \mathcal{A}(P(S_k))$ such that $B = G^k_{d_B}(x)$ for $x \neq x_k$ then $B \in \mathcal{I}_k(S)$.*

PROOF. Assume that the assumptions of the lemma hold, $\mathcal{A}(P(S_{k-1})) = \mathcal{I}_{k-1}(S)$ and $B \in \mathcal{A}(P(S_k))$ is as in the lemma. Let x be the leftmost generator of $B = G^k_{d_B}(x)$. By the assumption, $x \neq x_k$. Let $B' = G^{k-1}_{d_B}(x)$. By definition, $B' \in \mathcal{A}(P(S_{k-1})) = \mathcal{I}_{k-1}(S)$. By Lemma 5.4.4, either $B = B'$ or $B = B' \cup \{x_k\}$:

- *Case 1:* $B = B'$. If there are no $z \in B'$ and d_{zx_k} such that $z \overset{d_{zx_k}}{\rightarrow} x_k \in S_k$ then B' is not extended by loop k and, since $B' \in \mathcal{I}_{k-1}(S)$, it follows that $B = B' \in \mathcal{I}_k(S)$, as required.

  Assume, therefore, that there do exist $z \in B'$ and d_{zx_k} such that $z \overset{d_{zx_k}}{\rightarrow} x_k \in S_k$. By directional uniqueness (Lemma 5.1.25) there is only one such z in B'. By definition, either $z = x$ or $x \overset{d'}{\rightarrow} z \in P(S_{k-1})$ for some $d' \leq d_B$.

  First, I show that $B' = G^{k-1}_0(x)$. If this does not hold then $d_B = 1$ and there is $u \in B'$ such that $x \overset{1}{\rightarrow} u \in P(S_{k-1})$. By transitivity (Lemma 5.3.17) for $x \in (z, x_k)$, by linear transitivity (Lemma 5.3.16) for $z \in (x, x_k)$ and trivially for $x = z$ this implies that $x \overset{d''}{\rightarrow} x_k \in P(S_k)$ for $d'' \leq d_B$ and therefore $x_k \in B$, contrary to the assumption.

  Next, I show that $d_{zx_k} = 1$ and either $x = z$ or $x \overset{0}{\rightarrow} z \in P(S_{k-1})$ and $z \overset{0}{\rightarrow} x \in P(S_{k-1})$:

  1. Assume $x = z$. If $d_{zx_k} = 0$ then $x_k \in B$, contrary to the assumption.
  2. Assume $x \in (z, x_k)$. By blocking, $d_{zx_k} = 1$ and $x \overset{0}{\rightarrow} z \in P(S_{k-1})$. By connectedness and the equality of depth of opposite links (Lemma 5.1.14), $z \overset{0}{\rightarrow} x \in P(S_{k-1})$.
  3. Assume $z \in (x, x_k)$. Since $z \in B' = G^{k-1}_0(x)$, $x \overset{0}{\rightarrow} z \in P(S_{k-1})$. By linear transitivity, if either $d_{zx_k} = 0$ or there is no link $z \overset{0}{\rightarrow} x \in P(S_{k-1})$ then $x \overset{0}{\rightarrow} x_k \in P(S_k)$, which implies that $x_k \in B$, contrary to the assumption.

  By Lemma 5.4.3, since $B' = G^{k-1}_0(x)$, and either $x = z$ or $x \overset{0}{\rightarrow} z \in P(S_{k-1})$ and $z \overset{0}{\rightarrow} x \in P(S_{k-1})$, $B' = G^{k-1}_0(z)$, which, by Lemma 5.3.20, is the smallest bracket covering z. Together with $d_{zx_k} = 1$, this implies that B' is not extended by loop k of the incremental algorithm and therefore $B' \in \mathcal{I}_k(S)$, as required.

- *Case 2:* $B = B' \cup \{x_k\}$. Since $x_k \in B$, there is a link $x \overset{d_{xx_k}}{\rightarrow} x_k \in P(S_k)$ with $d_{xx_k} \leq d_B$. By connectedness, it follows that $[x, x_{k-1}] \subset B'$. By definition, there is an S_k-path from x to x_k. Let $z \overset{d_{zx_k}}{\rightarrow} x_k \in S_k$ be the last link in this path. Since $[x, x_{k-1}] \subset B'$, $z \in B'$. If $d_{zx_k} = 0$ then step 1 of loop k

extends B' to cover x_k and therefore $B \in \mathcal{I}_k(S)$. If $d_{zx_k} = 1$ then there are two possibilities:

1. There is no link $z \xrightarrow{0} x \in P(S_{k-1})$ or there is $u \in B'$ such that $x \xrightarrow{1} u \in P(S_{k-1})$. In either case, B' is not the minimal bracket covering z and therefore step 2a of the incremental algorithm extends B' to cover x_k. The resulting bracket is B and therefore $B \in \mathcal{I}_k(S)$.

2. $z \xrightarrow{0} x \in P(S_{k-1})$ and $B' = G_0^{k-1}(x)$. Since $z \in B'$, $x \xrightarrow{0} z \in P(S_{k-1})$. By linear transitivity, $d_{xx_k} = 1$ (that is, $x \xrightarrow{1} x_k \in P(S_k)$) and, since $x_k \in B$, $d_B = 1$. By the definition of $P(S_k)$, this implies that there is $u \in [x_1, x_k]$ such that $x \xrightarrow{1} u \in S_k$. Since $d_B = 1$, $B' = G_0^{k-1}(x) = G_1^{k-1}(x)$ and therefore $u = x_k$. Since $z \xrightarrow{1} x_k$, directional uniqueness implies that $x = z$.

   I next show that there is no link $u \xrightarrow{d} v \in S_{k-1}$ such that $u \in B'$ and $v \notin B'$. Assume, by contradiction, that such a link exists. Since $u \xrightarrow{d} v \in S_{k-1}$, $v \neq x_k$ and therefore $v \notin B$. But, since $u \in B'$, $x \xrightarrow{0} u \in P(S_{k-1})$ and, since $x \xrightarrow{1} x_k \in S_k$, this implies (by transitivity) that $x \xrightarrow{d'} v \in P(S_k)$ for some $d' \leq 1$. Since $d_B = 1$, $v \in B$, which is a contradiction.

   Since there is no link $u \xrightarrow{d} v \in S_{k-1}$ such that $u \in B'$ and $v \notin B'$, step 2b of the algorithm creates a bracket which covers x_k and the minimal bracket covering z (which is B'). This bracket is B and therefore $B \in \mathcal{I}_k(S)$.

□

5.4.6. LEMMA. *Let $\mathcal{C}$ be a bracketing over an utterance $U = \langle x_1, \ldots, x_n \rangle$ such that all links in $R_{\mathcal{C}}$ are of depth 0 or 1 and let S be a shortest common cover link set for $\mathcal{C}$. If, for some $1 < k \leq n$, $\mathcal{A}(P(S_{k-1})) = \mathcal{I}_{k-1}(S)$ and $B \in \mathcal{I}_k(S)$ such that $x_k \notin B$ then $B \in \mathcal{A}(P(S_k))$.*

PROOF. Assume the assumptions of the lemma hold, $\mathcal{A}(P(S_{k-1})) = \mathcal{I}_{k-1}(S)$ and B is as in the lemma. Since any bracket created by the incremental algorithm in loop k must cover x_k, it follows that $B \in \mathcal{I}_{k-1}(S) = \mathcal{A}(P(S_{k-1}))$ and B was not extended by the incremental algorithm in loop k. There are two possibilities:

1. There exist no $z \in B$ and d such that $z \xrightarrow{d} x_k \in S_k$. Since $B \in \mathcal{A}(P(S_{k-1}))$, $B = G_{d_B}^{k-1}(x)$ for some $x \in [x_1, x_{k-1}]$. To prove that $B \in \mathcal{A}(P(S_k))$ it is enough to show that there is no $d \leq d_B$ such that $x \xrightarrow{d} x_k \in P(S_k)$ because then $B = G_{d_B}^{k-1}(x) = G_{d_B}^{k}(x) \in \mathcal{A}(P(S_k))$.

Assume, by contradiction, that there exists $d_{xx_k} \leq d_B$ such that $x \overset{d_{xx_k}}{\rightarrow} x_k \in P(S_k)$. There is an S_k-path from x to x_k. Let $z \overset{d_{zx_k}}{\rightarrow} x_k \in S_k$ be the last link in this path. By assumption, $z \notin B$. By connectedness, there is a link $x \overset{d_{xz}}{\rightarrow} z \in P(S_k)$ and, by monotonicity, $d_{xz} \leq d_{xx_k} \leq d_B$. Because, by assumption, there is no link $x \xrightarrow{1} x_k \in S_k$, there is no u such that $x \xrightarrow{1} u \in S_k \setminus S_{k-1}$ and because $z \neq x_k$ it follows that $x \overset{d_{xz}}{\rightarrow} z \in P(S_{k-1})$ and therefore $z \in B$, in contradiction to the assumption that no $z \in B$ and d exist such that $z \xrightarrow{d} x_k \in S$.

2. There exists $z \in B$ such that $z \xrightarrow{1} x_k \in S_k$ and B is the smallest bracket in $\mathcal{I}_{k-1}(S) = \mathcal{A}(P(S_{k-1}))$ covering z. Therefore, $B = G_0^{k-1}(z)$. Since $z \xrightarrow{1} x_k \in S_k$, $z \xrightarrow{0} x_k \notin P(S_k)$ and therefore $B = G_0^k(z) \in \mathcal{A}(P(S_k))$.

□

5.4.7. Lemma. *Let $\mathcal{C}$ be a bracketing over an utterance $U = \langle x_1, \ldots, x_n \rangle$ such that all links in $R_{\mathcal{C}}$ are of depth 0 or 1 and let S be a shortest common cover link set for $\mathcal{C}$. If, for some $1 < k \leq n$, $\mathcal{A}(P(S_{k-1})) = \mathcal{I}_{k-1}(S)$ and $B \in \mathcal{I}_k(S)$ was created by step 1 or step 2 of loop k then $B \in \mathcal{A}(P(S_k))$.*

Proof. Assume the assumptions of the lemma hold, $\mathcal{A}(P(S_{k-1})) = \mathcal{I}_{k-1}(S)$ and $B \in \mathcal{I}_k(S)$.

- *Case 1:* B was created by step 1 or step 2a of loop k. The bracket B was created by extending a bracket $B' \in \mathcal{I}_{k-1}(S) = \mathcal{A}(P(S_{k-1}))$. This means that there is $z \in B'$ such that $z \xrightarrow{d} x_k \in S_k$. Let x be the leftmost generator of B' in $\mathcal{A}(P(S_{k-1}))$ and let $d_{B'}$ be such that $B' = G_{d_{B'}}^{k-1}(x)$. By Lemma 5.4.4, $z \in [x, x_k)$. Since $z \in B'$, $x = z$ or $x \xrightarrow{d'} z \in P(S_{k-1})$ for some $d' \leq d_{B'}$.

  If B was created by step 1 of the loop then $z \xrightarrow{0} x_k \in S_k$ and, by linear transitivity, it follows that $x \xrightarrow{d'} x_k \in P(S_k)$. Therefore, since $d' \leq d_{B'}$, $x_k \in G_{d_{B'}}^k(x)$.

  If B was created by step 2a of the loop then $z \xrightarrow{1} x_k \in S_k$ and B' is not the smallest bracket covering z in $\mathcal{I}_{k-1}(S) = \mathcal{A}(P(S_{k-1}))$. This means that $B' \neq G_0^{k-1}(z)$ and, therefore, either $x \neq z$ and there is no link $z \xrightarrow{0} x \in P(S_{k-1})$ or $d_{B'} = 1$ and there is a link $x \xrightarrow{1} v \in P(S_{k-1})$. In either case, by linear transitivity, there is a link $x \xrightarrow{d''} x_k \in P(S_k)$ with $d'' \leq d_{B'}$. Therefore, $x_k \in G_{d_{B'}}^k(x)$.

In either case, $x_k \in G^k_{d_{B'}}(x)$. Since x is the leftmost generator of $G^{k-1}_{d_{B'}}(x)$, it follows from Lemma 5.4.4 that $G^k_{d_{B'}}(x) = G^{k-1}_{d_{B'}}(x) \cup \{x_k\}$. This means that $[x, x_{k-1}] \subseteq G^{k-1}_{d_{B'}}(x) = B'$ and, therefore, $B = G^k_{d_{B'}}(x) \in \mathcal{A}(P(S_k))$, as required.

- *Case 2:* B was created by step 2b. This means that there is $z \in [x_1, x_{k-1}]$ such that $z \xrightarrow{1} x_k$. Let B' be the smallest bracket covering z in $\mathcal{I}_{k-1}(S)$. Because $\mathcal{I}_{k-1}(S) = \mathcal{A}(P(S_{k-1}))$, Lemma 5.3.20 implies that $B' = G^{k-1}_0(z)$. B is, by definition, the smallest bracket covering $G^{k-1}_0(z)$ and x_k. Let x be the leftmost generator of $G^{k-1}_0(z)$, so $G^{k-1}_0(x) = G^{k-1}_0(z)$. There is no $x \xrightarrow{1} y \in S_{k-1}$ (because if there was, then $y \notin G^{k-1}_0(x) = B'$ and this contradicts the conditions of step 2b). This implies, by definition, that there is also no link $x \xrightarrow{1} y \in P(S_{k-1})$ and, therefore, $G^{k-1}_0(x) = G^{k-1}_1(x)$. Since $z \xrightarrow{1} x_k$ and either $x = z$ or $x \xrightarrow{0} z \in P(S_{k-1})$, $x_k \in G^k_1(x)$. Since x is the leftmost generator of $G^k_1(x)$, it follows from Lemma 5.4.4 that $G^k_1(x) = G^{k-1}_1(x) \cup \{x_k\}$. The bracket $G^k_1(x)$ is the smallest bracket covering $G^{k-1}_1(x)$ and x_k and, therefore, $B = G^k_1(x) \in \mathcal{A}(P(S_k))$, as required.

□

Up to this point the lemmas dealt with brackets which either are created in steps 1 and 2 of loop k or are not changed by loop k. It remains to prove that the brackets created in steps 3 and 4 of loop k are exactly those brackets in $\mathcal{A}(P(S_k))$ whose leftmost generator is x_k.

5.4.8. LEMMA. *Let $\mathcal{C}$ be a bracketing over an utterance $U = \langle x_1, \ldots, x_n \rangle$ such that all links in $R_{\mathcal{C}}$ are of depth 0 or 1 and let S be a shortest common cover link set for $\mathcal{C}$. If, for some $1 < k \leq n$, $\mathcal{A}(P(S_{k-1})) = \mathcal{I}_{k-1}(S)$ then a bracket B is generated by step 3 or 4 of loop k iff $B \in \mathcal{A}(P(S_k))$ and x_k is the leftmost generator of B.*

PROOF. Assume the assumptions of the lemma hold and $\mathcal{A}(P(S_{k-1})) = \mathcal{I}_{k-1}(S)$. Let $\mathcal{B}$ be the set of brackets maintained by the incremental algorithm when it begins steps 3 of loop k.

- *Case 1:* $B = G^k_0(x_k)$. The bracket $G^k_0(x_k)$ has a generator $x \neq x_k$ in $\mathcal{A}(P(S_k))$ iff $x \xrightarrow{0} x_k \in P(S_k)$ and $x_k \xrightarrow{0} x \in P(S_k)$ (that this is necessary is by definition and that this is sufficient is by Lemma 5.4.3). Using Corollary 5.3.14, this holds iff there is $x_k \xrightarrow{0} x_i \in S_k$ such that $x_i \xrightarrow{0} x_k \in P(S_k)$ (and x_i is then a generator of $G^k_0(x_k)$). If $x_i \neq x_k$ is a generator of $G^k_0(x_k)$ then, by Lemma 5.4.5, $G^k_0(x_k) = G^k_0(x_i) \in \mathcal{B}$. This means that x_k is the leftmost generator of $G^k_0(x_k)$ iff there is no x_i such that $x_k \xrightarrow{0} x_i \in S_k$ and

$x_k \in B_0^{\mathcal{B}}(x_i)$. This is exactly the condition for creating a link by step 3 of loop k and therefore a bracket is created by this step iff x_k is the leftmost generator of $G_0^k(x_k)$. Assume this holds. I next show that the bracket created is $G_0^k(x_k)$.

By definition, $x \in G_0^k(x_k) \setminus \{x_k\}$ iff $x_k \xrightarrow{0} x \in P(S_k)$. If $x_k \xrightarrow{0} x \in P(S)$ then (by linear transitivity) $x_k \overset{0,S_k}{\to} x$, which shows that x is inside the bracket created by step 3. Now assume that $x_k \overset{0,S_k}{\to} x$. It follows that $x_k \xrightarrow{d} x \in P^*(S_k)$ for some $d \leq 1$. If $d = 1$ then, by definition of $P^*(S_k)$, there exists $y \in (x_k, x)$ such that $x_k \xrightarrow{0} y \in P^*(S_k)$ and $y \xrightarrow{0} x_k \in P^*(S_k)$. Since (by definition) both these links are also in $P(S_k)$, it follows from Lemma 5.4.3 that $G_0^k(x_k) = G_0^k(y)$, contrary to the assumption that x_k is the leftmost generator of $G_0^k(x_k)$. Therefore, $d = 0$ and, by definition, $x_k \xrightarrow{0} x \in P(S_k)$ which shows that $x \in G_0^k(x_k)$. Therefore, the bracket created by step 3 is contained in $G_0^k(x_k)$, which together with what has been already shown shows that the two brackets are equal, as required.

- *Case 2:* $B = G_1^k(x_k) \neq G_0^k(x_k)$.

Let $\mathcal{B}'$ be the set of brackets maintained by the incremental algorithm when it begins steps 4 of loop k. By what has been shown above, $\mathcal{B}' = \mathcal{B} \cup \{G_0^k(x_k)\}$. By definition, $G_1^k(x_k) \neq G_0^k(x_k)$ iff there is a link $x_k \xrightarrow{1} x_i \in S_k$. Therefore, if $G_1^k(x_k) = G_0^k(x_k)$ step 4 does not add any bracket. I will next show that if there exists a link $x_k \xrightarrow{1} x_i \in S_k$ then the bracket defined in step 4 of the loop is equal to $G_1^k(x_k)$. Therefore, if $G_1^k(x_k) \neq G_0^k(x_k)$, step 4 adds $G_1^k(x_k)$ to $\mathcal{B}'$ iff $G_1^k(x_k) \notin \mathcal{B}'$. By Lemma 5.4.5 and Lemma 5.4.7, $G_1^k(x_k) \notin \mathcal{B}'$ iff its leftmost generator is x_k, which means that step 4 adds $G_1^k(x_k)$ to $\mathcal{B}'$ iff x_k is the leftmost generator of $G_1^k(x_k)$, as required.

To complete the proof, assume that there exists a link $x_k \xrightarrow{1} x_i \in S_k$ (and therefore also $G_1^k(x_k) \neq G_0^k(x_k)$). Let C be the minimal bracket covering x_k and all x such that $x_k \overset{S_k}{\to} x$. This is the bracket created by step 4 and it remains to show that $C = G_1^k(x_k)$. Let $x \in G_1^k(x_k) \setminus \{x_k\}$. By definition, $x_k \to x \in P(S_k)$, which means that $x_k \overset{S_k}{\to} x$. Therefore $x \in C$ which implies that $G_1^k(x_k) \subseteq C$. Now let $x \in C \setminus \{x_k\}$. By definition, $x_k \overset{S_k}{\to} x$ and $x_k \xrightarrow{d} x \in P^*(S_k)$ for some $d \leq 1$. Because $x_k \xrightarrow{1} x_i \in S_k$, it follows from the definition of $P(S_k)$ that $x_k \xrightarrow{d} x \in P(S_k)$. Therefore, $x \in G_1^k(x_k)$ which shows that $C \subseteq G_1^k(x_k)$, which together with the opposite inclusion shown above proves that $C = G_1^k(x_k)$.

□

Collecting all these lemmas together allows the correctness of the incremental bracket reconstruction algorithm to be proved.

5.4.9 (2.6.6). Theorem (reconstruction). *Let $\mathcal{C}$ be a bracketing over an utterance $U = \langle x_1, \dots, x_n \rangle$ and let S be a shortest common cover link set for this bracketing. If S contains only links of depth 0 or 1 then applying Algorithm 5.4.1 to S outputs $\mathcal{C}$. The algorithm runs in time linear in the length of U.*

Proof. Assume the assumptions of the theorem hold. I will show that for every $1 \leq k \leq n$, $\mathcal{A}(P(S_k)) = \mathcal{I}_k(S)$. This proves the theorem because the bracketing $\mathcal{I}_n(S)$ is the bracketing output by the incremental algorithm and the bracketing $\mathcal{A}(P(S_n)) = \mathcal{A}(P(S)) = \mathcal{C}$ (by Lemma 5.3.21).

The case $k = 1$ follows directly from the definitions ($\mathcal{I}_0(S) = \{\langle x_1 \rangle\}$ and $\mathcal{A}(P(S_1)) = \{\langle x_1 \rangle\}$). The induction step from $k-1$ to k follows from the lemmas proved above.

To complete the proof, I show that the algorithm runs in time linear in the length of U. From directional uniqueness (Lemma 5.1.25), it follows that at each step of the algorithm there is at most one link $x_i \xrightarrow{0} x_k \in S_k$ or $x_i \xrightarrow{1} x_k \in S_k$. It can then be easily verified that with an appropriate implementation, steps 1 and 2 of the algorithm loop can each be performed in constant time. Similarly, it is easy to check that the longest paths $x_k \overset{0,S_k}{\to} x$ and $x_k \overset{S_k}{\to} x$ can be maintained by the algorithm in linear time. Steps 3 and 4 of the algorithm loop can then also be carried out in constant time. $\square$

5.5 Incremental Parsing

A common cover link parser is required to output a shortest common cover link set for the utterance it is parsing. I also require that the parser be incremental, that is, it should read the words of the utterance one by one and as each word is read, may add only links between that word and previously read words. To define incremental parsing, the following notation is used.

Notation Given an utterance U, I write $x_k(U)$ for the k'th word in U (when there is no risk of confusion, I simply write x_k). I also write U_k for the prefix $[x_1(U), x_k(U)]$ of length k of U. Similarly, if S is a shortest common cover link set for a bracketing over U, I write S_k for the set $\{x \xrightarrow{d} y \in S \ : \ x, y \in U_k\}$, the restriction of S to U_k.

It is now possible to define incremental common cover link parsing.

5.5.1 (3.1.1). Definition. [incremental parsing]

1. A common cover link parser is a function $\mathcal{P}$ on the set of utterances such that, for any utterance U, $\mathcal{P}(U)$ is a shortest common cover link set over U.

2. The parser $\mathcal{P}$ is incremental if for every prefix U_k of U, $\mathcal{P}(U_k) = S_k$ is a shortest common cover link set over U_k and, for each $2 \leq k \leq n$, $S_{k-1} \subseteq S_k$ and $S_k \setminus S_{k-1}$ contains only links which have one end at $x_k(U)$.

Incremental parsing is well-defined because, by Lemma 5.3.4, if S is a shortest common cover link set over an utterance U then the restriction S_k of S to a prefix U_k of U is itself a shortest common cover link set for a bracketing of U_k.

The parser adds links to the parse one by one. To ensure that the set of links S_k constructed by the incremental parser for a prefix U_k of an utterance is indeed a shortest common cover link set, the set S_k must satisfy the characterizing conditions of Definition 5.3.1. The first five conditions, *monotonicity*, *minimality*, *connectedness*, *blocking* and *equality* are satisfied with every link added to the parse. The last property, *resolution*, cannot be always satisfied but, when it is violated, the violation can always be repaired by adding additional links. What links may be added in each step is described in the following sections.

5.5.1 Adjacency

I begin with the first four properties, *monotonicity*, *minimality*, *connectedness* and *blocking* which I call the *adjacency properties*. To ensure that the adjacency properties are satisfied with each link being added, a link should be added from a word x to a word y only if y is *adjacent* to x and the pair $\langle x, y \rangle$ is not covered by previous links, as defined and shown by the following definitions and lemma.

5.5.2 (3.2.1). Definition. [adjacency] Let L be a set of common cover links of depth 0 or 1 over an utterance U. A word y is adjacent to x with depth $d \leq 1$ relative to L (written $x \dashv^L_d y$) iff for every $z \in (x, y)$:

1. $x \overset{L}{\rightarrow} z$ (connectedness).

2. $z \rightarrow y \notin L$ (minimality).

3. There is no $w \in U$ such that $z \overset{1}{\rightarrow} w \in L$ and $z \overset{L}{\rightarrow} x$ (blocking).

The depth d is 1 iff there exists $z \in (x, y)$ such that $z \overset{L}{\rightarrow} x$ (blocking) or $x \overset{1}{\rightarrow} z \in L$ (monotonicity). Otherwise, $d = 0$.

An adjacency $x \dashv^L_d y$ is *unused* if $x \rightarrow y \notin L$.

Notation I write $x \dashv^L y$ if there exists $d \leq 1$ such that $x \dashv^L_d y$.

5.5.3 (3.2.2). Definition. [covering links] A pair of words $\langle x, y\rangle$ in an utterance U is *covered* by a link $u \xrightarrow{d'} v$ over U if $x \in (u, v)$ and $y \in [u, v]$. The pair $\langle x, y\rangle$ is covered by a set L of common cover links over U if there exists a link $u \xrightarrow{d'} v \in L$ such that $u \xrightarrow{d'} v$ covers $\langle x, y\rangle$.

5.5.4 (3.2.3). Lemma. *Let U be an utterance. Let L be a set of common cover links over U such that all links in L are of depth 0 or 1.*

1. *If L satisfies the adjacency properties and if $x \dashv^L_d y$ and the pair $\langle x, y\rangle$ is not covered by L then, for any $d \leq d' \leq 1$, $L \cup \{x \xrightarrow{d'} y\}$ satisfies the adjacency properties.*
2. *There exists a set L and a link $x \xrightarrow{d'} y$ such that L satisfies all characterizing conditions of Definition 5.3.1, $x \dashv^L_d y$ for $d \leq d'$ but $L \cup \{x \xrightarrow{d'} y\}$ violates minimality or blocking.*

Proof.

1. Assume that L and $x \xrightarrow{d'} y$ are as in the first part of the lemma and let $L' = L \cup \{x \xrightarrow{d'} y\}$. Monotonicity and connectedness of L' are immediate from the definition of adjacency.

   Assume, by contradiction, that L' does not satisfy minimality. There is, therefore, a link $u_1 \xrightarrow{d''} u_2 \in L'$ and $v \in (u_1, u_2)$ such that $u_1 \xrightarrow{L'} v$ and $v \xrightarrow{L'} u_2$. If $u_1 \xrightarrow{d''} u_2 = x \xrightarrow{d'} y$ then $v \xrightarrow{L} y$ for $v \in (x, y)$, in contradiction to the assumption that $x \dashv^L_d y$. Therefore, $u_1 \xrightarrow{d''} u_2 \in L$. Let $w \xrightarrow{d'''} u_2$ be the last link in the L'-path from u_1 to u_2 via v. Clearly, $w \in [v, u_2) \subset [u_1, u_2)$. By the connectedness of L, $u_1 \xrightarrow{L} w$ and since L satisfies minimality, $w \xrightarrow{d'''} u_2 \notin L$, so $w \xrightarrow{d'''} u_2 = x \xrightarrow{d'} y$. But, then, for $u_1 \xrightarrow{d''} u_2 \in L$, it holds that $x \in (u_1, u_2)$ and $y \in [u_1, u_2]$. Therefore, L covers $\langle x, y\rangle$, contrary to the assumptions of the lemma. So L' satisfies minimality after all.

   Assume, by contradiction, that L' does not satisfy blocking. There are, therefore, $u_1 \xrightarrow{d_1} u_2 \in L'$, $v \in (u_1, u_2)$ and $v \xrightarrow{d_2} w \in L'$ such that $v \xrightarrow{L'} u_1$ and either $d_1 = 0$ or $d_2 = 1$. If $u_1 \xrightarrow{d_1} u_2 = x \xrightarrow{d'} y$ then $v \in (x, y)$, $v \xrightarrow{d_2} w \in L$ and $v \xrightarrow{L} x$ which implies that either there is no adjacency $x \dashv^L_d y$ (if $d_2 = 1$) or $d = 1$, in which case $0 = d_1 = d' < d$. In either case, this contradicts the assumptions of the lemma. Therefore, $u_1 \xrightarrow{d_1} u_2 \in L$. As before, the covering condition in the assumptions of the lemma imply that every link on the L'-path from v to u_1 must be in L and, therefore (because L satisfies blocking), $d_1 = 1$. By the assumption that either $d_1 = 0$ or $d_2 = 1$, this implies $d_2 = 1$ and (because L satisfies blocking) $v \xrightarrow{d_2} w \notin L$, which means that $v \xrightarrow{d_2} w = x \xrightarrow{d'} y$. If $y = w \in [u_1, u_2]$ then (because $x = v \in (u_1, u_2)$ and $u_1 \xrightarrow{1} u_2 \in L$) L covers $\langle x, y \rangle$, contrary to the assumption. Therefore, $y = w \notin [u_1, u_2]$ and there remain two possibilities to examine: $u_1 \in (x, y)$ and $u_2 \in (x, y)$. In either case, by connectedness of L, $u_1 \xrightarrow{L} x$. If $u_1 \in (x, y)$ then $u_1 \xrightarrow{L} x$ and $u_1 \xrightarrow{1} u_2 \in L$ contradict the assumption that $x \dashv^L_d y$. Now assume that $u_2 \in (x, y)$. If $x \xrightarrow{L} u_2$ then this together with $u_1 \xrightarrow{L} x$ is a contradiction to the minimality of L. But if there is no L-path from x to u_2 then this contradicts the adjacency $x \dashv^L_d y$. In either case we get a contradiction, which proves that L' does satisfy blocking, after all.

2. It is sufficient to give here two simple examples. A violation of minimality is given by: $$\begin{tikzcd} w \arrow[r] \arrow[rr, bend left] & x \arrow[r, dashed] & y \end{tikzcd}$$ and a violation of blocking is given by: $$\begin{tikzcd} w & x \arrow[r, dashed, shift right] & y \arrow[l, shift right] \arrow[ll, bend right] \end{tikzcd}$$. The solid links in the diagrams are in L. All links are of depth 0.

$\square$

The following lemma describes the adjacencies of each word when the set of links satisfies the adjacency properties.

5.5.5 (3.2.4). Lemma. *Let L be common cover link set over an utterance U such that all links in L are of depth 0 or 1 and such that L satisfies the adjacency properties.*

1. *If x and y are consecutive words in the utterance then $x \dashv^L_0 y$ and $y \dashv^L_0 x$.*

2. *If $x \xrightarrow{d'} y \in L$ then $x \dashv^L_d y$ for some $d \leq d'$.*

3. *If, for $y_1 \in (x, y_2)$, $x \dashv^L_{d_1} y_1$ and $x \dashv^L_{d_2} y_2$ then $x \xrightarrow{d'} y_1 \in L$ for some $d_1 \leq d'$ (in words: on each side of x there is at most one unused adjacency and this word is the adjacent word furthest away from x on that side).*

Proof. Let L be as in the lemma.

1. If x and y are consecutive words in the utterance then (x, y) is empty and it follows directly from the definition that $x \dashv^L_0 y$ and $y \dashv^L_0 x$.

2. Let $x \xrightarrow{d'} y \in L$. That $x \dashv^L_d y$ for $d \leq d'$ follows directly from the adjacency properties of L.

3. Assume that $y_1 \in (x, y_2)$, $x \dashv^L_{d_1} y_1$ and $x \dashv^L_{d_2} y_2$. Since $x \dashv^L_{d_2} y_2$, it follows by definition that $x \xrightarrow{L} y_1$. Assume, by contradiction, that $x \rightarrow y_1 \notin L$. There is, therefore, $v \in (x, y_1)$ such that $x \xrightarrow{L} v$ and $v \rightarrow y_1 \in L$. But $v \rightarrow y_1 \in L$ contradicts the definition of $x \dashv^L_{d_1} y_1$. Therefore, $x \xrightarrow{d} y_1 \in L$, after all. From the monotonicity and blocking of L it follows directly that $d_1 \leq d$.

□

5.5.2 Incremental Parser

This section defines the incremental parsing algorithm and proves its correctness. It is shown that the parser always outputs a shortest common cover link set and that any shortest common cover link set can be produced by the incremental parser.

The Incremental Parsing Algorithm

The definition of the incremental parsing algorithm requires the definition of those links which may be added to the parse at each step. The conditions imposed on these links are a combination of incrementality, adjacency, equality and an additional condition (forcing) which is required to ensure that resolution violations can be properly repaired.

5.5.6 (3.2.5). Definition. [incrementally addable link] Let L be a set of common cover links over a prefix U_k of an utterance U such that all links in L are of depth 0 or 1 and such that L satisfies the adjacency properties and equality. A link $x \xrightarrow{d'} y$ is incrementally addable to L over U_k iff:

1. (*incrementality*) $x_k \in \{x, y\} \subseteq U_k$ and $d' \leq 1$.

2. (*adjacency*) For some $d \leq d'$, $x \dashv^L_d y$ is an unused adjacency in L.

3. (*non-covered*) L does not cover the pair $\langle x, y \rangle$.

4. (*equality*) The set $L \cup \{x \xrightarrow{d'} y\}$ satisfies equality.

5. (*forcing*) If there is $u \xrightarrow{1} x \in L$ such that $y \in [u, x)$ then $d' = 1$.

An additional definition required by the incremental algorithm is that of a resolution violation.

5.5.7 (3.2.6). Definition. [resolution violation] Let L be a set of links over an utterance U such that all links in L are of depth 0 or 1. A tuple $\langle x, z, d_1, d_2 \rangle$ is a *resolution violation* in L if $x, z \in U$, $x \in [x_1(U), z)$ and there are $y \in (x, z)$, $x' \in (x, y]$ and $z' \in (z, y]$ such that $x \xrightarrow{d_1} x' \in L$, $x' \xrightarrow{L} y$, $z \xrightarrow{d_2} z' \in L$ and $z' \xrightarrow{L} y$ but there is no v such that either $x \xrightarrow{d_1} v \in L$ and $v \xrightarrow{L} z$ or $z \xrightarrow{d_2} v \in L$ and $v \xrightarrow{L} x$.

A tuple $\langle x, z, d_1, d_2 \rangle$ is a *minimal resolution violation* in L if there is no resolution violation $\langle u, w, d_3, d_4 \rangle$ in L such that $[u, w] \subset [x, z]$ or $[u, w] = [x, z]$ and $d_3 > d_1$.

With the definition of incrementally addable links and resolution violations, the incremental parsing algorithm is simple.

5.5.8 (3.3.1). Algorithm. [incremental parser] Let U be an utterance of length n. The algorithm maintains a set L of common cover links.

- Initialize $L = \emptyset$ and $k = 2$.
- While $k \leq n$:
  1. If there is a minimal resolution violation $\langle x, z, d_1, d_2 \rangle$ in L, add an addable link $u \xrightarrow{d} v$ such that $u, v \in [x, z]$ and such that if $u = x$ then $d = d_1$ and if $u = z$ then $d = d_2$.
  2. Otherwise, increment k by 1 or add to L a link incrementally addable to L over U_k.

Correctness of the Parsing Algorithm

The rest of this section is dedicated to the proof of Theorem 5.5.17 which shows that the incremental parser is well-defined, always outputs a shortest common cover link set and can output any shortest common cover link set. This proof relies on several lemmas.

The first lemma describes a simple property of link paths which follows from the adjacency properties.

5.5.9. Lemma. *Let L be a set of common cover links over an utterance U such that all links are of depth 0 or 1 and such that L satisfies the adjacency properties. If $y \in (x, z]$, $x \xrightarrow{L} y$, $w \in [x, y]$ and $w \xrightarrow{L} z$ then w is on the L-path from x to y.*

PROOF. Assume that L, x, w, y and z are as in the assumptions of the lemma. By connectedness, $x \xrightarrow{L} y$ implies $x \xrightarrow{L} w$ and $w \xrightarrow{L} z$ implies $w \xrightarrow{L} y$. Therefore, w is on an L-path from x to y and since, by Lemma 5.3.7, this path is unique, the claim holds. □

The next two lemmas describe the possibilities available to an incremental parser when it creates links between an already parsed prefix U_k of an utterance and the next word x_{k+1}. The lemmas show that any shortest common cover link set can be constructed by an incremental parser which only adds links between adjacent words which are not covered by previously constructed links. Moreover, they show that at most one link can be constructed in this way from U_k to x_{k+1}.

Notation I write $R_k = \{x \xrightarrow{d} x_k \in S_k\}$ for the set of links in $S_k \setminus S_{k-1}$ which end at x_k and $L_k = \{x_k \xrightarrow{d} x \in S_k\}$ for the set of links in $S_k \setminus S_{k-1}$ which begin at x_k. The *length* of a link $x \xrightarrow{d} y$ over U is $|x \xrightarrow{d} y| = |(x, y]|$, the size of the set $(x, y]$. This is the distance between x and y in U.

5.5.10. LEMMA. *Let $\mathcal{C}$ be a bracketing over an utterance U such that all links in $R_{\mathcal{C}}$ are of depth 0 or 1 and let S be a shortest common cover link set for $\mathcal{C}$. Let $S_k \subseteq L \subseteq S_{k+1}$.*

1. *The set R_{k+1} contains at most one link.*
2. *If $R_{k+1} = \{x \xrightarrow{d'} x_{k+1}\}$ then $x \dashv_d^L x_{k+1}$ for some $d \leq d'$.*
3. *If $x_{k+1} \xrightarrow{d'} x \in L_{k+1}$ then $x_{k+1} \dashv_d^L x$ for some $d \leq d'$ iff $\{l \in L_{k+1} \ : \ |l| < |x_{k+1} \xrightarrow{d'} x|\} \subseteq L$.*

PROOF. Assume that L is as in the lemma.

1. That R_{k+1} contains at most one link follows directly from directional uniqueness (Lemma 5.1.25).
2. Assume that $R_{k+1} = \{x \xrightarrow{d'} x_{k+1}\}$. Since S_{k+1} satisfies monotonicity, minimality, connectedness and blocking, it follows from Lemma 5.5.5 that $x \dashv_d^{S_{k+1}} x_{k+1}$ for some $d \leq d'$. By directional uniqueness, for any $y \in U_k \setminus \{x\}$, $y \to x_{k+1} \notin S_{k+1}$. The adjacency $x \dashv_d^{S_{k+1}} x_{k+1}$ therefore depends only on links over U_k and therefore $x \dashv_d^L x_{k+1}$.

3. Let $x_{k+1} \xrightarrow{d'} x \in L_{k+1}$ and assume that there is $y \in (x_{k+1}, x)$ such that $x_{k+1} \xrightarrow{d''} y \in L_{k+1}$ (a link shorter than $x_{k+1} \xrightarrow{d'} x$). Assume, also, that $x_{k+1} \dashv^{L}_{d} x$ for some $d \leq d'$. By definition of adjacency, $x_{k+1} \xrightarrow{L} y$. Every L-path is also an S_{k+1}-path and since $x_{k+1} \xrightarrow{d''} y \in S_{k+1}$ is the S_{k+1}-path from x_{k+1} to y and by the minimality of S_{k+1} it follows that $x_{k+1} \xrightarrow{d''} y$ is also the L-path from x_{k+1} to y and therefore $x_{k+1} \xrightarrow{d''} y \in L$, as required.

   Now assume that $\{l \in L_{k+1} \ : \ |l| < |x_{k+1} \xrightarrow{d'} x|\} \subseteq L$. By Lemma 5.5.5, $x_{k+1} \dashv^{S_{k+1}}_{d''} x$ for some $d'' \leq d'$. Any link $S_{k+1} \setminus L$ can only block the adjacency of x to x_{k+1} or increase its d. Therefore, $x_{k+1} \dashv^{L}_{d} x$ for some $d \leq d'' \leq d'$, as required.

□

5.5.11. Lemma. *Let L be a set of common cover links of depth 0 or 1 over the prefix U_{k+1} of an utterance U.*

1. *For any $x \in U_k$, the pair $\langle x_{k+1}, x\rangle$ is not covered by L.*

2. *If, for some $x \in U_k$, $x \xrightarrow{d'} x_{k+1} \in L$ then, for any $y \in U_k \setminus \{x\}$, $y \dashv^{L}_{d} x_{k+1}$ implies that $\langle y, x_{k+1}\rangle$ is covered by L.*

Proof. Let L be a set as in the assumptions of the lemma.

1. That the pair $\langle x_{k+1}, x\rangle$ is not covered by L is immediate from the definition of a covering link.

2. Assume that , for some $x \in U_k$, $x \xrightarrow{d'} x_{k+1} \in L$, $y \in U_k \setminus \{x\}$ and $y \dashv^{L}_{d} x_{k+1}$. If $x \in (y, x_{k+1})$ then the link $x \xrightarrow{d'} x_{k+1} \in L$ contradicts the adjacency of x_{k+1} to y. Therefore, $y \in (x, x_{k+1})$ and the pair $\langle y, x_{k+1}\rangle$ is covered by $x \xrightarrow{d'} x_{k+1} \in L$.

□

Additional lemmas needed to prove the correctness of the incremental algorithm describe the state of the parser at any point in the processing of an utterance. A single step of the algorithm is either an addition of a link or the incrementation of the last word position indicator k. The following notation will be used to describe the internal state of the parser after i steps have been performed.

Notation A complete run of the parser over an utterance U is denoted by $\theta(U)$. The first i steps of a run $\theta(U)$ are denoted by $\theta_i(U)$. The internal state of the incremental parser after the completion of $\theta_i(U)$ is $\langle L(\theta_i(U)), k(\theta_i(U))\rangle$ where $L(\theta_i(U))$ is the set of links the parser has constructed in $\theta_i(U)$ and $k(\theta_i(U))$ is the value of the last word position indicator k after $\theta_i(U)$. Finally, for any l, I write $S_l(L(\theta_i(U)))$ for the restriction $\{x \xrightarrow{d} y \in L(\theta_i(U)) \ : \ x, y \in U_l\}$ of $L(\theta_i(U))$ to the utterance prefix U_l.

It is easy to check that at each step of the incremental parser the set of links maintained by the parser satisfies the adjacency properties.

5.5.12. Lemma. *Let $\theta(U)$ be a run of the parser over an utterance U. For every i, $L(\theta_i(U))$ satisfies the adjacency properties and equality.*

Proof. The proof is by induction on i. The claim clearly holds for $i = 1$. Because of the adjacency and non-covered conditions in the definition of an addable link (Definition 5.5.6) it follows from Lemma 5.5.4 that if $L(\theta_{i-1}(U))$ satisfies the adjacency properties then so does $L(\theta_i(U))$. That equality holds at each step follows from the equality condition in the definition of an addable link. □

It now remains to show that the resolution property is satisfied by the set of links generated by the incremental parsing algorithm. The proof of this property is more complicated because the resolution property can be violated at some intermediate steps of the parse. The following lemma describes some properties of the resolution violations which may be created by the incremental parser.

5.5.13. Lemma. *Let $\theta(U)$ be a run of the parser over an utterance U and let $\langle L, k\rangle = \langle L(\theta_i(U)), k(\theta_i(U))\rangle$ be the state of the parser for a prefix $\theta_i(U)$ of the run. If $S_{k-1}(L)$ is a shortest common cover link set and $\langle x, z, d_1, d_2\rangle$ is a resolution violation in L then $z = x_k(U)$ and the following holds:*

1. *There is $y \in (x, x_k)$ and $x' \in (x, y]$ such that $x_k \xrightarrow{d_2} y \in L$, $x \xrightarrow{d_1} x'$ and $x' \xrightarrow{L} y$.*
2. *If $\langle x', x'_k, d'_1, d'_2\rangle$ is a resolution violation in L then $d_2 = d'_2$.*
3. *If $y \in (x, x_k)$ and $x' \in (x, y]$ are such that $x \xrightarrow{d_1} x' \in L$, $x' \xrightarrow{L} y$ and $x_k \xrightarrow{d_2} y \in L$ then for any $w \in (x, y]$, there is no L-path from w to x_k and, for any $w \in [x_1, y]$, if $x_k \xrightarrow{d} w \in L$ then $d = d_2$.*
4. *If there is $v \in (x, x_k]$ such that $x \xrightarrow{d} v \in L$ and $v \xrightarrow{L} x_k$ then $d = d_2 = 1$.*

5. *There is no L-path from x_k to x.*

Proof. Assume the assumption of the lemma holds and $S_{k-1}(L)$ is a shortest common cover link set. Assume that $\langle x, z, d_1, d_2 \rangle$ is a resolution violation in L. If $x, z \in U_{k-1}$ then this is a resolution violation in $S_{k-1}(L)$, in contradiction to the assumption that $S_{k-1}(L)$ is a shortest common cover link set. Since, by definition, $x \in [x_1(U), z)$, it follows that $z = x_k(U)$.

1. By definition, there are $y \in (x, z)$, $x'' \in (x, y]$ and $z' \in (z, y]$ such that $x \xrightarrow{d_1} x'' \in L$, $x'' \xrightarrow{L} y$, $z \xrightarrow{d_2} z' \in L$ and $z' \xrightarrow{L} y$. To complete the proof of this part of the lemma, it is enough to show that there is $x' \in (x, z']$ such that $x \xrightarrow{d_1} x'$ and $x' \xrightarrow{L} z'$. Assume, by contradiction, that no such x' exists. Specifically, $y \neq z'$ follows and there exist $z'' \in (z', y]$ and d_3 such that $z' \xrightarrow{d_3} z'' \in L$ and $z'' \xrightarrow{L} y$. Since $x, z' \in U_{k-1}$ and since $S_{k-1}(L)$ is a shortest common cover link set, it follows from resolution that there is $x' \in [x, z']$ such that either $x \xrightarrow{d_1} x' \in L$ and $x' \xrightarrow{L} z'$, or $z' \xrightarrow{d_3} x' \in L$ and $x' \xrightarrow{L} x$. If $z' \xrightarrow{d_3} x' \in L$ and $x' \xrightarrow{L} x$ then (because $z \xrightarrow{d_2} z' \in L$) $\langle x, z, d_1, d_2 \rangle$ is not a resolution violation, contrary to the assumption. Therefore, there exists $x' \in (x, z']$ such that $x \xrightarrow{d_1} x' \in L$ and $x' \xrightarrow{L} z'$, which completes the proof.

2. Let i_0 be the first step such that $k(\theta_{i_0}(U)) = k$. The proof is by induction, showing that the claim holds for $L(\theta_j(U))$ for every $i_0 \leq j \leq i$. Since $S_{k-1}(L)$ is a shortest common cover link set, there are no resolution violations in $L(\theta_{i_0}(U))$ and the claim holds.

   Assume the claim holds for $j-1$. By the previous part of the lemma, any resolution violation created in step $i_0 < j$ is created when a link $x_k \xrightarrow{d} y$ is added by the parser. All resolution violations created by adding such a link are of the form $\langle x, x_k, d_1, d \rangle$ (same d in all violations). Therefore, if there are no resolution violations in $L(\theta_{j-1}(U))$ then all resolution violations in $L(\theta_j(U))$ (if any) were created by adding a link $x_k \xrightarrow{d} y$ at step j and therefore all are of the form $\langle x, x_k, d_1, d \rangle$ and the claim holds. If there were resolution violations in $L(\theta_{j-1}(U))$ then, by the induction hypothesis, they were all of the form $\langle x, x_k, d_1, d \rangle$ with the same d. By the incremental parser algorithm, if the algorithm adds a link $x_k \xrightarrow{d'} y$ in step j then $d' = d$. Therefore, any new violation created in this step is also of the form $\langle x, x_k, d_1, d \rangle$ (with the same d) and the claim $d'_2 = d_2$ holds.

3. Let $y \in (x, x_k)$ and $x' \in (x, y]$ be such that $x \xrightarrow{d_1} x' \in L$, $x' \xrightarrow{L} y$ and $x_k \xrightarrow{d_2} y \in L$. Let $w \in [x_1, y]$. Assume that $x_k \xrightarrow{d} w \in L$. If $w = y$ it is obvious that $d = d_2$, so assume that $w \in [x_1, y)$. By Lemma 5.5.5 and the

definition of incrementally addable links, the link $x_k \xrightarrow{d} w \in L$ was added by the parser after the link $x_k \xrightarrow{d_2} y$ was added and therefore also after the resolution violation $\langle x, x_k, d_1, d_2 \rangle$ was created. By the algorithm of the incremental parser, it follows that $d = d_2$, as required.

Assume now that $w \in (x, y]$. I now show that there is no L-path from w to x_k. Assume, by contradiction, that $w \xrightarrow{L} x_k$. If $w \in (x, x')$ then, by Lemma 5.5.9, w is on the L-path from x to x', in contradiction to the assumption that $x \xrightarrow{d_1} x' \in L$. Therefore, $w \in [x', y]$ and, by connectedness, $x' \xrightarrow{L} w$. Together with $x \xrightarrow{d_1} x' \in L$ and $w \xrightarrow{L} x_k$, this contradicts the assumption that $\langle x, x_k, d_1, d_2 \rangle$ is a resolution violation.

4. By the first part of the lemma, there are $y \in (x, x_k)$ and $x' \in (x, y]$ such that $x_k \xrightarrow{d_2} y \in L$, $x \xrightarrow{d_1} x' \in L$ and $x' \xrightarrow{L} y$. Assume that $v \in (x, x_k]$ such that $x \xrightarrow{d} v \in L$ and $v \xrightarrow{L} x_k$. By the previous part of the lemma, $v \in (y, x_k]$. If $d = d_1$ then $\langle x, x_k, d_1, d_2 \rangle$ is not a resolution violation, contrary to the assumption. Therefore, $d \neq d_1$. Because $x' \in [x, v)$, monotonicity implies that $d_1 < d$ and therefore $d = 1$. If $v = x_k$, $d_1 < d$ also implies that the link $x \xrightarrow{1} x_k \in L$ was added to the set before the resolution violation was created. By the forcing property of incrementally addable links, this implies that $d_2 = 1$. If $v \in (y, x_k)$ then $v \xrightarrow{L} x_k$ implies, by blocking, that $d_2 = 1$.

5. By the first part of the lemma, there are $y \in (x, x_k)$ and $x' \in (x, y]$ such that $x_k \xrightarrow{d_2} y \in L$, $x \xrightarrow{d_1} x' \in L$ and $x' \xrightarrow{L} y$. Assume, by contradiction, that $x_k \xrightarrow{L} x$. Let $x_k \xrightarrow{d} w \in L$ be the first link in this path. If $w \in (x_k, y)$ then, by connectedness (from $w \xrightarrow{L} x$), $w \xrightarrow{L} y$ which together with $x_k \xrightarrow{d} w \in L$ and $x_k \xrightarrow{d_2} y \in L$ contradicts the minimality property of L. Therefore, $y \in [w, x_k)$ and, by part 3 of the lemma, $d = d_2$. This means that $x_k \xrightarrow{d_2} w \in L$ and $w \xrightarrow{L} x$, in contradiction to the assumption that $\langle x, x_k, d_1, d_2 \rangle$ is a resolution violation. This shows that there is no L-path from x_k to x, after all.

□

Now it can be shown that the incremental parser is well-defined. The one respect in which the algorithm is possibly not well-defined is in the requirement that, as long as there is a resolution violation, the algorithm must continue adding addable links (of a given depth) and is not allowed to advance to the next word. The following lemma shows that as long as there is a resolution violation, an appropriate addable link exists.

5.5.14. Lemma (incremental parser is well-defined). *Let $\theta(U)$ be a run of the parser over an utterance U and let $\langle L, k \rangle = \langle L(\theta_i(U)), k(\theta_i(U)) \rangle$ be the state* *of the parser for a prefix $\theta_i(U)$ of the run. If $S_{k-1}(L)$ is a shortest common cover link set and $\langle x, x_k, d_1, d_2\rangle$ is a minimal resolution violation in L then there is a link $u \xrightarrow{d} v$ which is incrementally addable to L over U_k and such that $u, v \in [x, z]$ and if $u = x$ then $d = d_1$ and if $u = z$ then $d = d_2$.*

Proof. Assume the assumption of the lemma hold and $S_{k-1}(L)$ is a shortest common cover link set. Assume that $\langle x, x_k, d_1, d_2\rangle$ is a minimal resolution violation in L and (using the first part of Lemma 5.5.13) let $y \in (x, x_k)$ and $x' \in (x, y]$ be such that $x_k \xrightarrow{d_2} y \in L$, $x \xrightarrow{d_1} x' \in L$ and $x' \xrightarrow{L} y$. By Lemma 5.5.13, there is no L-path from x_k to x and for every $w \in (x, y]$, there is no L-path from w to x_k and if $x_k \xrightarrow{d} w \in L$ then $d = d_2$. This shows that there exists $v \in [x, y)$ such that $x_k \dashv^L_{d_2} v$ is an unused adjacency in L. I will show that $x_k \xrightarrow{d_2} v$ is an incrementally addable link to L over U_k. The incrementality and adjacency conditions in the definition of incrementally addable links hold here by definition. By Lemma 5.5.11, the pair $\langle x_k, v\rangle$ is not covered by L and, therefore, the non-covered condition holds. It remains to show that the equality and forcing conditions hold.

If there is $z \in (v, x_k]$ such that $v \xrightarrow{d} z \in L$ and $z \xrightarrow{L} x_k$ then $v \in [x, y)$ together with parts 3 and 4 of Lemma 5.5.13 imply that $v = x$ and $d = d_2$. Therefore, $x_k \xrightarrow{d_2} v$ satisfies the equality condition for an L-path from v to x_k.

Now assume that for some $z \in [x_1, v]$, $z \xrightarrow{d} x_k \in L$. As before, $z \in [x_1, x]$. If $z = x$ then, by Lemma 5.5.13, $d = d_2$, so equality and forcing hold. If $z \in [x_1, x)$ then the link $z \xrightarrow{d} x_k$ must have been added before the resolution violation $\langle x, x_k, d_1, d_2\rangle$ was created. This means that it must have been added before the link $x_k \xrightarrow{d_2} y$. If $d = 1$, this means that when $x_k \xrightarrow{d_2} y$ was added, forcing required that $d_2 = 1$ and therefore both equality and forcing hold for the link $x_k \xrightarrow{d_2} v$. If $d = 0$ then, by blocking (because $v \in (z, x_k)$), there is no L-path $v \xrightarrow{L} z$ and therefore equality holds. Forcing holds trivially. This completes the proof. □

An immediate corollary of this last lemma is that the incremental parser eventually repairs any resolution violation it creates.

5.5.15. Corollary. *Any run of the incremental parser, Algorithm 5.5.8, outputs a shortest common cover link set.*

Proof. All that needs to be shown is that the parser always terminates and that when the parser increments the counter k, the set of links L maintained by the parser has no resolution violation (the adjacency and equality properties hold for L at every step of the parser, by the definition of an addable link and Lemma 5.5.4). This follows directly from Lemma 5.5.14, which shows that as long as a resolution violation exists, there are addable links which the parser can add.

Since the number of links is finite, the parser must terminate and the output of the parser contains no resolution violation. □

Next, I show that any shortest common cover link set can be constructed by some run of the incremental parser.

5.5.16. Lemma. *Let S be a shortest common cover link set over an utterance U of length n such that all links in S are of depth 0 or 1. There is a run $\theta(U)$ of the incremental parser such that $L(\theta(U)) = S$ and $k(\theta(U)) = n + 1$.*

Proof. To prove the lemma, I show that if the length of U is n then there is a run $\theta(U)$ such that for every $k \leq n$ there exists i_k such that $L(\theta_{i_k}(U)) = S_k$ and $k(\theta_{i_k}) = k + 1$. The proof is by induction on k. For $k = 1$ the claim is obvious since S_1 contains no links.

Assume now that the claim holds for i_k. By directional uniqueness (Lemma 5.1.25) there is at most one link of the form $x \xrightarrow{d} x_{k+1}$ ($x \in U_k$) in $S_{k+1} \setminus S_k$ and all other links in this set are of the form $x_{k+1} \xrightarrow{d} x$ ($x \in U_k$). Let $l_1, \ldots, l_m$ be the links in $S_{k+1} \setminus S_k$ ordered as follows. All links of the form $x_{k+1} \xrightarrow{d} x$ ($x \in U_k$) are ordered by increasing length. The (at most one) link $x \xrightarrow{d} x_{k+1}$ ($x \in U_k$) appears immediately after all links which are shorter or equal to it in length and have a smaller depth than it.

Because the adjacency condition for a link l_i does not depend on links l_j for $i < j$, it follows from Lemma 5.5.10 and Lemma 5.5.11 that adding the links $l_1, \ldots, l_m$ in this order adds each link when it is incrementally addable (the satisfaction of the equality condition in the definition of an incrementally addable link follows directly from the fact that S_{k+1} is a shortest common cover link set and the satisfaction of the non-covered and forcing conditions follows directly from the ordering chosen).

It remains to show that when a resolution violation is created, the links can still be added in the given order. Assume that after links $l_1, \ldots, l_i$ have been added there is a resolution violation and $\langle x, x_{k+1}, d_1, d_2 \rangle$ is the minimal resolution violation. This resolution violation must have been created by a link $l_r = x_{k+1} \xrightarrow{d_2} v$ for some $r \leq i$. Since the resolution violation does not exist in S_{k+1}, it must be repaired by one of the links $l_{i+1}, \ldots, l_m$.

Assume $l_{i+1} = w \xrightarrow{d} x_{k+1}$ ($w \in U_k$). If the resolution violation is repaired by l_{i+1}, then $w \in [x, x_{k+1})$ and if $w = x$ then $d = d_1$. Therefore, in this case l_{i+1} can be added by the incremental parser algorithm. Assume now that the resolution violation is repaired by some l_p with $i + 2 \leq p$. It follows that l_p must have depth d_2 and both its ends have to be in $[x, x_{k+1}]$. By the definition of the ordering, for every $j \leq i$, $l_j = x_{k+1} \xrightarrow{d_j} w_j$ with $d_j < d$. In particular this holds for l_r and therefore $d_2 < d$. Also by the definition of the ordering, for every $i + 2 \leq j$,

$l_j = x_{k+1} \xrightarrow{d_j} w_j$ such that either $d \leq d_j$ or $|l_{i+1}| < |l_j|$. Since $d_p = d_2 < d$ it follows that $|l_{i+1}| < |l_p|$ and therefore $w \in (x, x_{k+1})$, which implies that $l_{i+1} = w \xrightarrow{d} x_{k+1}$ can be added by the incremental parser algorithm.

Assume now that $l_{i+1} = x_{k+1} \xrightarrow{d} w$ ($w \in U_k$). Since the resolution violation is repaired by one of the links $l_{i+1}, \ldots, l_m$ and can only be repaired by a link with both ends in $[x, x_{k+1}]$ it follows from the definition of the ordering of the links that l_{i+1} has both its ends in $[x, x_{k+1}]$. It remains to show that $d = d_2$. If the resolution violation is repaired by a link $l_j = x_{k+1} \xrightarrow{d'} v$ ($i + 1 \leq j$) then $d' = d_2$. Since $r < i+1 \leq j$, it follows from the ordering of the links and monotonicity that $d = d_2$. If the resolution violation is repaired by a link $l_j = v \xrightarrow{d'} x_{k+1}$ ($i + 1 < j$) then by the definition of the ordering of the links, $d_2, d < d'$. This implies that $d_2 = d = 0$, as required. □

Together, these lemmas prove the correctness of the incremental parser.

5.5.17 (3.3.2). Theorem (incremental parser correctness). *The incremental parser, Algorithm 5.5.8, is well-defined and always outputs a shortest common cover link set. Moreover, it can output any common cover link set.*

Proof. This follows directly from Corollary 5.5.15 and Lemma 5.5.16 □

Time Complexity of the Incremental Parser

It is not difficult to see that with appropriate data structures it is possible to maintain the list of addable links in total time (for parsing the whole utterance) which is linear in the length of the utterance. This includes the extraction of the subset of addable links which may be added when there is a resolution violation. Because of directional uniqueness (Lemma 5.1.25), at most one link can enter a word on each side. This means that the total number of links in a shortest common cover link set cannot exceed $2n-2$, where n is the length of the utterance. Therefore, if there is a constant time oracle which selects at each step the link to be added to the parse, the parser runs in linear time in the length of the utterance.

At the same time, if a deterministic version of the parser needs to consider all possible addable links before selecting a link, the total number of links it may have to consider may be quadratic in the length of the utterance. If considering each link can be done in constant time (as it often can) the parser run-time complexity has an upper bound which is quadratic in the length of the utterance. In practice, links between consecutive words are often preferred (see section 6.3.2) and when such links are possible, other links are not considered. While this does not reduce the theoretical upper bound on the parser's run-time complexity, it does in practice considerably speed up the parser. Experiments (section 7.4.9) suggest that in practice, the run-time complexity of the parser can be close to linear in the length of the utterance.

Chapter 6
Learning and Parsing

The incremental parser Algorithm 3.3.1 is nondeterministic. It specifies a set of links which may be added at each step but does not indicate which of these links should be added (if at all). This is not surprising, since the algorithm is not language specific and different languages require different parsing decisions. To create a deterministic parser, further specification of the algorithm is needed. Part of this specification may be universal and common to all languages, while part of it must remain language specific. Given a set of example utterances from a language, it is then the task of the learning algorithm to induce the language specific part of the parsing algorithm.

To create a deterministic parser from the nondeterministic parser, parsing functions were introduced in chapter 3. These functions determine which link (if any) should be added at each step in the parse of an utterance. The present chapter describes a specific family of parsing functions and a learning algorithm which selects one of these functions. The properties common to all the functions in the family are the universal part of the algorithm while the properties specific to each function are the language specific component of the algorithm, which has to be learned.

The parsing function family is lexicalized. This means that any language specific information used by the parser to make parsing decisions must be associated with some word. This association is represented by a lexicon which is a function mapping every word to a lexical entry containing information relevant to the linking of that word with other words. When the parser needs to decide whether to link two words, it accesses the lexical entries of these two words and makes its linking decision based on the information found there. The difference between parsing functions in the family can therefore be expressed in terms of the lexicon function. It is the task of the learning algorithm to construct the lexicon.

Every lexical entry in the lexicon holds lists of labels. These labels describe the contexts in which the word appears and take over the role played by parts-of-speech in standard parsing algorithms. The calculation of the labels is local and, unlike most standard induction algorithms, does not require any clustering or global optimization. Parsing is then directly guided by these labels. Due to the Zipfian distribution of words, high frequency words dominate these lists of labels and parsing decisions for words of similar distribution are guided by the same labels. This method contrasts with standard induction algorithms which first cluster words into classes based on their most frequent neighbors (as in Schütze 1995; Clark 2000) and then use these induced classes for parsing. By avoiding parts-of-speech, these two steps are collapsed here into one. This not only greatly simplifies the induction process, but also allows much greater flexibility, since the exact label which is used at each parse step may depend on the parsing context. In addition, the labels on the left and right side of each word may remain independent.

Using the labels and the relations between them, basic linking properties are bootstrapped for each word. These properties describe how the word should be linked with other words when their labels match. This bootstrapping process is, again, local.

The parsing function family defines a way to convert the labels and linking properties stored for each word in the lexicon into parsing decisions. This part of the algorithm, common to all parsing functions in the family, is local and greedy. At each step, it calculates a weight for every link which the incremental parser may add at that step. The link with the largest non-zero weight is then added to the parse and the process is repeated. When all possible links are assigned a zero weight, the next word is read and parsing continues with this extended prefix of the utterance.

The first two sections in the chapter describe a general architecture for a lexicalized parsing function family with bootstrapping based learning and greedy parsing: section 6.1 defines the greedy parser and section 6.2 defines the learning process. One specific proposal for such a function family is given in section 6.3.

6.1 The Parsing Function Family

6.1.1 Greedy Parsing Functions

The parsing function family I will define in this section is a set of parsing functions (as defined in Definition 3.4.1). These parsing functions are greedy: at each step of the parse, they calculate a non-negative weight for each of the links which may possibly be added at that step and if there is a non-zero weight link, the link with the highest weight is added. Once all addable links are assigned a zero weight, no link is added by the parsing function and, by the definition of the incremental parser (Algorithm 3.4.2), the next word is added to the prefix and the process is repeated with the extended prefix. Formally, the weight function and the greedy parsing function are defined as follows:

6.1.1. Definition. [weight function] A function Wt is a *weight function* if its range is the non-negative real numbers and its domain is all possible tuples $\langle U_k, L, x \xrightarrow{d} y\rangle$ such that U_k is an utterance prefix, L is a set of links over U_k and $x \xrightarrow{d} y$ is addable to L over U_k (Definition 3.2.5).

6.1.2. Definition. [greedy parsing function] A function $\mathcal{P}$ is a *greedy parsing function* iff $\mathcal{P}$ is a parsing function (Definition 3.4.1) and there is a weight function Wt such that:

1. If $\mathcal{P}(U_k, L) = L$ then $\mathrm{Wt}(U_k, L, x \xrightarrow{d} y) = 0$ for all links $x \xrightarrow{d} y$ addable to L over U_k.
2. If $\mathcal{P}(U_k, L) = L \cup \{x \xrightarrow{d} y\}$ then $\mathrm{Wt}(U_k, L, x \xrightarrow{d} y) > \mathrm{Wt}(U_k, L, x' \xrightarrow{d'} y')$ for any link $x' \xrightarrow{d'} y' \neq x \xrightarrow{d} y$ addable to L over U_k.

This is a greedy parsing algorithm, optimizing every parse step rather than attempting to optimize the parse as a whole. This contrasts with many other unsupervised learning algorithms (such as Klein and Manning 2004; Bod 2006a; Bod 2006b) which assign every utterance the binary parse tree which maximizes some objective function (over binary trees). The function Wt has U_k and L as arguments, so it may attempt to optimize globally at least over all possible parses extending L over the prefix U_k. However, because the incrementality of the parser does not allow any changes to be made at this stage to links over U_{k-1}, the range of possibilities is very limited: the parser must be *prefix greedy*. The weight functions I actually consider in this chapter are even greedier than required by the definition because they calculate each weight locally based on the two ends of the link and almost without considering alternative addable links.

This greedy behavior is justified by the incremental nature of language processing by humans. Optimization over the full utterance is simply impossible if parsing decisions have to be made before the end of the utterance is processed. If the incrementality of the parser roughly resembles that of human processing, the incrementality of the parser should not in itself cause parsing errors except in those cases where humans are also likely to err. It is reasonable to assume that utterances which humans initially parse incorrectly (garden path sentences) are relatively infrequent because of the obvious problems in communication they create. In spoken language, garden path sentences are infrequent because most garden path effects can be corrected by appropriate intonation. In written language, writers are expected to correct garden path sentences. Whatever the frequency of garden path sentences, it seems that humans use different cognitive processes for initial parsing and for recovering from garden path parsing failures (the more difficult cases actually seem to require completely conscious analysis of the problematic sentence). This is also the approach I will take here, where the parser is not expected to correctly parse garden path sentences. It is assumed that some other mechanism (not specified here) is responsible for recovering from garden path errors, possibly by adding information to the prefix of the utterance. This enrichment of the prefix should be sufficient for the incremental parser to select the correct parse for the prefix. Applying the incremental parser to this enriched utterance should then result in the correct parse.

6.1.2 A Lexicalized Family of Parsing Functions

Depending on the weight function used, different greedy parsing functions can be defined. The assumption I make here is that any language can be parsed by a greedy parsing function, but that different languages require different weight functions. Given examples of the target language, a learning algorithm must be defined to select the weight function which is to be used to parse that language. Not all possible weight function need to be considered, of course, and the set of weight functions from which the learning algorithm can choose defines a *family* of parsing functions. There may be much in common between the different weight functions in a family of parsing function. To capture this, I define a parsing function family as being based on a single parameterized weight function, $\mathrm{Wt}(\theta)$. Each value of the parameter θ then defines a parsing function in the family.

In the present chapter I use lexicon functions as the parameter θ. A lexicon is a function which assigns every word x a lexical entry (using algorithmic terminology I will say that the lexicon *stores* for each word x a lexical entry). Each such lexical entry is a sequence of adjacency points, holding statistics relevant to the decision whether to link x to some other word when that word becomes adjacent to x. These statistics are given as strengths assigned to labels and linking properties. Multiple adjacency points are needed for each word because a word may have more than one link and each of these links may have different properties. This is similar to the specification of the arguments of a word in standard linguistic theories, but the adjacency points may differ from standard arguments, reflecting the differences between shortest common cover link sets and the standard dependency structure.

The Lexicon

Let W be the set of words in the corpus. The set of *labels* $L(W) = W \times \{0, 1\}$ consists of two labels based on every word w: a *class label* $(w, 0)$ (denoted by $[w]$) and an *adjacency label* $(w, 1)$ (denoted by $[w\_]$ or $[\_w]$). The two labels $(w, 0)$ and $(w, 1)$ are said to be *opposite labels* and for $l \in L(W)$ I write l^{-1} for the opposite of l. In addition to the labels, there is also a finite set P of *linking properties* $(P \cap L(W) = \emptyset)$. This set of properties is fixed for each version of the algorithm and will be specified below. It typically consists of elements such as *In* and *Out* describing inbound and outbound link strengths.

An *adjacency point* is a function $A : L(W) \cup P \to \mathbb{R}$ which assigns each label in $L(W)$ and each linking property in P a real valued *strength*. For each A, $\#(A)$ is the *count* of the adjacency point - the number of times the adjacency point was updated (see below). Based on this count, I also define a normalized version of A: $\bar{A}(l) = A(l)/\#(A)$.

A *lexicon* $\mathcal{L}$ is a function which assigns each word $w \in W$ a lexical entry $(\ldots, A^w_{-2}, A^w_{-1}, A^w_1, A^w_2, \ldots)$. Each of the A^w_i is an adjacency point for the *adjacency position* i (relative to w). A negative index i indicates an adjacency position to the left of w while a positive index indicates an adjacency position to the right of w. As the absolute value of the index i increases, A^w_i describes adjacencies further away from w, with A^w_{-1} and A^w_1 describing the *direct adjacencies*.

Lexicalized Greedy Parsing Function Family

Having defined the lexicon, it is now possible to define the parsing function family which will be used for the rest of this chapter:

6.1.3. Definition. [lexicalized greedy parsing function family] A set $\mathcal{PF}$ of greedy parsing functions is a lexicalized family if there is a function Wt such that for every parsing function $\mathcal{P} \in \mathcal{PF}$ there is a lexicon $\mathcal{L}$ such that $\mathrm{Wt}(\mathcal{L})$ is the weight function defining $\mathcal{P}$.

Since each of the functions in $\mathcal{PF}$ is uniquely defined by a lexicon function $\mathcal{L}$, I will write $\mathcal{P}_{\mathcal{L}}$ for a specific parsing function in the family $\mathcal{PF}$. Because it is often more natural to refer to $\mathcal{PF}$ in algorithmic terms, I will simply call it *the parser* and a specific parsing function $\mathcal{P}_{\mathcal{L}}$ will then simply be *the parser using lexicon* $\mathcal{L}$.

Of course, as long as the function Wt is not specified, the relation between the lexicon $\mathcal{L}$ and the weight function it defines, $\mathrm{Wt}(\mathcal{L})$, can be quite arbitrary. For the definition to be useful, however, the function $\mathrm{Wt}(\mathcal{L})$ has to be closely related to the function $\mathcal{L}$ it is based on. A specific choice for the function Wt will be defined in section 6.3.2. For the definition of this function to make any sense it is first necessary to consider the learning process which constructs the lexicon.

6.2 The Learning Process

Given a sequence of training utterances $(U_t)_{0\leq t}$, the *learner* constructs a sequence of lexicons $(\mathcal{L}_s)_{0\leq s}$ beginning with the zero lexicon $\mathcal{L}_0$ (which assigns a zero strength to all labels and linking properties). At each step, the learner uses the parsing function $\mathcal{P}_{\mathcal{L}_s}$ based on the previously learned lexicon $\mathcal{L}_s$ to extend the parse L of an utterance U_t. It then uses the result of this parse step (together with the lexicon $\mathcal{L}_s$) to create a new lexicon $\mathcal{L}_{s+1}$ (it may be that $\mathcal{L}_s = \mathcal{L}_{s+1}$). This operation is a *lexicon update*. The process then continues with the new lexicon $\mathcal{L}_{s+1}$. Any of the lexicons $\mathcal{L}_s$ constructed by the learner may be used for parsing any utterance U, but as s increases, parsing accuracy should improve. This learning process is open-ended: additional training text can always be added without having to re-run the learner on previous training data.

6.2.1 Lexicon Update

The incremental parser Algorithm 3.3.1 may only add a link from x to y when y is adjacent to x (Definition 3.2.1). This adjacency must, by definition, be unused (that is, there should not already be a link from x to y). The lexical entry of x should therefore hold statistics about the words that appear in unused adjacencies of x. When parsing, comparison of these statistics with the word y which actually appears in an unused adjacency of x determines whether a link should be created from x to y.

Empty Adjacencies

The adjacency relation was defined in Definition 3.2.1 as a relation between two positions in an utterance. In the context of parsing this is sufficient, because links can only be created between such positions. For learning, however, adjacency should also cover empty positions, that is, positions where no word appears. For example, if a word appears at the beginning of a sentence, it is adjacent to the left edge of the sentence, an empty position. It therefore does not have a link connecting it to the left in this sentence. If this happens often, it may indicate that also when the word appears in the middle of the sentence it should not have a link to the word to its left. In general, empty adjacencies are identical in all respects to normal adjacencies except for the absence of a word at the adjacency position. As a result, if a certain adjacency of a word is often empty this is a strong indication that also when the adjacency is not empty no link should be created. This observation is at the heart of the whole learning algorithm (compare this with the remarks in Klein and Manning 2004 (and Collins 1999 cited there) concerning the importance of including termination probabilities in dependency parsing models).

Empty adjacencies differ from standard adjacencies only in that there is no word in the adjacent position. This happens when the adjacent position falls outside the utterance (at the beginning or end of the utterance) or when punctuation blocks access to the adjacent position (indicating that a link cannot be created between the two words). I leave it open for now which adjacencies are blocked by which punctuation, as this remains to be specified for each different parsing function family (and may even be language specific).

6.2.1. Definition. [empty adjacency] Let L be a set of common cover links of depth 0 or 1 over an utterance $U = \langle x_1, \ldots, x_n \rangle$. Let $U' = \langle \emptyset_l, x_1, \ldots, x_n, \emptyset_r \rangle$. A word $x \in U$ has an empty adjacency of depth $d \leq 1$ relative to L over U if one of the following holds:

1. Over U', $x \dashv_d^L \emptyset_l$ or $x \dashv_d^L \emptyset_r$.

2. Over U, $x \dashv_d^L y$ and the adjacency is blocked by punctuation.

Empty adjacencies are well-defined because any common cover link set L over U is also a common cover link set over U'.

Notation To distinguish between the different empty adjacencies, the following notation is used. The obvious notation $x \dashv_d^L \emptyset_l$ and $x \dashv_d^L \emptyset_r$ is used to indicate the empty adjacencies at the left and right edges of the utterance (respectively). Empty adjacencies resulting from punctuation are written as $x \dashv_d^L \emptyset_y^p$ where y is the word at the adjacency position and p is the blocking punctuation symbol. The symbols $\emptyset_l$, $\emptyset_r$ and $\emptyset_y^p$ are the *empty symbols*.

Update Positions

To collect statistics relevant to the decision whether a word x should be linked to a word y which is adjacent to x, the lexical entry of x is updated by α's which are adjacent to x, where these α's are either words or empty symbols. This section describes the algorithm which determines which positions (empty or not) in an utterance containing x are used to update x.

To understand which adjacent positions should be used to update the statistics of a word x, it is necessary to understand the way in which adjacencies change as links are added to the parse. By Lemma 3.2.4, at each parse step, x may have multiple adjacencies, but has at most one unused adjacency on each side. Because of the incrementality of the parser, only one side of each word is active at each step. Therefore, at each parse step, at most one unused adjacency of each word may be used to add a link. However, in the course of the parse of a single utterance, a word x may have several different words y as unused adjacencies. Which of these should be used to update the lexical entry of x?

Let us examine how the unused adjacencies of x change in the course of parsing. The unused adjacency of x can change either by the addition of a link from x to its unused adjacency y or by the addition of another link.

1. When a link is added from x to y, y remains adjacent to x but the adjacency is not unused anymore. Moreover, y also remains adjacent to x in all subsequent steps of the parse (see Lemma 3.2.4). In this case I will say that the unused adjacency was *used*. When an adjacency is used, a new unused adjacency may be created (no new unused adjacency is created only when it is blocked by *blocking* as in Definition 3.2.1).

2. When a link other than a link from x to y is added, the unused adjacency y of x may (but does not have to) be replaced by a new unused adjacency y'. In this case y is no longer adjacent to x and therefore I will say that the unused adjacency was *moved*. No other adjacency of x can change in case the link being added is not from x to y.

Consider the following example. The adjacencies of the word x_1 are $\emptyset_l$, x_2 and x_3 while the adjacencies of x_2 are x_1 and x_3:

$$\begin{tikzcd} x_1 \arrow[r, "0"] & x_2 & x_3 \end{tikzcd}$$

A link may now be added from x_2 to x_3 and the parser may then read the next word, x_4:

$$\begin{tikzcd} x_1 \arrow[r, "0"] & x_2 \arrow[r, "0"] & x_3 & x_4 \end{tikzcd}$$

The adjacency x_3 of x_2 becomes a used adjacency and x_4 becomes a new unused adjacency of x_2. The word x_1 remains an adjacency of x_2 (x_1 is an unused adjacency of x_2 but because of the incrementality of the parser, this adjacency cannot be used anymore and will remain unchanged in all subsequent parse steps). The addition of a link from x_2 to x_3 also causes the unused adjacency of x_1 to move from x_3 to x_4. After this change the adjacencies of x_1 are $\emptyset_l$, x_2 and x_4.

If an unused adjacency of x should be used and a link should be created then the lexical entry should be updated by words which typically appear in the position to be attached, that is, the last position the adjacency was moved to. If the unused adjacency should remain unused, then it will often be moved until it becomes an empty adjacency (which cannot be moved anymore). Either way, to properly describe the linking behavior of x, the lexical entry of x must be updated by the last position each unused adjacency of x was moved to. A simple way to determine whether an unused adjacency has stopped moving is to wait until the full utterance has been parsed and then update each word x by its adjacency positions relative to that final parse.

Notation Given an utterance U, a word $x \in U$ and a shortest common cover link set L over U, the positions adjacent to x relative to L can be divided into two sets: adjacency positions to the left of x and adjacency positions to the right of x. Each of these sets of adjacency positions can be ordered by increasing distance from x. I write $\left(\alpha_i^{L,U}(x)\right)_{i=(-1)}^{(-m)}$ and $\left(\alpha_i^{L,U}(x)\right)_{i=1}^{n}$ for the (possibly empty) symbols appearing at the adjacency positions to the left and right (respectively) of x (relative to L over U). The index i, which I refer to as the *adjacency index* increases in absolute value with the distance of the adjacency position from x. When no confusion is possible, I may drop U, L or x from this notation.

Using this notation, the update algorithm, which formalizes the discussion above, can be formulated. It updates the symbol at adjacency position $\alpha_i^{L,U}(x)$ on the adjacency point A_i^x. Such an update specifies the values of A_i^x in the new lexicon $\mathcal{L}_{s+1}$ based on its values in the current lexicon, $\mathcal{L}_s$ (see section 6.3.1 for details). Any value not updated remains unchanged from $\mathcal{L}_s$ to $\mathcal{L}_{s+1}$.

6.2.2. Algorithm. [lexicon update algorithm] Given an utterance U and a lexicon $\mathcal{L}_s$, use $\mathcal{L}_s$ to parse U. When the parser has produced a final parse L, update the lexicon $\mathcal{L}_s$ as follows: for every $x \in U$ and every adjacency index i of x (relative to L over U), update A_i^x (in $\mathcal{L}_s$) by the symbol $\alpha_i^{L,U}(x)$.

An important property of adjacency (given in Lemma 3.2.4) is that the first adjacency position of each word is always the position immediately preceding or following that word in the utterance. As this is entirely independent of the links assigned by the parser, it implies that the symbols by which the direct adjacency points $A_{(-1)}^x$ and A_1^x are updated are entirely independent of $\mathcal{L}_s$.

The following example should clarify the picture. Assume that the parser is parsing the sentence *give the boy this book.* Assume that having read the first three words, the parser (based on $\mathcal{L}_s$) assigns the following (correct) link structure and reads the next word *this*:

$$\begin{tikzcd}
\text{give} \arrow[r, "0"] & \text{the} \arrow[r, leftrightarrow, "0"] & \text{boy} & \text{this}
\end{tikzcd}$$

As observed above, regardless of the links assigned by the parser, $A_{(-1)}^{give}$ is updated by $\emptyset_l$, A_1^{give} is updated by *the*, $A_{(-1)}^{the}$ is updated by *give*, etc. The update of A_2^{give} now depends on the way in which the parser will decide to attach the determiner *this* to the prefix. If a link is added from *boy* to *this* or from *the* to *this* then *this* will not be adjacent to *give* in the final parse of the utterance:

$$\begin{tikzcd}
\text{give} \arrow[r, "0"] & \text{the} \arrow[r, leftrightarrow, "0"] \arrow[rr, dotted, bend left] & \text{boy} \arrow[r, dotted] & \text{this}
\end{tikzcd}$$

However, if both these links (from *the* and *boy* to *this*) receive a zero weight from the weight function (as will turn out to be the case) then *this* will be adjacent to *give* in the final parse (this does not depend, for example, on whether a link is created from *give* to *this*). Therefore, A_2^{give} will be updated by *this*, which is indeed the second argument of the verb.

One possible problem with the update algorithm is that the adjacency positions used to update entries in a lexicon $\mathcal{L}_s$ depend on parsing decisions based on entries in $\mathcal{L}_s$. Such a process may lead to the perpetuation of errors found in the lexicon $\mathcal{L}_s$. It seems, however, that under reasonable assumptions about the structure of language this does not happen. The assumption needed for this to hold is that, sufficiently often, syntactic structure is simple. More precisely, the *syntactic simplicity assumption* is:

6.2.3. Assumption. [syntactic simplicity] In the correct parses of a language, two consecutive adjacency positions of a word x are separated sufficiently often by a non-branching structure, that is, one in which all links are between consecutive words.

Having discovered one adjacency position of a word x, the algorithm correctly identifies the next adjacency position of x if it correctly constructs the links between the words separating the two adjacency positions. As will be seen later, if all these links are between consecutive words then their weights only depend on direct adjacency points (entries of the form $A^y_{(-1)}$ and A^y_1). Therefore, if the learning algorithm correctly learns the direct adjacency points, it will also use the correct adjacency positions to update x. Since, as mentioned above, the adjacency positions by which the direct adjacency points are updated do not depend on the link structure assigned by the parser, there is no inherent problem with the update algorithm.

This argument is in no way a proof that x is actually updated by the correct adjacency position. To actually prove this, we would have to show that the parser can indeed deduce the correct links from the direct adjacency points. That the position used in updating the direct adjacency point does not depend on the link structure is a necessary but not a sufficient condition for this to hold. Moreover, the term "sufficiently often" used in the definition of the syntactic simplicity condition remains vague because its exact definition (how often is sufficient) depends on other details of the learning algorithm and, even more crucially, on properties of natural languages. Of course, one can prove the correctness of a learning algorithm given certain assumptions about natural languages, but as long as we do not have a sufficiently accurate theory of natural languages, the correctness of a learning algorithm remains an empirical question. What the argument above does show, however, is that there is no inherent problem with the update algorithm using the parser to determine the update positions. For now, this will have to do.

6.2.2 Prefix Independence

The problem raised at the end of the previous section for the update algorithm is a special case of a general problem. The learning process makes use of the lexicon $\mathcal{L}_s$ to construct the next lexicon $\mathcal{L}_{s+1}$. This allows bootstrapping, where properties already learned are used to induce additional properties. For example, the second argument of a verb can only be correctly identified when the parser is already able to identify and construct the first argument of the verb. Because of such examples, it is certainly sensible and perhaps even necessary for the learning process to make use of information already learned in previous steps.

A seemingly fundamental problem with this approach is that it is difficult to determine whether any parameter being learned has already stabilized to its correct value or whether additional learning is necessary for such convergence. If a parameter p_1 is used to learn another parameter p_2 before p_1 has stabilized to its correct value, this may result in incorrect learning for p_2. The problem is made worse if the parameter p_1 is used to learn p_1 itself (this can happen directly or, more commonly, indirectly, where p_1 is used in learning p_2 and p_2 is used in learning p_1). I will refer to this as the *positive feedback problem* where an initial incorrect value learned for p_1 is used by the learning algorithm to reinforce that incorrect value, thus making it impossible to recover from an initial error.

To avoid these problems, the learning algorithm should be able to recover from any initial error given sufficiently many additional examples. This also means that if the language properties change during the learning process, the learning algorithm can adapt to these changes (given sufficiently many examples). Since in the model presented here learning and parsing happen together and continuously, the learning algorithm can adapt to changes in the language being processed. The number of examples needed for such an adaptation depends on how strong the evidence for the initial setting of the parameters is.

To formalize these intuitions, I give here a simple condition which describes when bootstrapping is correctly performed and that ensures that a learning algorithm does not have the positive feedback problem. In subsequent sections, where the details of the learning algorithm are given, it will be easy to see that this condition indeed holds.

Let p be a parameter calculated by a learning algorithm ϕ. For every finite sequence of examples $e_1, \ldots, e_n$ the learning algorithm receives as input, the algorithm outputs a value for p. I write $\phi_p(e_1, \ldots, e_n)$ for this value and will assume it to be a real number.

6.2.4. Definition. [prefix independence] A learning algorithm ϕ is *prefix independent* for parameter p iff for any sequences of examples $e'_1, \ldots, e'_k$ and $(e_i)_{1 \leq i}$, $\lim_{n\to\infty} |\phi_p(e'_1, \ldots, e'_k, e_1, \ldots, e_n) - \phi_p(e_1, \ldots, e_n)| = 0$. The algorithm ϕ is *prefix independent* iff it is prefix independent for all its parameters.

The prefix independence condition is simple. It requires that whatever decision the learning algorithm reaches based on some initial set of examples can be undone by observing sufficiently many additional examples. This ensures that the learning algorithm does not have to know whether a certain learned parameter p_1 has stabilized to its correct value before it can use it in learning another parameter p_2. The algorithm can simply use the value of p_1 in every learning step, whether this value has already stabilized to its correct value or not. Once the value of p_1 has stabilized to its correct value, prefix independence ensures that sooner or later the value for p_2 induced by the learning algorithm will be sufficiently close to the value induced by the algorithm had its estimate for the value of p_1 been correct (in some miraculous way) from the very beginning.

The prefix independence condition also rules out the positive feedback problem. If $e_1, \ldots, e_k$, $e'_1, \ldots, e'_k$ and $(e_i)_{k+1 \leq i}$ are sequences of examples then prefix independence implies that $|\phi_p(e_1, \ldots, e_k, e_{k+1}, \ldots, e_n) - \phi_p(e'_1, \ldots, e'_k, e_{k+1}, \ldots, e_n)| = 0$. This means that two sequences of examples which differ only in some initial examples are equivalent in terms of learning. When a learning algorithm has positive feedback, this property is violated. If two different choices of initial examples $e_1, \ldots, e_k$ and $e'_1, \ldots, e'_k$ lead to different values for ϕ_p then these two values continue to be reinforced by subsequent examples and the difference does not converge to 0. Therefore, as long as prefix independence can be shown to hold, the algorithm cannot have positive feedback.

Actually proving that prefix independence holds may be anything from trivial to very difficult, depending on the specific algorithm (the most difficult cases are those where values of several parameters are each used in learning all the others). In the learning algorithms I present here, the proof of prefix independence is usually very simple (except for the adjacency update position which was discussed in the previous section). In fact, the algorithms were designed with prefix independence in mind and only algorithms for which prefix independence was easily seen to hold were considered. Interestingly, when prefix independence was violated in the course of development, this was immediately evident in the results of the experiments.

6.3 Basic Parsing Function Family

The previous sections outlined a general framework for a lexicalized parsing function family with bootstrapping based learning and greedy parsing. For a complete unsupervised parsing algorithm, two components still have to be specified. The first is the adjacency point update algorithm, that is, what happens when the lexicon update algorithm (Algorithm 6.2.2) updates the adjacency point A_i^x by a symbol α. The second component to be specified is the weight function $\mathrm{Wt}(\mathcal{L}, U_k, L, x \xrightarrow{d} y)$. One specific proposal for these two components, which I will call *the basic parsing function family*, is specified in this section.

6.3.1 Adjacency Point Update

The basic parsing function family uses a set of four linking properties $P = \{Stop, In^*, In, Out\}$. Roughly, for each side of each word, $Stop$ specifies the strength of non-attachment, In and Out specify the strength of inbound and outbound links and In^* is an intermediate value in the induction of inbound and outbound strengths. The update of A_i^x by a symbol α is given by operations $A_i^x(p) \mathrel{+}= f(A_{(-1)}^{\alpha}, A_1^{\alpha})$ which make the value of $A_i^x(p)$ in the new lexicon $\mathcal{L}_{s+1}$

equal to the sum $A_i^x(p) + f(A_{(-1)}^\alpha, A_1^\alpha)$ in the old lexicon $\mathcal{L}_s$. Two basic functions are used in describing the algorithm:

$$Sign(i) = \begin{cases} 1 & \text{if } 0 < i \\ -1 & \text{otherwise} \end{cases}$$

$$_\bullet A_i^\alpha = \begin{cases} true & \text{if } \nexists l \in L(W) \ : \ A_i^\alpha(l) > A_i^\alpha(Stop) \\ false & \text{otherwise} \end{cases}$$

The update algorithm uses these function and the *normalized* version $\bar{A}$ of the adjacency points to update A_i^x:

6.3.1. Algorithm. [adjacency point update] Update A_i^x by symbol α as follows:

- Increment the count of the adjacency point:

$$\#(A_i^x) \mathrel{+}= 1$$

- If α is an empty symbol or if x and α are words separated by stopping punctuation (full stop, question mark, exclamation mark, semicolon, comma or dash):

$$A_i^x(Stop) \mathrel{+}= 1$$

- Otherwise:
  - for every $l \in L(W)$:

$$A_i^x(l^{-1}) \mathrel{+}= \begin{cases} 1 & \text{if } l = [\alpha] \\ \bar{A}_{Sign(-i)}^\alpha(l) & \text{otherwise} \end{cases}$$

    (In practice, only $l = [\alpha]$ and the 10 strongest labels in $A_{Sign(-i)}^\alpha$ are updated. Because of the exponential decay in the strength of labels in $A_{Sign(-i)}^\alpha$, this is a good approximation.)
  - If $i = -1, 1$ perform the following bootstrapping:

$$A_i^x(In^*) \mathrel{+}= \begin{cases} -1 & \text{if } _\bullet A_{Sign(-i)}^\alpha \\ +1 & \text{if } \neg _\bullet A_{Sign(-i)}^\alpha \wedge {_\bullet A_{Sign(i)}^\alpha} \\ 0 & \text{otherwise} \end{cases}$$

$$A_i^x(Out) \mathrel{+}= \bar{A}_{Sign(-i)}^\alpha(In^*)$$

$$A_i^x(In) \mathrel{+}= \bar{A}_{Sign(-i)}^\alpha(Out)$$

Discussion

To understand the way the labels and properties are calculated, it is best to look at some examples. The following table gives the linking properties and strongest labels for the determiner *the* as learned from the complete Wall Street Journal corpus (only $A^{the}_{(-1)}$ and A^{the}_{1} are shown):

the

A_{-1}		A_1	
Stop	12897	*Stop*	8
*In**	14898	*In**	18914
In	8625	*In*	4764
Out	-13184	*Out*	21922
[the]	10673	[the]	16461
[of_]	6871	[a]	3107
[in_]	5520	[_the]	2787
[a]	3407	[of]	2347
[for_]	2572	[_company]	2094
[to_]	2094	['s]	1686
[on_]	2009	[in]	1388
[that_]	1495	[_U.S.]	1199
[and_]	1489	[and]	1129
[at_]	1149	[to]	876

A strong class label [w] indicates that the word w frequently appears in contexts which are similar to *the*. A strong adjacency label [w_] (or [_w]) indicates that w either frequently appears next to *the* or that w frequently appears in the same contexts as words which appear next to *the*.

Most of the labels which appear in the entry of *the* are not surprising. Both on the left and on the right, *the* has both [*the*] and [*a*] as class labels. This is not surprising because *the* and *a* are the most frequent determiners and it is only natural that they label words which appear in determiner contexts. On the left, *the* is also labeled by the adjacency labels based on several prepositions: [of_], [*in*_], [for_], etc. This is not surprising and it should only be pointed out that all labels are based on very frequent words.

On the right side of *the*, the labels are somewhat less expected. Two of the labels, [_*company*] and [_*U.S.*] are simply nouns which are common in the corpus the parser has been trained on. When training on another corpus, these labels will disappear, possibly being replaced by other frequent nouns. This simply reflects the fact that while closed class words (such as determiners and prepositions) are frequent because of the syntax of English, open class words (such as nouns) are frequent in a corpus because of its semantic content. This means that closed class based labels will be more stable across different corpora than open class labels.

More surprising on the right side of *the* is the label [_*the*]. At first sight this may suggest that the word *the* is often followed in the corpus by the word *the*. This is, of course, not the case. The reason the label [_*the*] appears on the right side of *the* is that the word *the* is often followed by nouns or adjectives which may also appear without a determiner. When a noun or adjective appears without a determiner, it appears in a position which is often taken by a determiner. This noun or adjective is therefore labeled (on the left) by the class label [*the*] which, in turn, labels *the* with [_*the*]. Is this label incorrect? That depends on how we use it to reach parsing decisions. In the next section we will see that because the label [*the*] is far more frequent than [_*the*], it will dominate parsing decision. Therefore, *the* will be attached to the word following it as if it and not the following word is the determiner.

The property *Stop* counts the number of times a boundary appeared next to *the*. Because *the* can often appear at the beginning of an utterance but must be followed by a noun or an adjective, it is not surprising that *Stop* is stronger than any label on the left but weaker than all labels on the right. In general, it is unlikely that a word has an outbound link on the side on which its *Stop* strength is stronger than that of any label. The opposite is not true: a label stronger than *Stop* indicates an attachment but this may also be the result of an inbound link, as in the following entry for *to*, where the strong labels on the left are a result of an inbound link:

to

A_{-1}		A_1	
Stop	822	*Stop*	48
*In**	-4250	*In**	-981
In	-57	*In*	-1791
Out	-3053	*Out*	4010
[to]	5912	[to]	7009
[%_]	848	[_the]	3851
[in]	844	[_be]	2208
[the]	813	[will]	1414
[of]	624	[_a]	1158
[a]	599	[the]	954
[according_]	573	[of]	889
[for]	551	[would]	717
[expected_]	435	[n't]	604
[and]	346	[and]	546

For this reason, the learning process is based on the property ${}_{\bullet}A_i^x$ which indicates where a link is *not possible*. Since an outbound link on one word is inbound on the other, the inbound/outbound properties of each word are then calculated by a simple bootstrapping process as an average of the opposite properties of the neighboring words.

The update of the first property, In^*, is based on the assumption that if ${}_\bullet A_i^\alpha$ often holds (on the adjacent word α) then it is unlikely that the updated word x has an inbound link. More precisely, the normalized value of the first property In^* is $\bar{A}_i^x(In^*) = P(\neg {}_\bullet A_{Sign(-i)}^\alpha \wedge {}_\bullet A_{Sign(i)}^\alpha) - P({}_\bullet A_{Sign(-i)}^\alpha)$ where the probability space is uniform over all *words* α which appear adjacent to x (that is, empty adjacencies are excluded). The two events ${}_\bullet A_{Sign(-i)}^\alpha$ and $\neg {}_\bullet A_{Sign(-i)}^\alpha \wedge {}_\bullet A_{Sign(i)}^\alpha$ are disjoint and their union is ${}_\bullet A_{(-1)}^\alpha \vee {}_\bullet A_1^\alpha$. Therefore, $\bar{A}_i^x(In^*)$ is negative if over all words α adjacent to x which have a *Stop* stronger than any other label this happens more often on the side of α which faces (may be attached to) x. Assuming that the frequency of this property due to noise is equal on both sides, a negative value of $\bar{A}_i^x(In^*)$ indicates that no inbound link is likely, while a positive value indicates that such a link is likely. As will be seen below, the linking properties are only used when there is already an indication that a link should be created, so the difference between a negative an a positive value here should only be interpreted as conditional on there being some link between the words.

The value learned for $\bar{A}_i^x(In^*)$ is, of course, only a first approximation. Since an inbound link for one word is an outbound link for the adjacent word, $\bar{A}_i^x(Out)$ is the average value of $\bar{A}_i^\alpha(In^*)$ on adjacent words α and $\bar{A}_i^x(In)$ is a similar average over $\bar{A}_i^\alpha(Out)$. If the value of $\bar{A}_i^x(In^*)$ is correct for sufficiently many x's, such averaging helps correct errors for those words where errors did occur in the first step.

The linking properties are only calculated for the direct adjacency positions $A_{(-1)}^x$ and A_1^x. The reason for this will become clear in the next section.

6.3.2 The Weight Function

To simplify notations, I will assume from now on that the lexicon $\mathcal{L}$, the prefix U_k and the set of links L are fixed and will therefore simply write $\mathrm{Wt}(x \xrightarrow{d} y)$ for $\mathrm{Wt}(\mathcal{L}, U_k, L, x \xrightarrow{d} y)$. At each step in the parse, the weight function $\mathrm{Wt}(x \xrightarrow{d} y)$ must assign a non-negative weight to each link $x \xrightarrow{d} y$ which may be added by the incremental parser, Algorithm 3.3.1, to the set of links L over the utterance prefix $U_k = \langle x_1, \ldots, x_k \rangle$. The weight could be assigned directly based on the *In* and *Out* linking properties of the lexical entries of x and y, but this method is not satisfactory for several reasons:

- The values of the linking properties on low frequency words are not reliable, as they are based on very few examples.
- The values of these properties on x and y may conflict. For example, should the value of *Out* on x or *In* on y be used for the link $x \xrightarrow{d} y$?

- Some words are ambiguous and require different linking in different contexts. Using the In and Out properties of a word directly would result in the word having the same links in all contexts.

To solve these problems, the weight of the link $x \xrightarrow{d} y$ is based on the linking properties of the best matching label between x and y. This label represents the relation between x and y and is usually a frequent word with reliable statistics. Moreover, when one of the words among x and y is far more frequent than the other, it is the properties of the frequent word which dominate the weight function (because the frequent word will be a label of the infrequent word, but not the other way around). This solves many of the problems related to infrequent words and seems to make smoothing unnecessary.

The use of the best matching label also explains why the learning algorithm only calculates the linking properties for the direct adjacency points $A^x_{(-1)}$ and A^x_1. When a link $x \xrightarrow{d} y$ may be added to the parse, y must be adjacent to x relative to L (the links already constructed by the parser). This means that there is a path of links all the way from x to the word directly adjacent to y. We can therefore see x and all words separating it from y as one unit headed by x which the parser may attach to y. Whether such an attachment should take place is determined in two steps. First, the appropriate adjacency point A^x_i of x is matched with $A^y_{Sign(-i)}$ to determine the best matching label. Here, a different index i may be used for each attachment of x, to reflect the different arguments a word may have. Each such match may result in a different best matching label (or in no matching label at all). For y, the direct adjacency $A^y_{Sign(-i)}$ is used because, taken as a whole, the unit headed by x which is considered for attachment to y is directly adjacent to y. Having determined the best matching label between the unit headed by x and y, the linking properties of the best matching label can be seen to represent the relation between the full unit headed by x and y. Since these two are directly adjacent, only the direct adjacency points of the matching label are used. For this reason, only the linking properties of direct adjacency need to be learned.

Preference for Direct Adjacency

The weight function has a preference for attaching directly adjacent (that is, consecutive) words before attaching more distant words. Formally, this is defined as follows:

- If x_0 and y_0 are directly adjacent, $x_0 \xrightarrow{d_0} y_0$ is addable to L over U_k and $\mathrm{Wt}(x_0 \xrightarrow{d_0} y_0) > 0$ then for any x and y which are not directly adjacent, $\mathrm{Wt}(x \xrightarrow{d} y) = 0$.

This is the only part of the weight function which is not entirely local and depends not only on the lexical entries of the two words x and y but also on the weight of other links.

Calculating the Best Matching Label

A label l is a *matching label* between A^x_i and $A^y_{Sign(-i)}$ if $A^x_i(l) > A^x_i(Stop)$ and either $l = (y, 1)$ or $A^y_{Sign(-i)}(l^{-1}) > 0$. The *match strength* of l is:

$$s(l) = \begin{cases} \bar{A}^x_i(l) & \text{if } l = (y,1) \\ \min(\bar{A}^x_i(l), \bar{A}^y_{Sign(-i)}(l^{-1})) & \text{otherwise} \end{cases}$$

The *best matching label* at A^x_i is the matching label l with the maximal match strength (in practice, as before, only the top 10 labels in A^x_i and $A^y_{Sign(-i)}$ are considered).

The *best matching label from x to y* is calculated between A^x_i and $A^y_{Sign(-i)}$ such that A^x_i is on the same side of x as y and was either already used to create a link or is the first adjacency point on that side of x which was not yet used. This means that the adjacency points on each side have to be used one by one, but may be used more than once. The reason is that optional arguments of x usually do not have an adjacency point of their own but have the same labels as obligatory arguments of x and can share their adjacency point. The A^x_i with the strongest matching label is selected, with a preference for the unused adjacency point (that is, if the unused adjacency point produces a best matching label, this label is taken). As in the learning process, label matching is blocked between words which are separated by stopping punctuation.

Calculating the Link Weight

The best matching label $l = (w, \delta)$ from x to y can be either a class ($\delta = 0$) or an adjacency ($\delta = 1$) label at A^x_i. If it is a class label, w can be seen as taking the place of x and all words separating it from y (which are already linked to x). If l is an adjacency label, w can be seen to take the place of y. The calculation of the weight $\mathrm{Wt}(x \xrightarrow{d} y)$ of the link from x to y is therefore based on the strengths of the linking properties of A^w_σ where $\sigma = Sign(i)$ if $l = (w, 0)$ and $\sigma = Sign(-i)$ if $l = (w, 1)$. In addition, the weight is bounded from above by the best label match strength, $s(l)$. This is because a weak match means that the label is weak either on x or on y and therefore the resulting link weight should not be high.

- If $l = (w, 0)$ and $A^w_\sigma(Out) > 0$:

$$\mathrm{Wt}(x \xrightarrow{0} y) = \min(s(l), \bar{A}^w_\sigma(Out))$$

- If $l = (w, 1)$:
  - If $A^w_\sigma(In) > 0$:

    $$\mathrm{Wt}(x \xrightarrow{d} y) = \min(s(l), \bar{A}^w_\sigma(In))$$

    where if $A^w_\sigma(In^*) < 0$ and $A^w_\sigma(Out) \leq 0$ then $d = 1$ and otherwise $d = 0$.
  - Otherwise, if $|A^w_\sigma(In^*)| \geq |A^w_\sigma(In)|$ and $A^w_\sigma(In^*) > 0$:

    $$\mathrm{Wt}(x \xrightarrow{0} y) = \min(s(l), \bar{A}^w_\sigma(In^*))$$

- If $A^w_\sigma(Out) \leq 0$ and $A^w_\sigma(In) \leq 0$ and either $l = (w, 1)$ or $A^w_\sigma(Out) = 0$:

  $$\mathrm{Wt}(x \xrightarrow{0} y) = s(l)$$

- In all other cases,

  $$\mathrm{Wt}(x \xrightarrow{d} y) = 0.$$

The first case is the most obvious. If $l = [w]$ is a class label of x then it can be seen to represent the unit headed by x which covers x and all words between x and y. The normalized strength of its outbound link property *Out* is therefore the strength assigned to the link from x to y (if this strength is negative, the weight is zero).

When $l = (w, 1)$, the best matching label is an adjacency label of x and w can be seen to represent y. Therefore, the inbound link properties of w should be used to determine the weight of the link $x \xrightarrow{d} y$. If the property In has positive strength, its normalized strength is used as the weight of the link. The link may, however, be either of depth 0 or of depth 1. A link $x \xrightarrow{1} y$ attaches x to y but does not place y inside the smallest bracket covering x. It is therefore a weaker attachment than a depth 0 link. This is indicated by a negative value of In^*. There is, however, one exception. If *Out* is positive, then a link in the opposite direction ($y \xrightarrow{d} x$) is indicated. Since this link is based on the *Out* property, it is a link of depth 0 (as in the first case of the weight function) and because back and forth links between words must be of equal depth, also the link $x \xrightarrow{d} y$ must be of depth 0.

If the strength of In is negative but the absolute value of In^* is greater than that of In, then In^* is considered a more reliable indication for the existence of a link. In this case, therefore, a positive normalized strength for In^* would be the weight assigned to the link $x \xrightarrow{d} y$. The depth of the link is always 0 in this case based on the same logic as above: if the link was already deduced at the first bootstrapping stage, In^*, then it indicates the stronger attachment of depth 0 links.

To explain the third case, recall that $s(l) > 0$ means that the label l is stronger than *Stop* on A_i^x. This implies a link unless the properties of w block it. One way in which w can block the link is to have a positive strength for the link in the opposite direction. Another way in which the properties of w can block the link is if $l = (l, 0)$ and $A_\sigma^w(Out) < 0$, that is, if the learning process has explicitly determined that no outbound link from w (which represents x in this case) is possible. The same conclusion cannot be drawn from a negative value for the *In* property when $l = (w, 1)$ because, as with standard dependencies, a word determines its outbound links much more strongly than its inbound links.

Let us return to the example of the lexical entry of the word *to* as learned from the Wall Street Journal corpus:

to

A_{-1}		A_1	
Stop	822	*Stop*	48
*In**	-4250	*In**	-981
In	-57	*In*	-1791
Out	-3053	*Out*	4010
[to]	5912	[to]	7009
[%_]	848	[_the]	3851
[in]	844	[_be]	2208
[the]	813	[will]	1414
[of]	624	[_a]	1158
[a]	599	[the]	954
[according_]	573	[of]	889
[for]	551	[would]	717
[expected_]	435	[n't]	604
[and]	346	[and]	546

On the left side of *to*, all linking properties have a negative strength. The potential link $x \xrightarrow{d} y$ may, for example, be a link from *to* to *going* in the sequence *going to*. The label [*to*] is then the best matching label for this link. Since this is a class label, it represents the position of x, that is, of the word *to* in this example (unsurprisingly, it is quite common for frequent words such as determiners and prepositions to be represented by their own class label). Therefore, the outbound linking property on the left side of the lexical entry of *to* must be used to calculate the weight of the link. Since the strength of *Out* is negative, the weight of the link is zero and no link is created.

Let us now consider the opposite link *going* $\xrightarrow{d}$ *to*. Here the best matching label is the adjacency label [_*to*]. Again, the left side linking properties of *to* are used. This time, however, the inbound rather than the outbound link properties are used. While these properties too are negative, the resulting weight is positive, because the third case of the weight function is applicable.

6.3.3 Simplified Basic Parsing Function Family

Examining the weight function defined for the basic parsing function family, one may wonder whether the use of the inbound link properties In^* and In cannot be simplified. Specifically, in the second case of the weight function ($l = (w, 1)$) it may be simpler to always use only the value of $A^w_\sigma(In)$ as the link weight rather than sometimes also using $A^w_\sigma(In^*)$. The weight function then becomes:

- If $l = (w, 0)$ and $A^w_\sigma(Out) > 0$:
$$\mathrm{Wt}(x \xrightarrow{0} y) = \min(s(l), \bar{A}^w_\sigma(Out))$$
- If $l = (w, 1)$ and $A^w_\sigma(In) > 0$:
$$\mathrm{Wt}(x \xrightarrow{d} y) = \min(s(l), \bar{A}^w_\sigma(In))$$
where if $A^w_\sigma(In^*) < 0$ and $A^w_\sigma(Out) \leq 0$ then $d = 1$ and otherwise $d = 0$.
- If $A^w_\sigma(Out) \leq 0$ and $A^w_\sigma(In) \leq 0$ and either $l = (w, 1)$ or $A^w_\sigma(Out) = 0$:
$$\mathrm{Wt}(x \xrightarrow{0} y) = s(l)$$
- In all other cases,
$$\mathrm{Wt}(x \xrightarrow{d} y) = 0.$$

I will refer to this as the *simplified basic parsing function family.* As the experiments reported in chapter 7 show, the two versions of the basic parsing function family perform quite similarly, with sometimes one performing somewhat better and sometimes the other.

6.3.4 On the Use of Punctuation

It is well known that using punctuation can improve the parsing accuracy of supervised parsers (Gregory et al. 2004). This is not surprising, as punctuation is intended to aid the human reader in understanding the text. For this reason, it also makes sense to use punctuation when the parser is unsupervised. Since punctuation is part of the original text, not of the annotation, its use does not compromise in any way the unsupervised nature of the algorithm. If unsupervised parsing is seen as an engineering problem, punctuation should certainly be used wherever it is available.

Klein (2005) removes punctuation in an attempt to make the data "better represent the data available to a human learner" (p. 14). The claim is that punctuation does not appear in the spoken language from which children learn. Klein qualifies this claim by pointing out that "it is arguable that at least some punctuation is correlated with information in an acoustic signal, e.g. prosody". Indeed, the relation between punctuation and prosody is not simple. It is well accepted that the two are related but that there is no one-to-one mapping. On the one hand, there is evidence (Altenberg 1987; Croft 1995) that there is a strong correlation between prosodic and syntactic boundaries. On the other hand, when Gregory et al. (2004) tried to use prosodic cues instead of punctuation when parsing transcribed speech, they discovered that punctuation improved parsing accuracy but prosodic cues decreased parsing accuracy. It should be stressed, however, that Gregory et al. do not claim that this proves that prosody cannot be used to improve parsing but only that their parsing methods could not make use of their specific coding of prosodic cues.

This still leaves open the question whether ignoring punctuation can make the algorithm more relevant to research on language acquisition by children. The question is not restricted only to punctuation, of course, but can be extended to the whole question of using written text to study issues in language acquisition. Even the transcribed text of child directed language is not a perfect representation of the input a child acquiring a language receives but because of the need to evaluate parsing accuracy against a syntactically annotated corpus, unsupervised parsers have actually been applied in recent years mainly to newspaper text (see the experiments in chapter 7). The first sentence of one such newspaper corpus, the Wall Street Journal Corpus, reads: *Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.* The question whether the two commas here are indeed correlated with the prosody which may have been used in uttering this sentence in the presence of a young child seems to be of secondary importance when considering the relevance of such examples to language acquisition. Moreover, singling out punctuation as unrepresentative of the input available to children while fully relying on the segmentation of the text into sentences and words seems unjustified to me.

There is not only no good reason to ignore punctuation but it is probably important to include punctuation in the model as long as it is the best substitute we have for the prosody available in the acoustic signal. This is because even if prosodic cues are not described accurately by punctuation, they play a similar role in constraining the syntactic structure without actually being part of it. Incorporating punctuation into the input available to the learner and the parser therefore creates a place in the model for cues going beyond the bare sequence of words and which are not directly part of the parse tree. Of course, the details of the algorithm may have to be changed when punctuation is replaced by prosodic cues, but this is also true of the treatment of words when written text is replaced by an acoustic signal. Even if the details may have to change, I believe that the principles may remain unchanged. In fact, the treatment of punctuation as defined above for the basic parsing function family is very crude and would have to be refined even for punctuation, but I believe that it correctly captures the principle that stopping punctuation should block links between words. In the present specification, all links are blocked by all stopping punctuation but this should be refined so that only some links are blocked.

6.4 Conclusion

In this chapter I presented a general architecture for a lexicalized family (Definition 6.1.3) of greedy parsing functions (Definition 6.1.2). The greedy parsing functions are based on a weight function which depends on a lexicon. The lexical entries in this lexicon consist of a sequence of adjacency points, each assigning weights to labels and linking properties. The architecture also specifies the learning process and which symbol is used to update each adjacency point at each step in the learning process (Algorithm 6.2.2).

A specific proposal for such a parsing function family was then fleshed out. This included the specification of how an adjacency point should be updated by each symbol and how the linking properties should be bootstrapped from the labels assigned each adjacency point (Algorithm 6.3.1). Finally, it specified the weight function which uses the labels and linking properties to assign a weight to each link $x \xrightarrow{d} y$ (section 6.3.2). This weight function was based on the linking properties of the best matching label between x and y.

The correctness of the parser function family and the learning algorithm is a matter of empirical examination. Some experiments conducted with these algorithms are reported in the next chapter. Of course, many choices had to be made and even within the framework proposed here many variants and modifications of the basic algorithm may be considered. Only a thorough empirical examination of these different alternatives, comparing the strengths and weaknesses of each variant on different corpora, may reveal the properties of syntactic structure which we are looking for. In this work the emphasis was on setting up the framework for the syntax learning task rather than on a thorough examination of the alternatives available within this framework.

I would like to conclude this chapter by pointing out which of the choices made in the design of the algorithm seem to me to be of a potentially lasting nature and which are more ad-hoc. This only refers to choices made within the framework outlined here, that is, to the choice of linking properties, update algorithm and weight function.

The way the labels are updated seems to me to be appropriate. Having examined the lexical entries of many words (admittedly, mainly in English) I believe that the labels and their strengths are an appropriate starting point for a bootstrapping algorithm. Similarly, the use of the properties of the best matching label between two words to determine the weight of the link between them seems to me appropriate and solves many of the standard problems in parsing, such as ambiguity and sparsity of the training corpus (which contains very few examples for most words but very many examples for the most frequent words). Also the use of the predicate $_\bullet A_i^\alpha$ as the starting point for the bootstrapping process seems to me to belong in future versions of the algorithm. Beyond this, I have opted for the simplest possible version of the algorithm, with only three linking properties derived directly from the predicate $_\bullet A_i^\alpha$. There is no doubt that the linking properties and their bootstrapping can be refined to distinguish between different situations. For example, a link added to solve a resolution violation is probably of a different nature than a link used to extend a structure by attaching an argument to a head. The current proposal treats all links equally, thus leading to many mistakes when resolving resolution violations.

These observations reflect my experience working with this model, but only further modifications and experiments can confirm or refute the validity of these observations.

Chapter 7

Experiments

7.1 Introduction

Natural language is a natural phenomenon and any theory of natural language should therefore be evaluated by testing it empirically. Such testing is complicated because much of natural language theory deals with entities, such as meaning and syntactic structure, which are not directly observable and measurable. One way to solve this problem is to make use of the fact that language is produced and interpreted by humans and is therefore a reflection or part of cognitive processes. Empirical questions about language can then be coined as questions about these language related cognitive processes and psychological experiments can be set up to examine the correctness of different linguistic hypotheses. This is the domain of psycholinguistic research. While these methods can reveal much about natural language, it is often difficult (and always expensive) to conduct well-controlled psychological experiments to test some of the deeper and more complex parts of linguistic theory. Moreover, the interpretation of the results of psycholinguistic experiments is often difficult and depends on the linguistic theoretical framework being used.

A weaker version of this approach is to examine the judgments of humans about different linguistic questions (such as grammaticality and structure) in uncontrolled settings. The number of subjects in these experiments is often as small as one and the subject may often be the linguist herself or a colleague. The subject is then often aware of the purpose of the experiment and may be required to have sufficient knowledge of the theory behind it to be able to answer the question. These are all considered fatal flaws in the setting of a psychological experiment, but the ease with which such experiments may be conducted means that a considerable body of evidence has been collected, which cannot be ignored. This method (together with simple introspection) has long been used by linguists and underlies much of linguistic theory.

One version of these sort of experiments, which is relevant to the present chapter, is the annotation of corpora. An annotator is given guidelines (usually with examples) as to how a corpus of language should be annotated. The annotator then uses these guidelines together with his natural competence in the language to annotate the corpus. The annotation guidelines certainly constrain the annotation the annotator can produce but do not determine it completely (as evidenced by the fact that a computer could not perform the same task). The choices made by the annotator within the constraints of the guidelines reflect the cognitive processes involved in performing the annotation. When the same corpus is annotated by several annotators, the annotations can be compared. Similarities between the annotations reflect (within the constraints of the guidelines) the common and stable parts of the cognitive processes involved.

The approach to linguistics outlined above, which sees the connection between form and meaning (and therefore also cognition) as the main subject of linguistic study is known as the *mediationalist* view of linguistics. A very different approach to linguistics is one which focuses on the combinatorial structure of language rather than on its relation with cognition. This is known as the *distributional* view of linguistics (see Huck and Goldsmith (1996) for further discussion). In the distributional approach, the main source of linguistic evidence is unannotated corpora (to the extent that such corpora exist, as any writing system is already a form of annotation). The goal of research is to discover structural relations which hold in these corpora.

This approach seems appealing because it does not have to deal (at least directly) with hidden cognitive entities and with the general messiness of human cognition. An experiment is simply the application of an algorithm to some corpus. The problem with such experiments is to meaningfully interpret their results. The mere fact that an algorithm has produced some output on a corpus does not in itself mean much: the algorithm simply calculates the value of some property (simple or complex) on the corpus.

The first way of showing that such a property is linguistically meaningful is to show that the value of the property remains stable across different corpora. This is then a discovery of a linguistic invariant. Many such linguistic invariants can be discovered, but most of them result in little generalization. For example, calculating the most frequent preposition appearing with a verb does not generalize beyond a specific verb in a specific language and is sometimes even domain dependent. This is still very useful information if one wishes to construct a dictionary, but does not lead to any theoretical generalization about the structure of language.

An example of a generalization which can be discovered by this method is the Zipfian distribution of words, which states that words in a language obey a power law probability distribution. This law seems to hold with little variation across different languages and domains and can be verified to hold without having to make any reference to the meaning of language or to any related cognitive processes. But Zipf's law is a rare exception. Very few linguistic generalizations have been discovered or proved by strictly distributional methods.

Because it turns out to be so difficult to arrive at any significant linguistic results based on purely distributional methods, researchers have introduced the use of annotated corpora. When these corpora are annotated by humans (as is usually the case) this introduces mediationalist results into the distributionalist method. The annotated corpora may be used both as input for an experiment and to evaluate the results of that experiment. Specifically, one can compare the output of an algorithm on an *un*annotated corpus with an annotated version of the corpus.

The first and most important question in the present work is whether there is a relation between the surface statistics of languages and their hidden syntactic structure and whether the suggested unsupervised incremental parser is a (rough) approximation of such a relation. In other words, can the incremental parser (and its learning algorithm) deduce the hidden syntactic structure of a language from example utterances in the language? To test this, the output of the incremental parser on an unannotated corpus can be compared with an annotated version of that corpus. If the agreement between the two is sufficiently good, the parser must have discovered at least part of the syntactic structure.

The annotation of a corpus depends very much on the linguistic theory used by the annotators as well as on various arbitrary decisions made when defining the annotation guidelines. It is, of course, unreasonable to expect an unsupervised parser to capture all the arbitrary choices made when annotating a corpus. This means that when the parser fails to agree with the annotation, this may reflect not only errors on the part of the parser but also legitimate alternative syntactic analyses. If, however, the parser and the annotation agree on more of the analysis than could be expected by chance (or by some trivial method) this can be seen as a clear indication that the parser has captured some syntactic structure which is also present in the annotation. So, while failure can be blamed on the annotation, success cannot be an artifact of the annotation, as long as the success threshold is set sufficiently high.

The present chapter is dedicated to experiments conducted with several annotated corpora and shows that the agreement between the parses produced by the incremental parser and the annotation of these corpora is better not only than that which would have been expected by chance but even better than some higher success threshold. This can be seen as evidence that the learning algorithm successfully discovers at least some of the syntactic structure of these corpora.

7.2 Experiments and Evaluation

For the experiments, corpora annotated for syntactic structure were needed. Because the parser treats words as atomic units and performs no morphological analysis of words, only languages with a weak morphological component were

	WSJ	Negra	CTB
empty	-NONE-	tags starting with *	-NONE-
punctuation	, . : “ ” -LRB- -RRB-	$. $, $*LRB* $*RRB*	PU
currency	$ #		

Table 7.1: Tags used to categorize the empty, punctuation and currency symbol leaf types in each corpus.

chosen for the experiments. This leaves one with a variety of languages with different syntactic structures for which large syntactically annotated corpora exist. Of these, I used those corpora which have recently been used by other researchers (Klein and Manning 2002; Klein and Manning 2004; Bod 2006a; Bod 2006b) to evaluate unsupervised parsing: the Wall Street Journal Corpus version 2.0 (Marcus et al. 1993) for English, the Negra Treebank version 2.0 (Skut et al. 1997) for German and the Chinese Treebank version 5.0 (Xue et al. 2002). For the first two corpora I used the same version used by the researchers cited above, so results are completely comparable. For the Chinese Treebank, I used the latest version 5.0 while Klein and Manning used version 3.0. The differences between the versions of the Chinese Treebank are significant and therefore the results are not directly comparable (see section 7.4 for additional discussion of the differences).

For each corpus, the complete plain text of the corpus (including punctuation) was used for learning. This first pass produced a lexicon (as described in chapter 6) which was then used to parse different subsets of the corpus (as specified below). These parses were evaluated against the annotation in the corpus. To allow easy comparison, I used the same evaluation method and all the subsets of the corpora used in the work cited above (as well as additional subsets). In this I followed the specifications given in chapter 2 of Klein (2005).

7.2.1 Input to the Parser

The input to the parser (the plain text) was extracted from the annotated corpora. All three treebanks use the same labeled bracketing annotation format for syntactic structure (but with different labels). The leaves of the annotated tree structures are either words, punctuation and currency symbols or empty nodes. To determine the type of each leaf, its part-of-speech tag (in the annotation) was used. Table 7.1 gives the tags used for each corpus to determine which nodes are empty nodes, currency symbols or punctuation symbols. All other leaf nodes were considered to be words (these included, among others, symbols such as ’%’).

Klein does not treat currency symbols as words because in spoken language they are not pronounced in the same place as they appear in the written text (as opposed to ’%’ which is written and spoken in the same position). Because

Klein is interested in the implications of his work to the acquisition of language by children, he tries to make the input to his algorithm as similar as possible to the spoken input children may be exposed to. While any claims made about child language acquisition based on work with the Wall Street Journal Corpus must be tenuous, I adopt Klein's decision not to treat currency symbols as words. This simplifies the comparison of the results of the different parsing algorithms and the treatment of currency symbols makes little difference for the overall results (as Klein himself stresses). Only the Wall Street Journal Corpus contains such currency symbols.

Having categorized the tokens appearing in the leaves of the annotated sentences in each corpus, the words and the punctuation symbols were taken as the input to the parser (while empty nodes and currency symbols were discarded). The words were converted to all lower-case (so that a token of a word at the beginning of a sentence will be identical to the token of that word appearing elsewhere in a sentence). The resulting sequence of words and punctuation symbols is very similar to that which could have been extracted directly from the plain corpus text except that word segmentation was determined by the annotation. Usually this agrees with the trivial segmentation based on white space between words, but in some cases the annotation splits single plain text tokens. For example, in the Wall Street Journal Corpus, the word *weren't* is split into two tokens: *were* and *n't*. While this perhaps deviates from strictly working with plain text, it does not seem to be of significant importance and greatly simplifies the evaluation against the annotated corpus (and comparison with previous work).

While punctuation was included in the input made available to the parser, only the words were considered to occupy a position in the syntactic structure of the sentences. Punctuation was therefore only treated as indicating some structural properties rather than being part of the structure itself. The length of sentences is then defined to be the number of *words* in each sentence. Subsets of sentences of bounded length were then extracted from each corpus. I write WSJX, Negra$X$ and CTBX for the subset of sentences of length less than or equal to X of the Wall Street Journal Corpus, the Negra Treebank and the Chinese Treebank, respectively. The plain text of the full corpora was used for learning while evaluation was performed on different subsets of the corpora. These subsets include all those subsets on which previous authors (Klein and Manning 2002; Klein and Manning 2004; Bod 2006a; Bod 2006b) report their results.

There are three differences between the input used here and the input used in Klein and Manning's and Bod's experiments. These should be examined closely to ensure that the comparison between the experiments is valid.

1. Both Klein and Manning's and Bod's algorithms parse from sequences of parts-of-speech. In most experiments, part-of-speech sequences are extracted directly from the annotated corpus. In some of Klein and Manning's experiments, the part-of-speech sequences were induced from the words of

the sentences using an unsupervised tagger similar to that of Schütze (1995) (see section 2.1.4 of Klein (2005) for details). In the present work, all learning and parsing is performed directly from plain text as it could have been extracted from the original source of the text. This means that results are directly comparable only for those tests which Klein and Manning conducted on their automatically labeled corpus. They did this only for the WSJ10.

2. Neither Klein and Manning nor Bod use the punctuation available in the corpus while my algorithm uses punctuation whenever it finds it beneficial to do so. I have explained already in section 6.3.4 why I believe the use of punctuation does not compromise the unsupervised nature of the parser. If other algorithms choose to ignore this information available in plain text, they may do so. The results of the experiments remain comparable. This having been said, I also report (in section 7.4.5) parsing results without punctuation and analyze the effect punctuation has on the accuracy of the incremental parser.

3. Klein, Manning and Bod use the same reduced subsets of the corpora both for training and for evaluation. My algorithm uses the full corpora for training and only the reduced subsets for evaluation. This may suggest that my algorithm uses significantly more training data. However, because my algorithm parses directly from plain text, the true comparison is with the amount of data needed not only to train the other parsers on part-of-speech sequences but also for training an unsupervised tagger to induce these sequences. When Klein and Manning do induce the part-of-speech sequences they do so using not only the full Wall Street Journal Corpus but also additional 1994-1996 Wall Street Journal newswire (Klein 2005). The training data I use are therefore less and not more than that used by others for comparable tasks.

7.2.2 Evaluation

The results were evaluated by comparing the unlabeled bracketing produced by the parser with the unlabeled bracketing in the annotated corpora. Because empty nodes, punctuation and currency symbols are not part of the parse produced by the parser, a bracket is defined (as in section 2.2) as the set of word tokens it covers. Brackets in the annotated corpora are labeled and may cover leaf nodes which are not words. Therefore, when these brackets were reduced to the set of words they cover, some brackets which were different in the annotation may have reduced to identical brackets and may even have disappeared (for example, a bracket covering a single empty node). The reduction did not change, however, the constituency relations between words in the sentence, that is, which words belong to the same constituents. This is the relation on which the parser was evaluated and the evaluation was therefore performed by comparing the brackets (as sets of words) produced by the parser with those annotated in the corpus. Brackets covering only a single word indicate a trivial constituent and were therefore ignored in the comparison.

Let P be the set of brackets produced by the parser on a corpus C and let B be the set of brackets annotated in the corpus (both sets after the reduction described above and the removal of single word brackets). Two scores were used to evaluate the accuracy of the parse produced by the parser: unlabeled precision, $\mathrm{UP}(P,B)$, and unlabeled recall, $\mathrm{UR}(P,B)$:

$$\mathrm{UP}(P,B) = \frac{|P \cap B|}{|P|}$$

$$\mathrm{UR}(P,B) = \frac{|P \cap B|}{|B|}$$

These two numbers represent (respectively) what fraction of brackets suggested by the parser were indeed correct and what fraction of the brackets in the annotated corpus were discovered by the parser. These are standard measures of parsing accuracy and follow exactly the evaluation in previous work (Klein and Manning 2004; Bod 2006a; Bod 2006b). One possible objection to this evaluation metric is that the top bracket (which covers the whole sentence) is also included. Because this bracket is trivial this tends to increase the score. This does not matter much, though, because the absolute precision and recall values do not mean much anyway if they are not 0 or 1.0 (what does a UP of 0.6 say about a parser?). What is important is that these scores allow us to compare the performance of the parser with different baselines and with other parsers.

To condense the evaluation metric into a single number, the harmonic mean of precision and recall is often reported:

$$F_1(P,B) = \frac{2 \cdot \mathrm{UP}(P,B) \cdot \mathrm{UR}(P,B)}{\mathrm{UP}(P,B) + \mathrm{UR}(P,B)}$$

This form of averaging penalizes a low score in one of its components more severely than the standard arithmetic mean (a score of 0 in one component and 1 in the other results in a 0 F_1 score). It is customary to report the scores as percentage numbers (ranging between 0 and 100 rather than 0 and 1) and I follow this convention here.

7.3 Baselines

The evaluation scores condense the performance of a parser into a single number (or two). This hides many of the details of the parser's performance but allows a quick first analysis of the parser: did it manage to detect syntactic structure successfully or not? The answer is seldom a simple yes or no. Parsers almost never achieve a perfect score so the question remains whether a certain score indicates success or not. A rough answer to this question can be given by comparing the scores of the parser to scores achieved by other methods. These methods then define the threshold values for different levels of success. These are often referred to as baselines for the evaluation.

Random Baseline The lowest level of success is usually taken to be the score achieved by generating parses randomly. Any parser scoring not much better than such random parses cannot be claimed to detect any syntactic structure. However, there are many ways of generating random parses and these may result in different baselines. Each method of generating random parses may be biased towards less or more linguistically plausible parses.

Left/Right Branching Baseline A second baseline often used is the left-branching or right-branching parsing heuristic. This heuristic assigns any utterance $\langle x_1, \ldots, x_n \rangle$ either the right-branching parse $[x_1\ [x_2 \ldots [x_{n-1}, x_n] \ldots]\ ]$ or the left-branching parse $[\ [\ldots [x_1, x_2] \ldots x_{n-1}]\ x_n]$ ($\mathcal{B} = \{\langle x_i, \ldots, x_n \rangle\}_{i=1}^{n-1}$ and $\mathcal{B} = \{\langle x_1, \ldots, x_i \rangle\}_{i=2}^{n}$, respectively). This heuristic captures the strong left or right branching tendency found in many languages: most heads tend to take their arguments on the same side of the head. Such a tendency is a linguistic property of the language and discovering that a language has it is already a discovery of linguistic structure. Therefore, scoring above the left and right branching baselines is in no way a minimal threshold for success. In fact, it seems that Klein and Manning (2002) were the first to propose an unsupervised parser which does better than the right branching heuristic for English.

Left/Right Branching with Punctuation Since the way the incremental parser uses punctuation is fixed and not learned, it is interesting to introduce an additional baseline, combining the left or right branching heuristic with punctuation. The right (left) branching parse is induced by the shortest common cover link set in which there is a link from each word to the word to its right (left). The right (left) branching with punctuation baseline is then defined to be the bracketing induced by the shortest common cover link set in which there is a link from every word to the word to its right (left) unless they are separated by stopping punctuation. The effect of punctuation on the left/right branching heuristic coded in this baseline is similar to the effect of punctuation on the parses produced by the incremental parser, where no links are allowed to cross stopping punctuation.

Klein and Manning report left-branching, right-branching and random baselines for their results. Their random baseline uses a uniform distribution over all binary trees with a given number of leaves. Their results show that for all three corpora (WSJ, Negra and CTB) the right-branching baseline is higher than the random baseline. I therefore only use the left and right branching baselines (with and without punctuation) as the lowest baselines against which parsing results are compared.

7.4 Results and Discussion

7.4.1 Results

The incremental parsing algorithm was evaluated for both the *basic parsing function family* (section 6.3) and the *simplified parsing function family* (section 6.3.3). In the following tables these are referred to as the *incremental* and *simplified incremental* algorithms (respectively). For each of the three corpora, the parser first learned a lexicon using the plain text of the full corpus and then the parser with this lexicon was evaluated on different subsets of the corpus with bounded sentence lengths. For each corpus these subsets were the sentences of length up to 10 words, up to 20 words and up to 40 words as well as the full corpus.

All three languages tested have some right branching tendency (to a greater or lesser degree). To test whether the parser can also handle languages with a left branching tendency, the learner and parser were also applied to the reversed sentences of each language.[1] Because the parser is incremental, learning and parsing from the reversed language is not the same as learning and parsing the original language. Each experiment was therefore conducted twice: once for the original language and once for the reversed language. Parsing of the reversed language will be referred to as right to left parsing.

The parsing results are summarized in the tables which appear in the following pages. Each table is divided into three sections. The top section gives the left and right branching baselines (with and without punctuation) for each subset of each corpus. The second section gives the results reported by other researchers (Klein and Manning 2004; Bod 2006a; Bod 2006b) for unsupervised parsing from sequences of parts-of-speech (as read from the annotated corpus). The algorithms CCM, DMV and DMV+CCM(POS) are from Klein and Manning (2004) while U-DOP and UML-DOP are from Bod (2006a) and Bod (2006b). The bottom section of each table reports parsing results directly from plain text. These are mainly the results for the different variants of the incremental parser of the present work. In addition, for the WSJ10 there are also results for Klein and Manning's DMV+CCM algorithm when running from automatically induced sequences of

[1] I would like to thank Alexander Clark for suggesting this to me.

Algorithm	WSJ10 (7422 sentences) UP	UR	UF$_1$	WSJ20 (25523 sentences) UP	UR	UF$_1$
Baselines						
Right-branching	55.2	70.0	61.7	42.4	55.9	48.2
Right-branching+Punct.	59.1	74.4	65.8	50.4	64.9	56.7
Left-branching	25.7	32.6	28.7	15.2	20.0	17.2
Left-branching+Punct.	28.2	35.5	31.5	19.3	24.9	21.8
Parsing from POS						
DMV	46.6	59.2	52.1			
CCM	64.2	81.6	71.9			
DMV+CCM (POS)	69.3	88.0	77.6			
U-DOP	70.8	88.2	78.5			
UML-DOP			82.9			
Parsing from plain text						
DMV+CCM (DISTR.)	65.2	82.8	72.9			
Incremental	75.6	76.2	75.9	65.7	63.7	64.7
Incremental (right to left)	75.9	72.5	74.2	66.0	59.6	62.6
Simplified Incremental	75.3	76.1	75.7	65.5	63.6	64.5
Simplified Incremental (right to left)	75.7	72.6	74.1	65.8	59.6	62.5

Algorithm	WSJ40 (47385 sentences) UP	UR	UF$_1$	Full WSJ (49208 sentences) UP	UR	UF$_1$
Baselines						
Right-branching	35.4	47.4	40.5	34.5	46.3	39.5
Right-branching+Punct.	44.5	57.7	50.2	43.8	56.8	49.4
Left-branching	10.6	14.1	12.1	10.1	13.5	11.6
Left-branching+Punct.	15.2	19.7	17.2	14.8	19.2	16.7
Parsing from POS						
U-DOP			63.9[1]			
UML-DOP			66.4[1]			
Parsing from plain text						
Incremental	58.9	55.9	57.4	58.1	55.0	56.5
Incremental (right to left)	59.3	52.2	55.6	58.4	51.3	54.6
Simplified Incremental	58.7	55.9	57.2	57.9	55.0	56.4
Simplified Incremental (right to left)	59.0	52.3	55.5	58.1	51.3	54.5

[1]Unlike other results reported here, the results for U-DOP and UML-DOP on WSJ40 were calculated on a random 90/10 split of WSJ40, not the full WSJ40. This probably doesn't change the results much (Bod 2006a).

Table 7.2: Parsing results for the Wall Street Journal Corpus.

Algorithm	Negra10 (7542[1] sentences) UP	UR	UF$_1$	Negra20 (15675 sentences) UP	UR	UF$_1$
Baselines						
Right-branching	33.9	60.1	43.3	21.8	42.2	28.8
Right-branching+Punct.	35.4	62.5	45.2	24.7	46.9	32.4
Left-branching	27.4	48.6	35.1	16.8	32.5	22.2
Left-branching+Punct.	29.5	52.1	37.7	20.6	39.2	27.0
Parsing from POS						
DMV	38.4	69.5	49.5			
CCM	48.1	85.5	61.6			
DMV+CCM(POS)	49.6	89.7	63.9			
U-DOP	51.2	90.5	65.4			
UML-DOP			67.0			
Parsing from plain text						
Incremental	51.0	69.8	59.0	39.2	54.7	45.7
Incremental (right to left)	50.4	68.3	58.0	37.9	52.1	43.9
Simplified Incremental	45.3	65.6	53.6	33.6	51.0	40.5
Simplified Incremental (right to left)	44.8	63.8	52.6	32.0	47.2	38.1

Algorithm	Negra40 (20301 sentences) UP	UR	UF$_1$	Full Negra (20602 sentences) UP	UR	UF$_1$
Baselines						
Right-branching	17.6	35.0	23.4	17.2	34.2	22.8
Right-branching+Punct.	20.9	40.4	27.6	20.1	39.8	27.1
Left-branching	13.1	26.0	17.4	12.7	25.2	16.9
Left-branching+Punct.	17.6	33.9	23.1	17.2	33.3	22.7
Parsing from plain text						
Incremental	34.8	48.9	40.6	34.3	48.2	40.1
Incremental (right to left)	32.9	45.5	38.2	32.3	44.8	37.6
Simplified Incremental	29.4	45.5	35.7	29.0	44.9	35.3
Simplified Incremental (right to left)	27.2	40.7	32.6	26.7	40.0	32.0

[1]The number of sentences in Negra10 reported here (7542) is significantly larger than the 2175 sentences reported for Negra10 in Klein and Manning (2004) and Klein (2005). However, the left and right branching scores reported here are identical to those reported by Klein and Manning. This suggests that the corpora used are identical and that the difference in reported number of sentences has its source elsewhere.

Table 7.3: Parsing results for the Negra Corpus.

Algorithm	CTB10 v3.0 (2437 sentences) UP	UR	UF$_1$
Baselines			
Right-branching	29.0	53.9	37.8
Left-branching	26.3	48.8	34.2
Parsing from POS			
CCM	34.6	64.3	45.0
DMV	35.9	66.7	46.7
DMV+CCM(POS)	33.3	62.0	43.3
U-DOP	36.3	64.9	46.6
UML-DOP			47.2

	CTB10 v5.0 (4626 sentences)			CTB20 v5.0 (9491 sentences)		
Algorithm	UP	UR	UF$_1$	UP	UR	UF$_1$
Baselines						
Right-branching	43.3	60.4	50.4	32.4	40.7	36.1
Right-branching+Punct.	46.6	64.4	54.1	40.9	49.7	44.9
Left-branching	30.8	42.9	35.8	19.0	23.9	21.2
Left-branching+Punct.	34.6	47.9	40.2	27.8	33.7	30.4
Parsing from plain text						
Incremental	54.2	55.1	54.6	48.7	40.4	44.1
Incremental (right to left)	49.6	58.6	53.7	41.8	41.6	41.7
Simplified Incremental	53.4	61.5	57.2	48.4	47.3	47.8
Simplified Incremental (right to left)	50.0	60.4	54.7	42.6	43.6	43.1

	CTB40 v5.0 (16323 sentences)			Full CTB v5.0 (18787 sentences)		
Algorithm	UP	UR	UF$_1$	UP	UR	UF$_1$
Baselines						
Right-branching	23.5	29.2	26.0	19.7	24.6	21.8
Right-branching+Punct.	37.0	43.1	39.8	35.7	41.2	38.3
Left-branching	12.7	15.8	14.1	10.5	13.2	11.7
Left-branching+Punct.	23.8	27.7	25.6	22.3	25.7	23.9
Parsing from plain text						
Incremental	43.9	33.5	38.0	42.2	31.6	36.1
Incremental (right to left)	34.0	32.8	33.4	30.5	29.6	30.0
Simplified Incremental	44.2	40.8	42.4	42.8	38.9	40.8
Simplified Incremental (right to left)	35.2	34.9	35.0	32.0	31.7	31.9

Table 7.4: Parsing results for the Chinese Treebank versions 3.0 and 5.0.

parts-of-speech (Klein and Manning 2004; Klein 2005). Klein, Manning and Bod report results only for the shortest sentences in each corpus (up to length 10), the only exception being Bod's results for WSJ40. Parsing directly from plain text is only reported by Klein and Manning for WSJ10. Where results do not appear in the tables below, they were not reported by the researchers.

7.4.2 Baselines

Before examining the parsing accuracy of the incremental parser, let us first examine the baselines calculated for the corpora. For all subsets of all corpora, the right-branching heuristic performs better than the left-branching heuristic. Moreover, right-branching scores significantly better on the English corpus than on the other two corpora and somewhat better on Chinese than on German. This agrees with what we know about the structure of these languages: English is a strongly right-branching language while Chinese and German have both right and left branching constructions.

Adding punctuation to the left/right-branching heuristic always improves the performance of the heuristic. As sentences become longer, this improvement becomes more significant. This is not surprising because punctuation is used to segment long sentences into smaller segments and as sentences become longer they typically consist of more such segments. The effect of using punctuation is strongest for Chinese, somewhat weaker for English and significantly weaker for German.

It is interesting to note the big difference between the right-branching scores calculated by Klein and Manning for CTB10 version 3.0 and those calculated here for CTB10 version 5.0. This shows that the two versions of the corpus are very different in the sort of syntactic trees they contain. It is not clear to me whether this is due merely to a difference in annotation or to a more fundamental difference in the types of short sentences which appear in the two versions of the corpus.

7.4.3 Incremental Parsers Compared with Baseline

A first rough analysis of the success of the incremental parser in discovering syntactic structure can be based on the comparison of its evaluation scores with those of the highest baseline, right-branching with punctuation. Since this baseline codes significant information about the language being parsed it provides a reasonably high threshold for success.

The incremental parser using the *simplified* basic parsing function family achieves a higher F_1 score than the right-branching baseline with punctuation on all subsets of all corpora. On the Chinese Treebank the difference is small but on the other corpora it is quite significant. Moreover, on all subsets of the WSJ and Negra corpora, all versions of the incremental parser have an F_1 score above the baseline.

Because the right-branching (with punctuation) baseline typically creates more brackets than the incremental parser, the incremental parser improves more on the baseline precision than on the recall. On the subsets of the Wall Street Journal Corpus, all versions of the incremental parser improve on the baseline precision but score either a bit higher or somewhat lower than the baseline recall. On the subsets of the Negra Corpus, all versions of the incremental algorithm improve both on the precision and the recall of the baseline. Finally, on the subsets of the Chinese Treebank, recall of all incremental parsers is lower than the baseline but the precision of the left-to-right incremental parsers is higher than the baseline precision.

To summarize, the simplified incremental parser always succeeds to improve in total on the baseline but on the Chinese Treebank this improvement is small. Improvement is most significant on the Negra Corpus where both precision and recall are improved significantly and consistently. On the Wall Street Journal Corpus improvement on the baseline is mostly due to improvement on precision rather than on recall. This is probably because the right-branching tendency of English is the strongest among the languages tested.

In absolute terms, parsing was most successful on English and least successful on Chinese, with German somewhere in between. This pattern is common to all unsupervised parsers (and probably also to most supervised parsers) and reflects the fact that English has been the main language of study in modern linguistics and even more so in computational linguistics (the present work being no exception to this). As a result, English has a simpler syntactic structure than other languages when viewed through the linguistic and computational tools we commonly use. Whether English is also simpler in structure in some theory independent way is a question which is open to debate.

7.4.4 Comparing Different Incremental Parser Variants

Having compared the incremental parser to the baseline, we can now compare the different versions of the incremental parser. In general, the differences in performance are not great. On the Negra Treebank the basic parsing function family does somewhat better than the simplified basic parsing function family while on the Chinese Treebank it is the other way around. On the Wall Street Journal Corpus the two versions of the algorithm achieve practically identical accuracy.

Left-to-right (forward) parsing scores consistently better than right-to-left (backward) parsing on all subsets of all corpora. The only exceptions are the precision on subsets of the Wall Street Journal Corpus and recall on the small subsets of the Chinese Treebank. In both cases, right-to-left parsing improves only marginally on left-to-right parsing. In general, the difference in parsing

Algorithm	WSJ10			Negra10			CTB10		
	UP	UR	UF_1	UP	UR	UF_1	UP	UR	UF_1
No Punctuation	68.7	65.5	67.1	39.8	61.2	48.2	48.5	47.8	48.1
Punct. in Learning	74.1	73.9	74.0	50.6	69.0	58.4	53.1	53.5	53.3
With Punctuation	75.6	76.2	75.9	51.0	69.8	59.0	54.2	55.1	54.6

Algorithm	WSJ40			Negra40			CTB40		
	UP	UR	UF_1	UP	UR	UF_1	UP	UR	UF_1
No Punctuation	47.4	43.3	45.3	22.8	37.6	28.4	29.1	22.0	25.1
Punct. in Learning	54.0	50.7	52.3	33.4	46.8	39.0	38.7	29.5	33.5
With Punctuation	58.9	55.9	57.4	34.8	48.9	40.6	43.9	33.5	38.0

Table 7.5: The effect of punctuation on the incremental parser. This table compares parsing accuracy of the incremental parser when punctuation is completely ignored, when it is used only in learning and when it is also used in parsing.

accuracy between left-to-right and right-to-left parsing is small, suggesting that the incremental parser can handle both right and left branching structures. The consistent advantage of forward parsing over backward parsing should not be surprising because humans use and process language from beginning to end and not the other way around. Left-branching languages are therefore not simply mirror images of right-branching languages and it is not unlikely that these mirror images contain constructions which are difficult to parse incrementally (and would therefore not appear in a real left-branching language).

7.4.5 The Effect of Punctuation

To examine the effect the use of punctuation has on the accuracy of the incremental parser, I repeated the experiments without punctuation. Table 7.5 shows how parsing accuracy of the incremental parser changes with the type of punctuation information available to the parser. It can be seen that when the parser completely ignores punctuation, its performance deteriorates quite significantly on all corpora. However, when punctuation is available to the parser while learning, but not while parsing, parsing accuracy is already much better. On all short sentence corpora and on Negra40, parsing accuracy is then quite close to parsing accuracy with full punctuation. This shows that while punctuation does assist directly in making correct parsing decisions (in the parsing phase) it is even more significant in helping the algorithm learn the correct linking properties of words. Because these properties are statistical in nature, this means that occasional errors in punctuation should have relatively little effect on the accuracy of the parser. As sentence length increases, it becomes more and more important to use punctuation directly in parsing (and not only in learning). This is no surprise, since punctuation is intended to assist the human reader in deciphering the structure of long and complicated sentences.

7.4.6 Comparison with Other Parsers

Direct comparison between the incremental parser and the unsupervised parsers of others (Klein and Manning 2004; Bod 2006a; Bod 2006b) is complicated by the fact that most of Klein and Manning's and all of Bod's[2] results are reported for parsing from sequences of part-of-speech as extracted from the annotated corpus while the incremental parser was tested on learning and parsing directly from plain text. All unsupervised parsing algorithms from part-of-speech sequences can be converted into unsupervised parsers from plain text by combining them with algorithms for the unsupervised induction of parts-of-speech. This was done by Klein and Manning for their CCM algorithm in Klein and Manning (2002) and for the DMV+CCM algorithm in Klein and Manning (2004). In both cases only the WSJ10 corpus was parsed. These results should be directly comparable with parsing results of the incremental parser but the results for the CCM algorithm reported in Klein and Manning (2002) use a somewhat different evaluation metric than the one used in all later papers (including the present work). It seems that the difference between the two metrics is not significant (see chapter 2 of Klein 2005 for details), as can be seen in table 7.6, which summarizes parsing results on the WSJ10 for the incremental parser and for Klein and Manning's CCM and DMV+CCM algorithms.

As can be seen in table 7.6, using induced parts-of-speech resulted in some degradation in the parsing accuracy of both the CCM and the DMV+CCM models. The DMV+CCM model does better than the CCM model both when parsing from the corpus part-of-speech sequences and when parsing from induced part of speech sequences. All versions of the incremental parser achieve a higher F_1 score than either CCM or DMV+CCM parsing from induced part-of-speech sequences. The incremental parser scores somewhat lower than DMV+CCM on recall but higher on precision. Comparing DMV+CCM with the results in table 7.5 shows that when no punctuation at all was used by the incremental parser it scored worse than DMV+CCM (but still with higher precision) while using punctuation only in the learning phase resulted in a somewhat higher F_1 score than that of DMV+CCM.

It is tempting to try to draw conclusions from Klein and Manning's experiments about the effect using induced parts-of-speech may have on the performance of unsupervised parsers parsing from sequences of parts-of-speech. However, two experiments using the same corpus and two related parsing algorithms can hardly

[2]In Bod (2007a) and Bod (2007b), U-DOP*, a variant of U-DOP, is applied directly to words, as a component in a machine translation algorithm. This does not provide results comparable to other experiments reported in this chapter.

	WSJ10 (7422 sentences)			
Algorithm	metric	UP	UR	UF$_1$
Parsing from POS				
CCM	old	63.8	80.2	71.1
CCM	new	64.2	81.6	71.9
DMV+CCM	new	69.3	88.0	77.6
Parsing from plain text				
CCM (DISTR.)	old	56.8	71.1	63.2
DMV+CCM (DISTR.)	new	65.2	82.8	72.9
Incremental	new	75.6	76.2	75.9
Incremental (right to left)	new	75.9	72.5	74.2
Simplified Incremental	new	75.3	76.1	75.7
Simplified Incremental (right to left)	new	75.7	72.6	74.1

Table 7.6: Parsing results on the WSJ10 for different algorithms. The 'new' evaluation metric is the metric defined in section 7.2.2 while the 'old' evaluation metric is the one defined in section 2.2.5 of Klein (2005).

provide sufficient evidence for any such conclusions. In fact, even the conclusion that using induced parts-of-speech must necessarily degrade parsing performance is not necessarily true. For example, Prescher (2005) showed that unsupervised refinement of the labels annotating the Wall Street Journal Corpus can improve the parsing performance of unlexicalized PCFG parsers. Unsupervised induction of parts-of-speech may, in principle, have a similar effect on unsupervised parsing algorithms. It is probably true, however, that at the current state of unsupervised part-of-speech induction and unsupervised parsing we are more likely to see a degradation in performance when using induced parts-of-speech. How large this degradation may be is also not clear. The CCM algorithm F_1 score drops by 7.9 percentage points when using induced parts-of-speech while the DMV+CCM algorithm's F_1 score drops by only 4.7 percentage points when using the same induced parts-of-speech.

All this having been said, it is still interesting to be able to roughly compare the incremental parser with unsupervised parsers which were not tested with induced parts-of-speech and on corpora for which induced part-of-speech experiments were not conducted. For such a comparison I will assume that induced part-of-speech F_1 scores are approximately 5 percentage points lower than the F_1 scores for the same parser parsing from the annotated parts-of-speech. Using this rough rule, the incremental parser scores within two, three and four percentage points from the best performing parser (UML-DOP) on the WSJ10, Negra10 and WSJ40 corpora (respectively). These are all the corpora on which comparison is at all possible (comparison on the Chinese Treebank is not possible because of the significant difference between the versions of the corpus used in the experiments). It is therefore possible to draw a rough conclusion that the incremental parser achieves a similar level of performance to that of the best unsupervised parsers currently available.

7.4.7 Error Analysis

To get some idea of the sort of errors made by the incremental parser, one can look at parsing accuracy for different types of brackets and at the types of brackets where errors are most frequent. Two ways of defining bracket type are by the sequence of parts-of-speech covered by the bracket and by the non-terminal assigned to the bracket (in the annotated corpus).

Table 7.7 lists the part-of-speech tag sequences which the incremental parser most often over- or under-proposed as brackets. Since short part-of-speech tag sequences are far more frequent than longer sequences, this analysis emphasizes local errors. Such errors are not only significant in themselves, but also play a role in the errors made in larger brackets, since an incorrect common cover link affects all brackets which cover that link. Most links are short and therefore most errors are reflected both in small and large brackets (which cover those smaller brackets). It is the smaller brackets, however, which most clearly indicate what type of error was made. This is also the reason why I have only used data from sentences of up to 10 words to perform the error analysis. In general, error analysis of the full corpora resulted in qualitatively similar results (with differences which mainly seem to stem from local structure distribution differences between the short-sentence and full corpora).

Of the sequences in table 7.7, some are both frequently over-proposed and under-proposed by the parser. Such sequences are mainly fragments of noun phrases (such as DT NN in English, ART NN in German and NN NN in Chinese) which can either be a complete noun phrase or part of a noun phrase. If the sequence is a complete noun phrase and the parser creates an outbound link of depth 0 based at the sequence, the bracket covering it is destroyed, and the parser fails to propose the sequence. On the other hand, if the sequence is only part of a noun phrase and the parser fails to connect it (with back-and-forth links) to the rest of the noun phrase, the sequence is incorrectly proposed as a bracket. Precision and recall for most of these sequences are reasonably high, but because these sequences are very frequent in the corpus, they contribute significantly to the total errors made. In the English and Chinese corpora precision for these sequences is higher than recall, suggesting that in parsing English and Chinese noun phrases, the parser fails to attach the parts of a noun phrase more often than it attaches words to a noun phrase which do not belong to it. In the German corpus, the situation is reversed, with precision lower than recall for noun phrase fragments. At least partially this has to do with the annotation of the

Over-proposed	#	UP	Under-proposed	#	UR
WSJ10					
NNP NNP	220	71.8	DT NN	451	75.6
DT NN	156	89.9	NNP NNP	306	64.7
POS NN	139	0.0	NNP POS	151	5.6
JJ NN	120	66.2	JJ NNS	132	73.0
PRP VBZ	107	20.7	NN NNS	123	55.4
PRP VBD	105	16.0	DT JJ NN	114	77.5
RB VBN	90	15.9	JJ NN	106	68.9
JJ NNS	82	81.3	DT NNS	103	76.2
NN NN	64	56.2	CD CD	103	84.3
NNP NNP NNP	63	53.0	CD NNS	95	59.9
Negra10					
ART NN	946	56.2	ART NN	501	70.8
ADJA NN	376	51.2	APPR ART NN	269	50.6
CARD NN	219	35.2	APPR NN	214	57.7
ART ADJA NN	200	61.8	NE NE	184	62.6
NN VVFIN	138	0.7	ADJA NN	140	73.8
KON NN	125	0.0	ART ADJA NN	140	69.8
ART NN VVFIN	118	3.3	APPR NE	133	60.3
NN VVPP	103	5.5	NN KON NN	111	24.5
ADJA NN NE	102	1.0	APPRART NN	86	80.7
NN VAFIN	99	0.0	APPR ADJA NN	80	56.8
CTB10					
DEG NN	222	0.0	NN NN	573	31.2
NN NN	183	58.7	NR NN	257	38.5
DEC NN	132	0.0	CD M	184	37.8
NN VV	92	32.8	VV NN	164	57.7
AD VV	90	53.6	JJ NN	161	37.4
NR VV	87	3.3	NN NN NN	132	25.0
VV NN	81	73.4	NR NN NN	110	34.1
NR NN	69	70.0	AD VV	95	52.3
NN NN NN	49	47.3	NN DEG	91	1.1
JJ NN	41	70.1	P NN	88	39.7

Table 7.7: Part-of-speech sequences most over- and under-proposed as brackets by the incremental parser. For each sequence, the table gives the number of times the sequence was over- or under-proposed and the unlabeled precision (for over-proposed sequences) and recall (for under-proposed sequences). Top brackets are not included. In CTB10, utterances of type FRAG (which have no internal structure in the annotation) are ignored.

Simple tags			Compound tags		
Tag	Frequency	Recall	Tag	Frequency	Recall
WSJ10					
NP	11746	62.4	VP	8710	77.5
VP	8712	77.5	NP	6744	65.1
PP	3883	82.4	NP-SBJ	3023	54.1
S	1598	67.5	PP	1635	78.6
SBAR	656	73.3	S	1068	65.5
ADJP	634	51.1	NP-PRD	773	71.3
ADVP	343	66.8	PP-CLR	593	86.8
QP	183	35.0	NP-SBJ-1	570	49.3
SQ	94	70.2	PP-DIR	557	93.5
PRN	93	66.7	SBAR	437	71.6
Negra10					
NP	5746	58.9	NP-SB	2721	50.0
PP	3754	55.8	PP-MO	2145	52.1
VP	1330	62.7	NP-OA	1128	59.3
S	746	54.0	VP-OC	1120	61.3
AP	587	32.0	PP-MNR	1037	69.1
MPN	554	63.5	NP-GR	577	78.2
CNP	424	23.3	NP-PD	336	72.3
AVP	215	37.7	MPN-NK	279	61.3
VZ	111	97.3	PP-*T1*	257	34.2
NM	83	8.4	NP	253	79.8
CTB10					
NP	3744	33.2	VP	3435	52.7
VP	3447	52.7	NP	1129	23.2
IP	1102	37.6	NP-SBJ	1035	39.2
PP	478	32.4	NP-OBJ	822	42.5
QP	393	30.8	IP	645	39.8
DNP	376	0.3	DNP	375	0.2
CP	301	2.7	QP	263	21.7
LCP	157	53.5	CP	256	0.8
PRN	109	44.0	IP-OBJ	219	36.5
DP	77	24.7	NP-PN	192	14.6

Table 7.8: Recall accuracy of the most frequent non-terminal tags. The columns on the left give the statistics for the basic tags (e.g NP in NP-SBJ) while the columns on the right give the statistics for the full compound tags. Top brackets are not included. In CTB10, utterances of type FRAG (which have no internal structure in the annotation) are ignored.

corpus: prepositional phrases (PPs) are assigned a flat structure. For example, the PP *in den Vordergrund*, is annotated as a single constituent, without any internal structure. The parser, however, creates a constituent for *den Vordergrund*, which results in the sequence ART NN being incorrectly proposed (relative to the annotation: most linguists would argue that *den Vordergrund* is a constituent).

Table 7.8 allows us to look at the parsing accuracy of noun phrases from a different perspective. The left column of the table gives the recall accuracy for all noun phrases (NPs) in each corpus. This can be compared with the recall accuracy of subsets of noun phrases given in the right column of the table. We can see that subject noun phrases in English and in German (NP-SBJ and NP-SB) have a somewhat lower recall than the average noun phrase. This is possibly due to incorrect attachment of the noun phrase to the verb on its right (such an attachment is correct if the depth of the link is 1 but incorrect if the depth of the link is 0). In Chinese, both subject and object noun phrases (NP-SBJ, NP-OBJ) have a somewhat higher recall accuracy than the average noun phrase while it is other noun phrases (NP, NP-PN) which have lower recall. This is probably the result of the many complex compound nouns in the Chinese Treebank. The NP and NP-PN labels (as well as other labels) are used to annotated parts of these compound nouns while NP-SBJ and NP-OBJ are used to annotated the full noun phrase. While even detecting the full noun phrases is not simple for the parser, detecting their internal structure seems even harder.

A second group of over-proposed part-of-speech sequences are sequences which end in a verb. Such sequences are the result of failure by the parser to attach the verb to the argument on its right. This may be combined with attachment to the argument on the left (typically the subject) by a link of depth 0 instead of depth 1. These mistakes are frequent because of the high frequency of the part-of-speech sequences involved but they do not reflect a systematic error in the analysis. In fact, as can be seen in table 7.8, verb phrases (VPs) have a relatively high recall rate in all languages. Had the parser systematically created subject-verb constituents, VP recall would have been much lower.

One error of this type is worth mentioning specifically. In the WSJ10, the constituent PRP VBZ is often created in sentences of the form *"... " (s)he said.* The pronoun *he* (or *she*) is attached by the parser to the verb *said* but the parser fails to extend this bracket to also cover the object of the verb, which in this case precedes the subject. Because this is a very common construction in the WSJ10, it results in many errors.

The noun and verb phrase errors are the most common errors made by the parser, but at the same time, many noun phrases and verb phrases are parsed correctly. These errors, therefore, are not the result of a systematically incorrect analysis but the result of incorrect properties assigned to specific words. This may suggest that the bootstrapping mechanism, which is supposed to transfer properties from one word to the other, is too weak and fails to correct errors made in some words by using the properties assigned to other words.

A different source of errors is the incorrect attachment of certain function words and particles. In English, the possessive particle *'s* (part-of-speech tag POS) is attached to the right instead of to the left. This may be the result of the parser's failure to recognize the ambiguous nature of *'s*, which is also a contraction of the verb *is*. In German, conjunction words such as *und* (part-of-speech tag KON) are attached only to the right, while the annotation places the conjunction word together with both its conjuncts in a single bracket. Finally, in Chinese both the noun complementizer (part-of-speech DEG) and the relative clause marker (part-of-speech DEC) are attached incorrectly to the right instead of to the left.

7.4.8 Non-Binary Branching Bracketing

A bracketing $\mathcal{C}$ over an utterance U is binary branching if $U \in \mathcal{C}$ and every bracket $X \in \mathcal{C}$ is a union $X = X_1 \cup X_2$ of brackets $X_1, X_2 \in \mathcal{C}$. One fundamental difference between the incremental parser and the unsupervised parsing algorithms of Klein, Manning and Bod is that the algorithms of Klein, Manning and Bod always produce a binary branching bracketing while the incremental parser can produce any bracketing. The syntactic annotation found in the corpora is also not necessarily binary branching. This is reflected in the parsing results of the different algorithms. The binary branching parsers typically produce more brackets than are found in the annotation and therefore tend to have high recall scores and low precision. The incremental parser, on the other hand, produces typically fewer brackets and therefore its precision and recall values are closer to each other (and to their F_1 average).

There are two reasons to require that the parser produce only binary branching structures. The first reason is practical - restricting the range of possible structures the parser and learning algorithm need to consider. The second is theoretical - some linguistic theories claim that all linguistic structures must be binary branching (e.g. chapter 2 section 5 of Haegeman 1994). If one accepts the assumption that syntactic structure must be binary branching then the annotation of the corpora is shallow - it does not give the full syntactic structure, since it is not binary branching.

While a full discussion of this question is beyond the scope of this chapter, it is interesting to look at table 7.9, where the number of brackets proposed by different parsing algorithms is compared with the number of brackets annotated in the different corpora. Since all binary branching bracketings of a sentence have the same number of brackets, a single entry in the table is sufficient to describe the number of brackets produced by all parsers which produce binary branching bracketings (CCM, DMV, DMV+CCM, U-DOP and UML-DOP). The last entry in the table describes the effect of punctuation on binary branching bracketing: a binary branching bracketing is generated between stopping punctuation symbols. This is the number of brackets generated for the right-branching heuristic with punctuation.

	WSJ10	Full WSJ	Negra10	Full Negra
Annotation	35302	730024	20201	141977
Incremental	35588 (1.01)	691179 (0.95)	27624 (1.37)	199563 (1.41)
Simplified Inc.	35696 (1.01)	694337 (0.95)	29268 (1.45)	219556 (1.55)
Binary	44826 (1.27)	979146 (1.34)	35833 (1.77)	282882 (1.99)
Binary+Punct.	44430 (1.26)	947202 (1.30)	35651 (1.76)	274080 (1.93)

	CTB10	Full CTB
Annotation	14229	330079
Incremental	14468 (1.02)	247366 (0.75)
Simplified Inc.	16399 (1.15)	299996 (0.91)
Binary	19845 (1.39)	412190 (1.25)
Binary+Punct.	19684 (1.38)	380645 (1.15)

Table 7.9: Number of brackets annotated in each corpus and the number of brackets produced by different parsing algorithms methods. Brackets covering a single word are not included. The number in parentheses is the ratio with the number of brackets found in the annotation.

Since a binary branching bracketing is a maximal set of non-crossing brackets, it is not surprising to see that both the annotation and the incremental parses contain fewer brackets than the binary branching bracketing. It is interesting, though, that the incremental parser produces significantly fewer brackets than a binary branching bracketing restricted by punctuation. This shows (again) that the incremental parser does not merely produce a variant of the right-branching with punctuation heuristic. Finally, on the Wall Street Journal Corpus, where the incremental parser does best, it produces more or less the same number of brackets as in the corpus annotation. The incremental parser captures, therefore, the same depth of syntactic structure as that used by the corpus annotators. Since the incremental parser is completely unsupervised, this suggests that the corpus annotation does capture some natural level of syntactic structure. This is not necessarily the full syntactic structure but it has its distinct defining properties which may be inferred from the distributional properties of the text.

7.4.9 Parsing Speed

One of the advantages of the incremental parser is that it is very fast. Table 7.10 gives the time the incremental parser required for learning and parsing the different corpora on a standard Intel Centrino laptop (1.86GHz). Because different variants of the algorithm required more or less the same amount of time, the time is reported only for the basic parsing function family (parsing from left to right). Learning was always performed from the full corpus and therefore no

	Learning		Parsing	
Corpus	total (sec.)	words/sec.	total (sec.)	words/sec.
WSJ10			12	4354
Full WSJ	321	3203	262	3925
Negra10			10	4319
Full Negra	83	3656	71	4274
CTB10			6	4078
Full CTB	132	3264	105	4104

Table 7.10: Learning and parsing time for the incremental parser using the basic parsing function family.

execution time is reported for learning on subsets of the corpus. From the table it can be seen that the word per second rate changes very little as the length of the sentences increases. This suggests that the parsing algorithm is linear (or close to linear) in the length of the sentence. Because sentences are parsed as part of the learning process, learning from a full corpus is somewhat slower than parsing a full corpus. However, the difference in speed is not very significant, which is not surprising since learning is nothing but a simple update of the adjacency point statistics during the parsing process.

7.5 Conclusion

The experiments described in this chapter allow a first rough analysis of the incremental parser's ability to capture the syntactic structure of different languages. This was done by measuring the accuracy with which the parser succeeded in reproducing the syntactic annotation of English, German and Chinese corpora. A comparison of these results with a threshold for success based on the right branching with punctuation baseline showed that the incremental parser was successful in capturing some of the syntactic structure of all three languages. Parsing was most successful on English and least successful on Chinese, with German somewhere in-between.

Because the parser is incremental, parsing from left to right is not the same as parsing from right to left. The incremental parser was therefore tested on parsing the test corpora in both directions. The differences in performance were small on all corpora, showing that the incremental parser is not inherently biased towards right-branching structures and can handle left-branching languages. While the differences in performance were small, parsing backwards (from right to left) resulted in somewhat lower scores than parsing forward (from left to right) on all corpora. This should not be surprising since languages are typically used in the forward direction and the mirror image of a language may violate certain basic properties of natural languages (such as the possibility to process it incrementally). Left-branching languages are not simply mirror images of right-branching languages.

Comparison with other unsupervised parsing algorithms proposed in recent years is only possible in a rather rough way because most of these parsers were not tested on parsing from plain text. The comparison shows that the incremental parser achieves an accuracy comparable with that of the best unsupervised parsers available today and its high efficiency allows it to handle unrestricted real language corpora, something which is more of a challenge for other methods. It is also clear that the parses produced by the incremental parser are different from those produced by other unsupervised parsers. This suggests that the different algorithms have different strengths and weaknesses and opens the way to combining the strengths of the different parsers to achieve higher parsing accuracy.

It remains tempting to compare the output of the parsing algorithm on many different unannotated corpora in an attempt to discover syntactic invariants. This has the same appeal as the purely distributionalist approach: it frees one from the variety and arbitrariness of annotation schemes and the linguistic theories they are based on. Moreover, it allows one to use the algorithm for languages and domains where little or no annotated resources are available. On the other hand, it is completely uncharted waters. It is not at all clear what should count as invariance across different corpora and domains. Moreover, even if one can discover such invariants, it is not clear how one can show that these are non-trivial and syntactic in nature. For these reasons, I have chosen not to explore this possibility in the present work. However, I do find it an interesting question for future research.

Chapter 8
Conclusions

On the way to constructing a learning algorithm for syntactic structure, this thesis introduced three main new components: a syntactic representation, an incremental parser and a learning algorithm. While each component serves as the foundation for the next one, it is also interesting in its own right.

The common cover link representation of syntactic structure shares many of the basic properties of dependency links. At the same time, it allows structures which dependencies do not allow. These include exocentric constructions and structures of a lesser degree of skewness than that of dependency structures. Most crucially, the common cover links represent a relation which is weaker than that defined by dependencies and is similar to that which has been proposed in various psycholinguistic models. For the purpose of the present work, the main advantage of this weaker relation is that it allows incremental parsing, but if it indeed models cognitive processing more accurately than dependencies then this may have additional consequences for syntactic analysis. These properties are not stipulated but are logical consequences of a few simple definitions. This simplicity makes this representation a good starting point for future extensions.

The incremental parser which is defined for the common cover link representation is also an interesting contribution in its own right. It provides a simple and efficient framework for incremental parsing. One specific instance of this parsing model was developed in this thesis, but many other algorithms, supervised as well as unsupervised, may be designed to perform parsing within this framework. The design of the parser is dictated in large part by the definition of incrementality and the properties of the representation which must be preserved at each step of the parse. As with the representation, the small number of arbitrary decisions in the design may make it a good starting point for future extensions.

At the heart of the third component, the learning algorithm, lies a simple labeling scheme, which constructs the labels not only based on adjacent words but also on the labels of those words. These sets of labels can be seen as providing a transformation of the original corpus, and very simple statistics on this transformation are all that is used to deduce the syntactic structure. To convert these statistics into parsing decisions, the best matching labels between words are used to determine the links between those words. By using the best matching label, two common problems in parsing are solved together: ambiguity and infrequent words. While many of the details of the learning algorithm must be refined, I believe that the labeling scheme and the use of the best matching label are sound methods which can remain at the heart of future extensions of the algorithm.

Linguistically, the subject of this thesis is the discovery of relationships between the hidden syntactic structure of a language and the statistics over its surface structure. To make such a relationship visible, both the syntactic representation and the surface structure statistics must be chosen carefully. Applying a transformation on either side, the statistics or the syntactic representation, may reveal relationships which would otherwise remain hidden.

While working on this thesis, I have spent much time examining the statistics of corpora through various transformations, eventually converging onto the labeling transformation presented here. When it became apparent that the statistics over the label transformation resemble the syntactic properties of the language, I began to adjust the syntactic representation. This was a back and forth process, working at both ends, the statistics and the representation, until they finally met. I can hardly pretend that the two sides match perfectly, and the process should be continued with more data and patience. This is, I feel, the essence of the empirical method, which remains close to the data and is driven by what it observes.

Bibliography

Abney, S. P. (1989). A computational model of human parsing. *Journal of Psycholinguistic Research 18*(1), 129–144. [73]

Adriaans, P. W., M. Trautwein, and M. R. Vervoort (2000). Towards high speed grammar induction on large text corpora. In G. Hlavac, V. Feffrey, and J. Wiederman (Eds.), *SOFSEM 2000: Theory and practice of Informatics*, Volume 1963 of *Lecture Notes in Computer Science*, pp. 173–186. Springer. [12, 14]

Altenberg, B. (1987). *Prosodic patterns in spoken English: studies in the correlation between prosody and grammar*. Lund University Press. [168]

Anderson, J. R. (1977). Induction of augmented transition networks. *Cognitive Science 1*(2), 125–157. [10]

Baker, J. K. (1979). Trainable grammars for speech recognition. In J. J. Wolf and D. H. Klatt (Eds.), *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pp. 547–550. [16]

Beavers, J. (2003). More heads and less categories: A new look at noun phrase structure. In S. Müllers (Ed.), *Proceedings of the HPSG-2003 Conference, Michigan State University*, pp. 47–67. CSLI Publications. `http://cslipublications.stanford.edu/HPSG/4/`. [67]

Bever, T. G. (1970). The cognitive basis for linguistic structures. In J. R. Hayes (Ed.), *Cognition and the Development of Language*, pp. 279–362. Wiley. [22]

Bod, R. (2006a). An all-subtrees approach to unsupervised parsing. In *Proceedings of the International Conference on Computational Linguistics/Association for Computational Linguistics (COLING-ACL 2006)*, pp. 865–872. [8, 17, 19, 20, 149, 174, 175, 177, 179, 180, 186]

Bod, R. (2006b). Unsupervised parsing with U-DOP. In *Proceedings of the 10th Conference on Natural Language Learning*, pp. 85–92. [8, 17, 19, 20, 149, 174, 175, 177, 179, 186]

Bod, R. (2007a). Is the end of supervised parsing in sight? In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 400–407. [17, 19, 20, 186]

Bod, R. (2007b). Unsupervised syntax-based machine translation: The contribution of discontiguous phrases. In *Proceedings of Machine Translation Summit XI.* [186]

Carroll, G. and E. Charniak (1992). Two experiments on learning probabilistic dependency grammars from corpora. In C. Weir, S. Abney, R. Grishman, and R. Weischedel (Eds.), *Working Notes of the Workshop Statistically-Based NLP Techniques*, Menlo Park, California, pp. 1–13. AAAI Press. [16]

Chomsky, N. (1965). *Aspects of the Theory of Syntax.* MIT Press. [4]

Chomsky, N. (1995). *The Minimalist Program.* MIT Press. [200]

Chomsky, N. and H. Lasnik (1993). The theory of principles and parameters. In J. Jacobs, A. von Stechow, W. Sternefeld, and T. Vannemann (Eds.), *Syntax: An International Handbook of Contemporary Research*, pp. 506–569. Walter de Gruyter. Reprinted in Chomsky (1995). [4]

Church, K. (1980). On parsing strategies and closure. In *Proceedings of the 18nd Annual Meeting of the Association for Computational Linguistics*, pp. 107–111. [22]

Clark, A. (2000). Inducing syntactic categories by context distribution clustering. In *Proceedings of the 4th Conference on Natural Language Learning*, pp. 91–94. [12, 148]

Clark, A. (2001). Unsupervised induction of stochastic context-free grammars using distributional clustering. In *Proceedings of the 5th Conference on Natural Language Learning*, pp. 105–112. [12, 13, 14, 20]

Clark, S. and J. R. Curran (2004). Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pp. 103–110. [22]

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing.* Ph. D. thesis, University of Pennsylvania. [152]

Cook, C. M., A. Rosenfeld, and A. R. Aronson (1976). Grammatical inference by hill climbing. *Informational Sciences (now Information Sciences) 10*, 59–80. [12, 14, 15]

Crocker, M. W., M. Pickering, and C. Clifton (2000). *Architectures and Mechanisms for Language Processing.* Cambridge University Press. [21, 47]

Croft, W. (1995). Intonation units and grammatical structure. *Linguistics 33*, 839–882. [168]

de la Higuera, C. (2005). A bibliographical study of grammatical inference. *Pattern Recognition 38*(9), 1332–1348. [6]

Deacon, T. W. (1997). *The Symbolic Species: The Co-evolution of Language and the Brain*. W.W. Norton. [3]

Dennis, S. (2005). An exemplar-based approach to unsupervised parsing. In *Proceedings of CogSci 2005*. [8, 20]

Dudau-Sofronie, D., I. Tellier, and M. Tommasi (2001). From logic to grammar via types. In L. Popelínský and M. Nepil (Eds.), *Proceedings of the Third Learning Language in Logic Workshop*, pp. 35–46. [10]

Gleitman, L. R. (1990). The structural sources of verb meanings. *Language Acquisition 1*(1), 3–55. [9]

Gold, E. M. (1967). Language identification in the limit. *Information and Control 10*, 447–474. [5]

Goodall, G. (1987). *Parallel Structures in Syntax: Coordination, Causatives and Restructuring*. Cambridge University Press. [77, 83]

Gorrell, P. (1995a). Japanese trees and the garden path. In R. Mazuka and N. Nagai (Eds.), *Japanese Sentence Processing*, pp. 331–350. Lawrence Erlbaum Associates. [22, 74]

Gorrell, P. (1995b). *Syntax and Parsing*. Cambridge University Press. [22, 74]

Gregory, M., M. Johnson, and E. Charniak (2004). Sentence-internal prosody does not help parsing the way punctuation does. In D. M. Susan Dumais and S. Roukos (Eds.), *HLT-NAACL 2004: Main Proceedings*, Boston, Massachusetts, USA, pp. 81–88. Association for Computational Linguistics. [167, 168]

Haegeman, L. M. V. (1994). *Introduction to Government and Binding Theory* (2nd ed.). Blackwell. [29, 65, 67, 192]

Hamburger, H. and K. Wexler (1975). A mathematical theory of learning transformational grammar. *Journal of Mathematical Psychology 12*(2), 137–177. [10]

Harris, Z. (1946). From morpheme to utterance. *Language 22*(3), 161–183. [8, 11]

Hays, D. G. (1964). Dependency theory: A formalism and some observations. *Language 40*(4), 511–525. [62]

Hockenmaier, J. and M. Steedman (2002). Generative models for statistical parsing with combinatory categorial grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 335–342. [22]

Hoekstra, H., M. Moortgat, B. Renmans, M. Schouppe, I. Schuurman, and T. van der Wouden (2003). CGN Syntactische Annotatie. `http://lands.let.kun.nl/cgn/doc_Dutch/topics/version_1.0/annot/syntax/syn_prot.pdf`. [61]

Huck, G. and J. Goldsmith (1996). *Ideology and Linguistic Theory*. London: Routledge. [172]

Hudson, R. A. (1987). Zwicky on heads. *Journal of Linguistics 23*, 109–132. [68, 69]

Hudson, R. A. (1990). *English Word Grammar*. Basil Blackwell. [60, 62, 77]

Hudson, R. A. (2003). An encyclopedia of english grammar and word grammar. online encyclopedia. `http://www.phon.ucl.ac.uk/home/dick/enc-gen.htm`. [62, 82]

Ingram, D. (1989). *First Language Acquisition: Method, Description and Explanation*. Cambridge University Press. [3]

Johnson, K. (2004). Gold's theorem and cognitive science. *Philosophy of Science 71*, 571–592. [6]

Jurafsky, D. and J. H. Martin (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall. [11]

Kanazawa, M. (1998). *Learnable Classes of Categorial Grammars*. CSLI Publication. [6]

Kirby, S. and J. R. Hurford (2002). The emergence of linguistic structure: an overview of the iterated learning model. In A. Cangelosi and D. Parisi (Eds.), *Simulating the Evolution of Language*, pp. 121–148. Springer. [3]

Klein, D. (2005). *The Unsupervised Learning of Natural Language Structure*. Ph. D. thesis, Stanford University. [167, 174, 176, 181, 183, 186, 187]

Klein, D. and C. D. Manning (2002). A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 128–135. [8, 16, 17, 18, 20, 174, 175, 178, 186]

Klein, D. and C. D. Manning (2004). Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pp. 478–485. [8, 17, 18, 20, 63, 149, 152, 174, 175, 177, 179, 181, 183, 186]

Lamb, S. M. (1961). On the mechanization of syntactic analysis. In *1961 Conference on Machine Translation of Languages and Applied Language Analysis (National Physical Laboratory Symposium No. 13)*, Volume II, pp. 674–685. Her Majesty's Stationery Office, London. [6, 12]

Landau, B. and L. R. Gleitman (1985). *Language and Experience: Evidence from the Blind Child*. Harvard University Press. [9]

Langacker, R. W. (1987). *Foundations of Cognitive Grammar Vol. 1: Theoretical Prerequisites*. Stanford University Press. [5]

Langacker, R. W. (1991). *Foundations of Cognitive Grammar Vol. 2: Descriptive Application*. Stanford University Press. [5]

Lari, K. and S. J. Young (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language 4*, 35–56. [16]

Lecerf, Y. (1960). Programme des conflits, modèle des conflits. *La traduction automatique : bulletin trimestriel de l'Association pour l'Étude et le Développement de la Traduction Automatique et de la Linguistique Appliquée (ATALA) 1*(4,5), 11–20,17–36. [62]

Lee, L. (1996). Learning of context-free languages: a survey of the literature. Technical Report TR-12-96, Harvard University, Center for Research in Computing Technology. [6]

Lewis, R. L. (1993). *An Architecturally-based Theory of Human Sentence Comprehension*. Ph. D. thesis, Carnegie Mellon University. [73]

Marcus, M. P., D. Hindle, and M. M. Fleck (1983). D-theory: Talking about talking about trees. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, Morristown, NJ, USA, pp. 129–136. Association for Computational Linguistics. [73]

Marcus, M. P., B. Santorini, and M. A. Marcinkiewicz (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguisitics 19*(2), 313–330. [174]

McDonald, R., K. Crammer, and F. Pereira (2005, June). Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, Ann Arbor, Michigan, pp. 91–98. Association for Computational Linguistics. [63]

Mel'čuk, I. A. (1988). *Dependency Syntax: Theory and Practice*. Albany State University of New York Press. [60, 61, 82]

Moltmann, F. (1992). *Coordination and Comparatives*. Ph. D. thesis, Massachusetts Institute of Technology. [77, 83]

Mori, S. and M. Nagao (1995). Parsing without grammar. In *Proceedings of the 4th International Workshop on Parsing Technologies*, pp. 174–185. [12, 13, 14, 20]

Muadz, H. (1991). *Coordinate Structures: A Planar Representation*. Ph. D. thesis, University of Arizona. [77, 83]

Nivre, J. (2004). Incrementality in deterministic dependency parsing. In F. Keller, S. Clark, M. Crocker, and M. Steedman (Eds.), *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, Barcelona, Spain, pp. 50–57. Association for Computational Linguistics. [49]

Nivre, J. and M. Scholz (2004, Aug 23–Aug 27). Deterministic dependency parsing of english text. In *Proceedings of COLING 2004*, Geneva, Switzerland, pp. 64–70. [63]

Oates, T., T. Armstrong, J. Harris, and M. Nejman (2003). Leveraging lexical semantics to infer context-free grammars. In C. de la Higuera, P. W. Adriaans, M. van Zaanen, and J. Oncina (Eds.), *ECML Workshop on Learning Contex-Free Grammars*, pp. 65–76. [10]

Paskin, M. A. (2002). Grammatical bigrams. In T. G. Dietterich, S. Becker, and Z. Ghahramani (Eds.), *Advances in Neural Information Processing Systems*, Volume 14, Cambridge, MA, pp. 91–97. MIT Press. [17]

Pereira, F. and Y. Schabes (1992). Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pp. 128–135. [16]

Pinker, S. (1979). Formal models of language learning. *Cognition 7*(3), 217–283. [12]

Pinker, S. (1996/1984). *Language Learnability and Language Development.* Harvard University Press. [4, 5, 9]

Potter, S. (1950). *Our Language.* Penguin. [68]

Prescher, D. (2005). Head-driven PCFGs with latent-head statistics. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Vancouver, British Columbia, pp. 115–124. Association for Computational Linguistics. [187]

Pritchett, B. L. (1992). *Grammatical Competence and Parsing Performance.* University of Chicago Press. [73]

Sakakibara, Y. (1997). Recent advances of grammatical inference. *Theoretical Computer Science 185*(1), 15–45. [6]

Schabes, Y., M. Roth, and R. Osborne (1993). Parsing the Wall Street Journal with the inside-outside algorithm. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics (EACL 93)*, pp. 341–347. [16]

Schütze, H. (1995). Distributional part-of-speech tagging. In *Proceedings of the Seventh Conference of the European Chapter of the Association for Computational Linguistics (EACL 95)*, pp. 141–148. [12, 148, 176]

Seginer, Y. (2007). Fast unsupervised incremental parsing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 384–391. [26]

Skut, W., B. Krenn, T. Brants, and H. Uszkoreit (1997). An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing ANLP-97*, Washington, DC, pp. 88–95. [174]

Smith, N. A. and J. Eisner (2005). Guiding unsupervised grammar induction using contrastive estimation. In *Proceedings of the IJCAI Workshop on Grammatical Inference Applications*, pp. 73–82. [17]

Smith, N. A. and J. Eisner (2006). Annealing structural bias in multilingual weighted grammar induction. In *Proceedings of the International Conference on Computational Linguistics/Association for Computational Linguistics (COLING-ACL 2006)*, pp. 569–576. [17, 20]

Solan, Z., D. Horn, E. Ruppin, and S. Edelman (2005). Unsupervised learning of natural languages. *Proceedings of the National Academy of Sciences 102*, 11629–11634. [12, 13, 14]

Solomonoff, R. J. (1964). A formal theory of inductive inference. *Information and Control 7*, 1–22,224–254. [14]

Steedman, M. (2000). *The Syntactic Process.* MIT Press. [22]

Stolcke, A. (1994). *Bayesian Learning of Probabilistic Language Models.* Ph. D. thesis, University of California at Berkeley. [15, 20]

Sturt, P. and M. W. Crocker (1996). Monotonic syntactic processing: A cross-linguistic study of attachment and reanalysis. *Language and Cognitive Processes 11*(5), 449–492. [22, 72, 74]

Sturt, P., M. J. Pickering, and M. W. Crocker (1999). Structural change and reanalysis difficulty in language comprehension. *Journal of Memory and Language 40*(1), 136–150. [73]

Tellier, I. (1998). Meaning helps learning syntax. In V. Honavar and G. Slutzki (Eds.), *Grammatical Inference, 4th International Colloquium, ICGI-98*, Volume 1433 of *Lecture Notes in Computer Science*, pp. 25–36. Springer. [10]

Tomasello, M. (2003). *Constructing a Language: a Usage-Based Theory of Language Acquisition.* Harvard University Press. [5]

Tomasello, M. (2006). Acquiring linguistic constructions. In D. Kuhn and R. Siegler (Eds.), *Handbook of Child Psychology*, pp. 255–298. New York: Wiley. [5]

Tomasello, M. and E. Bates (2001). *Language Development: The Essential Readings.* Blackwell. [3]

Valiant, L. G. (1984). A theory of the learnable. In *STOC '84: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, New York, NY, USA, pp. 436–445. ACM Press. [6]

van Zaanen, M. (2000). ABL: Alignment-based learning. In *Proceedings of the 18th International Conference on Computational Linguistics*, pp. 961–967. [12, 14]

Weinberg, A. (1993). Parameters in the theory of sentence processing: Minimal commitment theory goes east. *Journal of Psycholinguistic Research 22*(3), 339–364. [22, 73]

Weinberg, A. (1995). Licensing constraints and the theory of language processing. In R. Mazuka and N. Nagai (Eds.), *Japanese Sentence Processing*, pp. 235–255. Lawrence Erlbaum Associates. [22, 74]

Wolff, J. G. (1982). Language acquisition, data compression and genralization. *Language & Communication 2*(1), 57–89. [12, 14, 15]

Xue, N., F.-D. Chiou, and M. Palmer (2002). Building a large-scale annotated Chinese corpus. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING-2002)*, Taipei, Taiwan. [174]

Xue, N. and F. Xia (2000). The bracketing guidelines for the Penn Chinese treebank (3.0). Technical Report 00-08, IRCS. [77]

Yuret, D. (1998). *Discovery of Linguistic Relations Using Lexical Attraction.* Ph. D. thesis, MIT. [17, 21, 25]

Zollmann, A. and K. Sima'an (2005). A consistent and efficient estimator for data-oriented parsing. *Journal of Automata, Languages and Combinatorics 10*(2/3), 367–388. [19]

Zuidema, W. (2003). How the poverty of the stymulus solves the poverty of the stymulus. In S. Becker, S. Thrun, and K. Obermayer (Eds.), *Advances in Neural Information Processing Systems 15 (Proceedings of NIPS'02)*, pp. 51–58. MIT Press. [3]

Zwicky, A. M. (1985). Heads. *Journal of Linguistics 21*, 1–29. [69]

Index

Entries which refer to numbered statements, such as definitions, lemmas and algorithms, give the number of the statement followed by the page number on which the statement appears (e.g. *1.2.3* 56).
Author names are not listed in the index and the reader is referred to the bibliography, where the page numbers on which each bibliographical entry is cited are given at the end of each entry (in square brackets).

$\mathcal{A}(L)$, 34, 99
$\mathcal{A}(P(S_k))$, 122
$\#(A)$, 151
$\bar{A}$, 151
${}_\bullet A_i^\alpha, {}_\bullet A_i^x$, 159, 161–162
A_i^w, 151
$B_d^\mathcal{C}(x), B_d(x)$, 31, 88
$G_d^k(x)$, *5.4.2* 123
$\mathcal{I}_k(S)$, 122
In, In^*, 150, 158, *6.3.1* 159, 161–162
$\mathcal{L}$, 151
$L(W)$, 150
Out, 150, 158, *6.3.1* 159, 161–162
P, 150
$P(L)$, 111, *see also* link sets $P^*(L)$ and $P(L)$
$P^*(L)$, 111, *see also* link sets $P^*(L)$ and $P(L)$
$R_\mathcal{C}$, *2.3.2* 33, *5.1.6* 89
$S(R)$, 93
$Sign(i)$, 159
S_k, 50, 122, 132
$Stop$, 158, *6.3.1* 159, 161
U_k, 49, 132
$d_B^\mathcal{C}(x), d_B(x)$, 31, 88
l^{-1}, 150
$(w,0), (w,1)$, 150
$[w], [w_\_], [\_w]$, 150
$x \overset{d}{\rightleftarrows} y \in L$, 33
$x \xrightarrow{L} y$, 41, 97
$x \xrightarrow{d,L} y$, 41, 97
$x \xrightarrow{d} y$, 33, 89
$x \rightarrow y \in L$, 33
$x_k(U), x_k$, 49, 132
$x \rightarrow y \notin L$, 33
$[x,y], [x,y), (y,x]$, 31, 88
$x \dashv_d^L y, x \dashv^L y$, 51, *3.2.1* 51, *5.5.2* 133, 134

adjacency, *3.2.1* 51, 51–53, 70–72, *5.5.2* 133, 133–140
  direct, *see* direct adjacency
  empty, *see* empty adjacency

unused, *see* unused adjacency
adjacency label, 150
adjacency point, 150
adjacency point count, 151
adjacency point update, *6.3.1* 159, 158–162
adjacency position, 151
adjacency properties, 51, 133
ambiguity, 22, 72–73, 163

back and forth equality, *5.1.14* 91
in $P^*(L)$, *5.3.15* 114
base of a common cover link, *2.3.1* 33, *5.1.5* 89
baseline, 177–184
basic parsing function family, 158–166, 179
simplified, 167, 179
best matching label, 163–164
binary branching, 17, 192–193
blocking, *2.7.1* 43, 44, 45, 50, 51, *5.3.1* 106, 107, 108, 133
bootstrapping, 25, 156–159
semantic, *see* semantic bootstrapping
syntactic, *see* syntactic bootstrapping
bracket, 31, 88
covering, 31, 88
non-crossing, 31, 88
bracket generator, *5.4.2* 123, 123–126
leftmost, *5.4.2* 123
bracket height, *2.2.4* 32, *5.1.4* 88
bracket reconstruction
from $R_{\mathcal{C}}$, *2.3.3* 34, *5.1.29* 98, 98–100
general case, *5.2.1* 101, 100–105
incremental, *2.6.5* 42, 42–43, *5.4.1* 121, 121–132
simple, 39–41
bracketing, *2.2.1* 31, *5.1.1* 88
basic properties, 89–90

candidate path links, *see* link sets $P^*(L)$ and $P(L)$
CCM, 17, 18, 179–182, 186–187, 192
CFG, *see* context free grammar
child language acquisition, 2–5
Chinese
preverbal modifiers, 77–78
Treebank, 76–80, 174, 182
verb compounds, 78–79
class label, 150
common cover link, *2.3.1* 33, *5.1.5* 89
common cover links
basic properties, 90–92
of a bracketing, *2.3.2* 33, *5.1.6* 89
computational learning theory, 5–6
connectedness, *2.7.1* 43, 44, 50, 51, *5.3.1* 106, 107, 108, 133
of $R_{\mathcal{C}}$, *5.1.12* 90
of $P(L)$, *5.3.11* 112
constituent, 7, 13
context free grammar, 10
induction, 11–15
coordination, 82–84
coordinator-less, 76–77
covering link, *3.2.2* 52, *5.5.3* 134
crossing links, *5.1.17* 92
crossing transitivity of $P^*(L)$, *5.3.17* 115
CTB, *see* Chinese Treebank
CTBX, 175
currency symbols, 174–175

D-theory, 73–74
dependencies, 60–63
and common cover links, 63–72
external, 65–66
internal, 65
no-loop conditions, 61–62
projective, *see* projectivity
depth
0 or 1, 28, 32, 39, 42, 43, 47, 66, 75–81, 95, 105
in a bracketing, *2.2.2* 31, *5.1.2* 88

of a common cover link, *2.3.1* 33, 69–70, *5.1.5* 89
direct adjacency, 151, 155, 162, 163
direct domination, 73–74
directional uniqueness, *5.1.25* 96, 95–97
distributional clustering, 11–15, 20–21
distributional methods, 8, 172
DMV, 17, 18, 179–182, 192
DMV+CCM, 179–182, 186–187, 192
domination, 73–74

empty adjacency, *6.2.1* 152, 152–153
empty symbol, 153
equality, *2.7.1* 44, 50, 53, *5.3.1* 106, 107, 109, 136, *see also* back and forth equality
error analysis, 188–192
exocentric constructions, 66–69
experiments, 173–195

forcing, 53, 136

garden path, 72–74
grammar induction, 5–6, 11–16, 20–21

head, 32
of a common cover link, *2.3.1* 33, *5.1.5* 89
of a dependency, 60

incremental parser, 24, *3.3.1* 54, *5.5.8* 137, 136–146
complexity, 145–146
deterministic, *3.4.2* 56
speed, 193–194
incremental parsing, *3.1.1* 50, 47–50, *5.5.1* 133, 149–150
deterministic, 147
incrementality, 21–22
incrementally addable link, *3.2.5* 53, *5.5.6* 136

inside-outside algorithm, 16

label, 150, 160–161
adjacency, *see* adjacency label
best matching, *see* best matching label
class, *see* class label
opposite, *see* opposite labels
labels, 147
language acquisition, 2, *see also* child language acquisition
language learning, 2
learning process, 151–152
left branching, 178, 179
lexicon, 150–151
lexicon update, *6.2.2* 155
linear path, *2.6.3* 41, *5.1.26* 97
linear transitivity, *2.5.1* 36, *5.1.16* 92, 103–105
of $P^*(L)$, *5.3.16* 114
link path, *2.6.4* 41, *5.1.27* 97
link sets $P^*(L)$ and $P(L)$, *5.3.6* 111, 110–120
back and forth equality, *5.3.15* 114
bracketing from $P(L)$, 118–120
connectedness, *5.3.11* 112
crossing transitivity, *5.3.17* 115
linear transitivity, *5.3.16* 114
monotonicity, *5.3.8* 112
resolution, *5.3.18* 116
uniqueness, *5.3.8* 111
linking properties, 150

minimality, *2.7.1* 43, 44, 50, 51, 106, *5.3.1* 106, 108, 133
monotonicity, *2.7.1* 43, 44, 50, 51, 106, *5.3.1* 106, 108, 133
of $P^*(L)$, *5.3.8* 112

Negra Corpus, 76–77, 174, 181
Negra$X$, 175

opposite labels, 150
over-proposed brackets, 189

parser induction, 20–21
parsing function, *3.4.1* 55, 147
  greedy, *6.1.2* 149, 148–150
  lexicalized, 147, *6.1.3* 151, 150–151
parts of speech, 24, 147, 148, 175, 186–187
  induction, 12–13
path links, *see* link sets $P^*(L)$ and $P(L)$
paths, 41, 97–98
PCFG, *see* probabilistic context free grammar
phrase-based parsing, 30–31
precision, 177
prefix independence, *6.2.4* 157, 156–158
probabilistic context free grammar, 11
  highest likelihood, 16
projectivity, 62–63, 81–82
punctuation, 152, 167–169, 174–176, 178, 185–186

reanalysis, 72–75
recall, 177
  by non-terminal, 190
relative clause, 61–62, 79, 84–85
representative subset, *2.4.1* 35, 37–38, *5.1.18* 93
  minimal, *2.6.1* 40, *5.1.20* 94, 93–95
resolution, *2.7.1* 44, 45, 50, 54, *5.3.1* 106, 107, 109
  in $P(L)$, *5.3.18* 116
resolution violation, 50, *3.2.6* 54, *5.5.7* 137, 140–142
right branching, 17, 178, 179
right to left parsing, 179–182, 184

semantic bootstrapping, 9–10
shortest common cover link set, *2.5.2* 37, *5.1.19* 93
  characterization, *2.7.1* 43, 43–46, *5.3.1* 106, 105–121
skewness, 22–23, *see also* depth 0 or 1
syntactic bootstrapping, 10

transitivity
  crossing, *see* crossing transitivity of $P^*(L)$
  linear, *see* linear transitivity
  of $R_{\mathcal{C}}$, *5.1.15* 91

U-DOP, UML-DOP, 17, 19, 179–182, 192
U-DOP*, 186
under-proposed brackets, 189
unsupervised parsing, 7–9
unused adjacency, *3.2.1* 51, *5.5.2* 133, 152–154
update position, 153–156
utterance, 31, 88

Wall Street Journal Corpus, 76–77, 79–80, 174–175, 180
weight function, 148, *6.1.1* 149, 162–166
word of minimal depth, *2.2.3* 32, *5.1.3* 88
word order, 85
word token, 31, 88
word type, 31, 88
WSJ, *see* Wall Street Journal Corpus
WSJX, 175

Zipfian distribution, 24–25, 172

Samenvatting

De syntactische structuur van taal is niet direct observeerbaar maar speelt een belangrijke rol in de taalkunde. De vraag die dit proefschrift probeert te beantwoorden is wat de relatie is tussen de observeerbare taal (de woorden en zinnen die wij horen en lezen en hun frequenties) en de syntactische structuur van taal. Dit is een belangrijke vraag in de taalkunde omdat ze sterk verbonden is met fundamentele vragen over de structuur van taal en de manier waarop kinderen hun moedertaal leren. Een manier om deze vraag te beantwoorden is een algoritme te ontwerpen dat data aangaande zinnen in een taal bijhoudt en deze data gebruikt om de syntactische structuur van zinnen in de taal te bepalen. Dit proces *leert* de syntactische structuur van een taal aan de hand van niet-geannoteerde voorbeelden (voorbeelden zoals ze in de taal voorkomen zonder extra informatie). Het algoritme codeert een relatie tussen de bijgehouden data van de observeerbare taal en de syntactische structuur. Als het algoritme tenminste een deel van de syntactische structuur van een taal weet te bepalen kunnen we zeggen dat het algoritme een benadering is van de relatie tussen de observeerbare taal en haar syntactische structuur. Zo'n algoritme heet een *unsupervised parser* (letterlijk: niet-begeleide ontleder). Dit proefschrift gaat over een voorstel voor een bepaalde unsupervised parser. Door de parser op corpora van verschillende talen te testen wordt aangetoond dat het algoritme een deel van de syntactische structuur van deze talen weet te ontdekken.

De relatie beschreven door de unsupervised parser is niet alleen afhankelijk van de keuze van welke data worden bijgehouden maar ook van de keuze van een bepaalde representatie van de syntactische structuur. De juiste representatiekeuze is om die reden belangrijk voor het vereenvoudigen van de parser. Het eerste deel van het proefschrift beschrijft een volledig nieuwe representatie van syntactische structuur (*common cover links*) en een parseermethode geschikt voor deze nieuwe representatie.

De common cover links maken het gemakkelijk voor de parser gebruik te maken van twee belangrijke eigenschappen van natuurlijke taal die ik vooronderstel:

taal wordt door mensen incrementeel verwerkt en de syntactische structuren van taal zijn scheef (elke deelboom van een ontledingsboom heeft een korte tak). Door het gebruik van common cover links kan er een incrementele parser worden gedefinieerd die de syntactische structuur geleidelijk opbouwt terwijl de woorden van een zin een-voor-een worden ingelezen. Deze representatie zorgt er ook voor dat alleen scheve syntactische bomen kunnen worden geproduceerd door de parser. Als gevolg hiervan is het aantal mogelijkheden dat de parser in beschouwing hoeft te nemen sterk beperkt. Dit maakt het parseren simpel en snel en maakt de relatie tussen de observaties van de taal en de beslissingen die de parser moet nemen eenvoudig.

Het tweede deel van het proefschrift beschrijft de data die de parser bijhoudt en hoe deze worden gebruikt tijdens het parseren. Belangrijk is dat een nieuwe zin eerst wordt geparseerd en dat pas daarna de data van de zojuist geparseerde zin worden bijgehouden. De parser wordt op deze wijze geleidelijk verbeterd: nieuwe data worden toegevoegd aan de oude data en samen worden ze gebruikt om de volgende zin te parseren en meer data te verzamelen.

De data worden voor ieder woord apart bijgehouden. Voor elk woord bestaan de data uit *labels* die de frequentie tellen van woorden die naast het woord voorkomen en van de labels van die woorden. Op basis van deze labels worden simpele eigenschappen geïnduceerd die bepalen hoe woorden met elkaar kunnen worden verbonden tijdens het parseren. Als gevolg van de Zipfdistributie van woorden in natuurlijke talen hebben labels gebaseerd op frequente woorden de grootste invloed op de eigenschappen van alle woorden. Op deze manier vervangen de meest frequente labels de traditionele woordsoorten. Het induceren van de eigenschappen van een woord wordt uitgevoerd door het optellen van eigenschappen van andere woorden. Dat maakt het leerproces net als het parseren simpel en snel.

De parser is getest op drie corpora, in het Engels, Duits en Chinees. Bij ieder van deze drie talen weet de parser een deel van de syntactische structuur van de taal te ontdekken. De parser behaalt veel efficiënter dan eerder geconstrueerde unsupervised parsers ongeveer even goede resultaten.