Wide-Coverage CCG Parsing with Quantifier Scope

Dimitrios Kartsaklis

Master of Science

Artificial Intelligence

School of Informatics

University of Edinburgh

2010

Abstract

This dissertation presents the development of a wide-coverage semantic parser capable of handling quantifier scope ambiguities with a novel way. In contrast with traditional approaches that deliver an underspecified representation and focus on enumerating the possible readings "offline" after the end of the syntactic analysis, our parser handles the ambiguities during the derivation using a semantic device known as generalized skolem term. This approach combines most of the benefits of the existing methods and provides solutions to their deficiencies with a natural way. Furthermore, this takes place in the context of the grammar itself, without resorting to ad-hoc complex mechanisms.

As a grammar formalism for this work we use Combinatory Categorial Grammar (CCG), exploiting its lexicalized nature and the surface-compositional semantics that provides. The logical forms are represented in first-order logic, using λ-calculus as a "glue" language, in the tradition of Montague. We base our parser on the OpenCCG framework, and we augment it by applying a well-established supertagger and by developing a head-driven probabilistic model. Our model is trained on CCGbank, a CCG version of the Penn Treebank. For the semantic component we develop a Java library capable of representing and unifying λ-calculus expressions, including the necessary support for the new semantic elements of generalized skolem terms.

We evaluate the syntactic component of our parser on Section 23 of CCGbank, getting very high lexical accuracy and coverage but less than optimal PARSEVAL and head dependencies measures. For the semantic part, we create a test suite with sentences that exhibit a wide-range of quantifier scope ambiguities, and we test the performance of our parser on this. We also evaluate the semantic component on a small test set derived from the FraCas framework. For almost all cases we get results that conform to the predictions of the theory. Although time did not permit the testing of the semantic component in a wide-coverage setting, we consider these results as a strong indication that this is actually feasible. Finally, we close this dissertation with specific suggestions of what we consider important to be done as a follow-up work.

Acknowledgements

Despite its inevitable naïveness to the experienced eye, this piece of work is the hardest thing I have ever accomplished, and I feel obliged to many people for that. I would like to thank Mark Steedman, my supervisor, for his guidance and his support all those months, and for the fact that he introduced me to the exciting field of Natural Language Processing with the best possible way. Most of all, though, I thank him because in a time where a large part of AI is expended on counting things and dividing by total, he offered me the opportunity to work on something really "intelligent".

I owe a lot to Christos Christodoulopoulos, who was always there when I needed him, providing useful remarks and spending his time trying to understand the nature of my project. Furthermore, his well-written MSc thesis was one of the most helpful resources for me during this period, since the nature of our works were actually pretty similar. I would also like to thank Mark McConville for his useful suggestions and help on OpenCCG framework.

During this year I met many remarkable people and a few really exceptional: I am grateful to Sia Togia, for her valuable advice on semantics, her support, and the kind words with which she was always referred to my work. Most of all, I thank her for her willingness to help under any circumstances, regardless deadlines or how busy she actually was.

I very much thank Yang Gao, who was a true friend and faithful partner all those hard months. Her clear way of thinking helped me to put things in order every time I was starting to feel that I am loosing control.

I can't thank enough my good friend Andreas Chatzistergiou, for our endless discussions about almost everything, his patience when I was trying to explain things like why smoothing is not working (despite the fact, as he later admitted, that even the title of my thesis was giving him a headache), and in general for sharing with me this one in a lifetime experience here in Edinburgh. People like him and Sia showed me that after all there might be some chance for this small piece of land at the southern end of Europe.

Finally, I would like to thank my family for their support, emotional and financial during this year. Without them this would be impossible.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Dimitrios Kartsaklis*)

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Although a very large part of the activity that takes place in the context of Natural Language Processing (NLP) is focused on analyzing text from a syntactical or a morphological perspective, it seems that for certain really "intelligent" applications this is not sufficient. Indeed, in tasks like machine translation, question-answering systems, or automatic essay scoring, to name just a few, there will always be a gap between raw linguistic information (such as part-of-speech labels, for example) and the knowledge of the real world that is needed for the completion of the task in a satisfactory way. Semantic analysis has exactly this role, aiming to close (or reduce as much as possible) this gap by linking the linguistic information with detailed semantic representations that embody this elusive real-world knowledge.

As we would expect, such task is not trivial, and semantic analysis has still to face a variety of problems coming from the ambiguity that is inherent to every natural language. This project addresses one the most known kinds of such problems, the so-called *quantifier scoping* that can be originated from alternations in the scope of the quantifiers in a sentence. Departing from traditional approaches, such underspecification, we will approach the problem with a novel way, by using non-quantificational devices known as generalized skolem terms. This idea has its origins in an earlier work of Mark Steedman (1999), and it takes its final form in his forthcoming book, "The Natural Semantics of Scope"[1]. The main argument is that the integration of such an account with the syntactic derivation mechanism of Combinatory Categorial Grammar (CCG; Steedman, 2000) brings important advantages over the existing approaches and can provide elegant and natural explanations to many linguistic phenomena that arise from quantifier ambiguities. At the same time, the concept of generalized skolem

[1] Currently in publication by MIT Press.

terms tends to fit better in a self-contained semantic theory, eliminating the need of resorting to ad-hoc complex algorithmic mechanisms.

The purpose of this project is the creation of a wide-coverage semantic parser that will use generalized skolem terms in order to handle quantifier scope ambiguities. By providing a realization of the theory, we aim to present a proof of concept for its applicability on the specific problem and the advantages that brings compared to current approaches. The parser uses CCG as the grammar formalism, and first-order logic combined with λ-calculus for the representation of the logical forms. The reasoning behind these decisions will become clear in the following chapters, but here we can briefly note that the main attraction of CCG for our purposes was its lexicalized nature and the transparent interface between syntax and semantics that provides, while the motivation for selecting first-order logic and λ-calculus was readability.

The creation of a wide-coverage parser is a multi-stage process, and we can roughly divide our work in three main tasks:

1. *Development of the syntactic parser*. Since the semantics we use is surface-compositional, that is, guided by the syntactic derivation, the syntactic part of the parser plays an important role in the success of the project. We base our implementation on the OpenCCG framework – additionally, in order to get an acceptable performance we apply a probabilistic model based on head-word dependencies.

2. *Realization of the semantic component*. This step consists of the creation of a Java library responsible for the representation and manipulation of first-order logic/λ-calculus formulas. Of course, our library has to support the new semantic elements of the theory.

3. *Integration of the two components*. In this step we add the machinery for integrating the two components in one coherent tool. Beyond the algorithmic aspect, an important part of this step is the creation of the semantic lexicon containing the logical forms for the lexical items.

We evaluate the syntactic part using the standard PARSEVAL measures and the more appropriate for CCG metric of labeled and unlabeled dependencies, and we find that, although not comparable with state-of-the-art CCG parsers, our parser can indeed parse Section 23 of CCGbank presenting acceptable performance. Due to time constraints, the semantic aspect of our tool is evaluated against a carefully selected set of sentences

that encompass a wide range of linguistic phenomena related to quantifier scoping. Since we wanted to also provide an evaluation that uses test cases from some other independent source, we also test the parser on a small set of sentences derived from the FraCas framework (Cooper et al., 1996), which has been created as a tool for testing textual entailment tasks. We indeed find that the results we get are in line with the predictions of the theory, and we consider this as a strong indication that further work is justified on this area, covering aspects that time did not permit us to test in the context of this project.

The rest of this dissertation describes in detail every stage of the development of the semantic parser, focusing mainly on the semantic component which is the central point of the present work. More specifically, the dissertation is structured with the following way:

Chapter 2 provides the necessary background for the problem we address, presents the related work, and discusses in detail the approach we are going to use. Chapter 3 serves as a short introduction to the work that has been done so far on CCG wide-coverage parsing. Chapter 4 describes in detail our methodology, using information presented in the previous chapters. This chapter consists of two large sections, one dedicated to the development of the probabilistic parser and one describing the creation of the semantic component. Chapter 5 presents the detailed results of our evaluation for both components. Finally, Chapter 6 summarizes this work, provides comments on its contributions, and discusses in detail future work that needs to be done in this area.

Chapter 2

Background

"There is in my opinion no important theoretical difference between natural languages and the artificial languages of logicians"

Richard Montague, *Universal Grammar*

2.1 Semantics and compositionality

Undoubtedly, one of the most influential works on the area of linguistic semantics is due to Richard Montague (1930-1971), who was the first who managed to present an algorithmic way of processing fragments of the English language in order to get semantic representations capturing their "meaning" (Montague, 1970a,b, 1973). By treating English as a formal language, Montague was able to use the notion of *compositionality* in order to combine simple logical forms of individual words into more complex semantic representations for larger language fragments such as phrases and sentences. It is this idea, that "the meaning of the whole is a function of the meanings of its parts", often called *Frege's principle*[1], that makes us possible to construct detailed representations aiming to capture the meaning of arbitrarily complex discourse elements.

To make such an account work, one would need two things: First, a resource which will provide the logical forms of each specific word (a lexicon); and second, a way to determine the correct order in which the discourse elements should be combined in order to end up with a valid semantic representation. A natural way to address the

[1] Jannsen (1997) convincingly argues that it is more accurate to speak of "Fregean principle" than of "Frege's principle" since, despite the fact Frege's latest writings were in the spirit of the concept, the principle itself was not stated in its known form in any of his published works.

second problem, and one traditionally used in computational linguistics, is to use the syntactic structure as a means of driving the semantic derivation (an approach called *syntax-driven semantic analysis*). In other words, we assume that there is an one-to-one mapping between syntactic and semantic types, and that the composition in the syntax level implies a similar composition in the semantic level. This is known as the *rule-to-rule hypothesis* (Bach, 1976) and, in turn, provides an insight for the exact form of the lexicon for addressing the first problem: Each entry in our lexicon should include three pieces of information: the surface form of the lexical item, its syntactic type, and its logical form. For the sentence "Every boy likes some girl", we can imagine a lexicon of the following form (here using simple context-free grammar categories):

(1) a. every $\vdash DT : \lambda p.\lambda q.\forall y[p(y) \rightarrow q(y)]$

b. boy $\vdash N : \lambda x.boy(x)$

c. likes $\vdash VB : \lambda x.\lambda y.likes(y,x)$

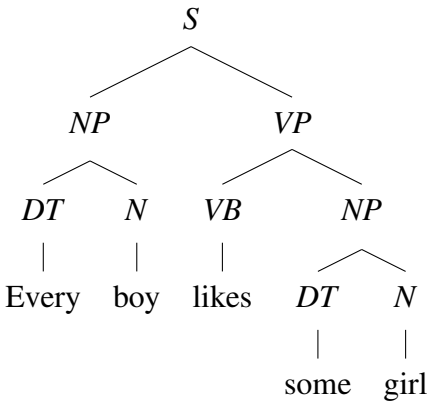
d. some $\vdash DT : \lambda p.\lambda q.\exists x[p(x) \wedge q(x)]$

e. girl $\vdash N : \lambda x.girl(x)$

In the above logical forms, λ-terms like λx or λq have the role of placeholders that remain to be filled. The predicate $likes(y,x)$ for example, defines a relation that holds between two entities: $likes(john, mary)$ simply means that John likes Mary. In the case of (1c), these entities are still unknown and they will be specified based on the syntactic combinatorics. Furthermore, the forms in (1a) and (1d) reflect the traditional way for representing universal and existential, respectively, quantifiers in natural language, where the still unknown part is actually the predicates acting over a range of entities.

Let's see how we can apply the principle of compositionality to get a logical form for the above sentence. The parse tree in (2) below provides us a syntactic analysis:

(2)



Our simple context-free grammar (CFG) consists of three rules, $NP \rightarrow DT\ N$, $VP \rightarrow VB\ NP$, and $S \rightarrow NP\ VP$, which essentially will drive the semantic derivation. Interpreted from a semantic perspective, the first rule states that the logical form of a noun phrase is the combination of the logical forms of its constituents (the determiner and the noun). So, in the logical form of (1a) λp will be substituted by the logical form for *boy* (1b), and the same will happen for *some* (1d) and *girl* (1e), yielding the following semantic representations for the two noun phrases:

(3) a. $\lambda q.\forall y[boy(y) \rightarrow q(y)]$

b. $\lambda q.\exists y[girl(x) \wedge q(x)]$

In order to get the logical form of the verb phrase, we need to unify the forms of the verb *likes* and the second noun phrase. This corresponds to substituting λq in (3b) with the logical form of *likes* (1c), which will result in the following semantic representation:

(4) $\lambda y.\exists x[girl(x) \wedge likes(y,x)]$

Finally, the third rule tells us that the logical form of the whole sentence is derived by the combination of the logical forms of the first noun phrase (3a) and the verb phrase (4):

(5) $\forall y[boy(y) \rightarrow \exists x[girl(x) \wedge likes(y,x)]]$

The above use of first-order logic in conjunction with λ-calculus was first introduced in Montague's seminal work (Montague, 1970b, 1973), and it can provide us a framework for getting logical forms for arbitrarily complex sentences. As we will see in Section 2.3, however, at the very heart of this compositional approach lies the problem of quantifier scoping that is the subject of our project. Before we move on to this though we have to introduce CCG, the grammar formalism we are going to use in this work.

2.2 Combinatory Categorial Grammar

2.2.1 Introduction

Combinatory Categorial Grammar (CCG; Steedman, 2000) is an extension of Categorial Grammar (Ajdukiewicz, 1935; Bar-Hillel, 1953), and has some important differences from typical context free grammars:

- It is a lexicalized grammar, which means that the grammar is entirely defined in the lexicon, with the form of syntactic categories assigned to individual words. In contrast with context-free grammars, such a grammar contains only a very small number of rules.

- Each category can be either a function that takes an argument (some other category) and returns a new syntactic type, or an argument that is used as input to a function category.

- It provides a completely transparent interface between syntax and semantics, meaning that each step of the syntactic derivation corresponds to a semantically interpretable structure.

A function CCG category can be of the form X/Y or $X\backslash Y$, where X is the result category, Y is the argument category, and the directionality of the slash defines the position in which the argument is expected: the forward slash ($/$) indicates that the position should be at the right of the functor, while the backslash ($\backslash$) indicates that the argument is expected at the left. X and Y can be arbitrarily complex categories, or atomic categories with no slashes that serve only as arguments (for example, *NP* or *PP*). Here are some examples, with the corresponding logical forms:

(6) a. Mary $\vdash NP : mary$

b. dances $\vdash S\backslash NP : \lambda x.dances(x)$

c. loves $\vdash (S\backslash NP)/NP : \lambda x.\lambda y.loves(y,x)$

The example in (6b) shows the category and the logical form for intransitive verbs: they need a noun phrase at their left (the subject – e.g. "Mary") to return a sentence ("Mary dances"). The case of a transitive verb is shown in (6c). It expects a noun phrase at its right (the object), to return something that can be combined with a noun phrase at its left (the subject), to make a sentence. This kind of category combination is known as functional application, and can be summarized as follows (again including semantics):

(7) *Functional Application*:

a. $X/Y : f \quad Y : a \quad \Rightarrow \quad X : f(a) \quad (>)$

b. $Y : a \quad X\backslash Y : f \quad \Rightarrow \quad X : f(a) \quad (<)$

Forward and backward functional application provide to a categorial grammar the expressive power of a context-free grammar, which is inadequate to capture complex syntactic phenomena like unbounded dependencies, coordination of non-standard constituents, and cross dependencies in Dutch and some other Germanic languages. In order to cover such cases, CCG has introduced some additional operations based on the work of Curry and Feys (1958):

(8) *Functional Composition*:

a. $X/Y : f \quad Y/Z : g \quad \Rightarrow_{\mathbf{B}} \quad X/Z : \lambda x.f(g(x)) \quad (>\mathbf{B})$

b. $X/Y : f \quad Y\backslash Z : g \quad \Rightarrow_{\mathbf{B}} \quad X\backslash Z : \lambda x.f(g(x)) \quad (>\mathbf{B}_{\times})$

c. $Y\backslash Z : g \quad X\backslash Y : f \quad \Rightarrow_{\mathbf{B}} \quad X\backslash Z : \lambda x.f(g(x)) \quad (<\mathbf{B})$

d. $Y/Z : g \quad X\backslash Y : f \quad \Rightarrow_{\mathbf{B}} \quad X/Z : \lambda x.f(g(x)) \quad (<\mathbf{B}_{\times})$

(9) *Type-raising*:

a. $X : a \quad \Rightarrow_{\mathbf{T}} \quad \mathrm{T}/(\mathrm{T}\backslash X) : \lambda f.f(a) \quad (>\mathbf{T})$

b. $X : a \quad \Rightarrow_{\mathbf{T}} \quad \mathrm{T}\backslash(\mathrm{T}/X) : \lambda f.f(a) \quad (<\mathbf{T})$

Composition (8) allows the combination of two functors into another functor, and the type-raising operators, shown in (9), can be used to convert an atomic category to a functor, if this is required by the derivation. These two operations (and, in a lesser degree for English, *substitution* which is not included in this short introduction), allows CCG to capture a wide-range of long-distance dependencies like in the case below.

(10)

the	song	that	Mary	likes
NP		$(NP\backslash NP)/(S/NP)$	NP	$(S\backslash NP)/NP$
			$\xrightarrow{>\mathbf{T}}$ $S/(S\backslash NP)$	
			$\xrightarrow{>\mathbf{B}}$ S/NP	
		$\xrightarrow{>}$ $NP\backslash NP$		
	$\xrightarrow{<}$ NP			

However, for certain sentences like the one in (11) (Steedman, 2000, p.42) a generalized form of composition is necessary, which can be applied over functors with different number of arguments but the same target (innermost) category:

(11)

I	offered,	and	may	give,	a flower	to a policeman
NP	$((S\backslash NP)/PP)/NP$	$CONJ$	$(S\backslash NP)/VP$	$(VP/PP)/NP$	NP	PP
			$\xrightarrow{>\mathbf{B}^2}$ $((S\backslash NP)/PP)/NP$			

The four rules of generalized composition are the following:

(12) *Generalized Functional Composition*:

a. $X/Y : f \;\; (Y/Z)/\$_1 : \ldots\lambda z.qz\ldots \Rightarrow_{\mathbf{B}^n} (X/Z)/\$_1 : \ldots\lambda z.f(g(z\ldots)) \;\; (>\mathbf{B}^n)$

b. $X/Y : f \;\; (Y\backslash Z)/\$_1 : \ldots\lambda z.qz\ldots \Rightarrow_{\mathbf{B}^n} (X\backslash Z)\backslash\$_1 : \ldots\lambda z.f(g(z\ldots)) \;\; (>\mathbf{B}^n_\times)$

c. $(Y\backslash Z)\backslash\$_1 : \ldots\lambda z.qz\ldots \;\; X\backslash Y : f \Rightarrow_{\mathbf{B}^n} (X\backslash Z)\backslash\$_1 : \ldots\lambda z.f(g(z\ldots)) \;\; (<\mathbf{B}^n)$

d. $(Y\backslash Z)/\$_1 : \ldots\lambda z.qz\ldots \;\; X\backslash Y : f \Rightarrow_{\mathbf{B}^n} (X/Z)/\$_1 : \ldots\lambda z.f(g(z\ldots)) \;\; (<\mathbf{B}^n_\times)$

The generalized form of composition uses the *$ convention* introduced in Ades and Steedman (1982), where the symbol $ schematizes over category clusters that have the same target category. In other words, the notation $S/\$$ could be used to denote the category set $\{S,\ S/NP,\ (S/NP)/NP,\ ((S/NP)/NP)/NP,\ \ldots\}$, where the innermost category (the target) in all cases is S.

2.2.2 Computational complexity of CCG

CCG has been proved by Vijay-Shanker and Weir (1994) to be weakly equivalent to Linear Indexed Grammar (LIG) and Tree-Adjoining Grammar (TAG; Joshi et al., 1975), belonging to a group of languages that occupies the low end of "mildly context-sensitive" family in the extented Chomsky hierarchy (Table 2.1) – a containment hierarchy of classes of formal grammars based on their expressive power. The theoretical power of CCG, LIG, and TAG is just above that of context-free grammars, so these languages can be also characterized as "nearly context-free". Even more interestingly, despite their low computational complexity it has been suggested that they provide sufficient level of expressiveness for capturing all syntactic phenomena of natural languages.

Grammar	Language	Automaton	Production rules
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$
Type-1	Context-sensitive	Linear-bound automaton	$\alpha A \beta \rightarrow \alpha\gamma\beta$
	Mildly context-sensitive	i-th order NPDA	$A \rightarrow f(\beta)$
	Nearly context-free	Nested push-down autom.	$A_{[(i),\ldots]} \rightarrow \alpha B_{[(i),\ldots]}\beta$
Type-2	Context-free	Push-down automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite-state automaton	$A \rightarrow a, A \rightarrow aB$

Table 2.1: The extended Chomsky Hierarchy. Lowercase English letters represent terminal symbols (words), uppercase letters are non-terminals (e.g. NP, VP), and lowercase Greek letters (α, β, γ) represent any combination of terminals and non-terminals.

2.3 Quantification problems

After this short introduction to our grammar formalism, we will now turn our attention to the main subject of this work, the quantifier scoping problem. We return to our example sentence, introduced in Section 2.1, and its logical form we produced by using λ-calculus (repeated here for convenience):

(13) Everyboy boy likes some girl:
$\forall y[boy(y) \rightarrow \exists x[girl(x) \wedge likes(y,x)]]$

How this expression can be interpreted? In plain English, the logical form implies that every boy y likes a (possibly) different girl x, which seems that indeed captures the intended meaning of the sentence. However, the problem here is that there is also a second reading which is not supported by the syntactic combinatorics:

(14) $\exists x[girl(x) \wedge \forall y[boy(y) \rightarrow likes(y,x)]]$

that is, there is a specific (very popular) girl x who is liked by every boy. This interpretation is equally valid with the one in (13), but since our semantics is driven by the surface form, and we have only one such form, it was left unaccounted. The two interpretations differ on which quantifier has the outer scope in the final expression: In the first case, the universal has scope over the existential, and the reverse is true for the second reading. The important point here is that a proper semantic account should be able to provide all the attested readings, despite the restrictions imposed by the syntactic form.

Interestingly, the expressiveness of CCG seems to provide a natural solution to the problem. While the conventional CCG derivation in (15a) will result in the narrow-scope reading, where the universal quantifier outscopes the existential, the use of composition and type-raising provides an alternative derivation, shown in (15b), covering the other case.

(15) a.

Every boy	likes	some girl
NP : $\lambda q.\forall y[boy(y) \rightarrow q(y)]$	$(S\backslash NP)/NP$: $\lambda x.\lambda y.likes(y,x)$	NP : $\lambda q.\exists x[girl(x) \wedge q(x)]$
	$S\backslash NP : \lambda y.\exists x[girl(x) \wedge likes(y,x)]$ (>)	
$S : \forall y[boy(y) \rightarrow \exists x[girl(x) \wedge likes(y,x)]]$ (<)		

b.

Every boy	likes	some girl
NP : $\lambda q.\forall y[boy(y) \rightarrow q(y)]$	$(S\backslash NP)/NP$: $\lambda x.\lambda y.likes(y,x)$	NP : $\lambda q.\exists x[girl(x) \wedge q(x)]$
$S/(S\backslash NP)$: $\lambda q.\forall y[boy(y) \rightarrow q(y)]$ (>T)		
S/NP : $\lambda x.\forall y[boy(y) \rightarrow likes(y,x)]$ (>B)		
S : $\exists x[girl(x) \wedge \forall y[boy(y) \rightarrow likes(y,x)]]$ (>)		

Despite the elegance of such a solution, however, this direct linking between scope and syntax implies that scope ambiguities can emerge only in cases where there are also ambiguous syntax derivations. Unfortunately, this is not always the case. If we had formulated our sentence as "Some girl, every boy likes", for example, we would get only the wide-scope derivation, where the $\exists_{girl}$ quantifier has the outer scope. However, the narrow-scope reading is still there, but this time it remains unaccounted by the syntax. Furthermore, although the description we provided above presents the core of the scoping problem, there are many more further complications that quantifiers can cause in a semantic analysis.

For example, one very common problem is that of spurious readings, where a sentence can be assigned more than one logical forms that bring the same meaning – that, is, they are equivalent. An example of this phenomenon is the sentence "Some representative showed some company some sample", which has six equivalent meanings that correspond to permutations of the three existential quantifiers:

(16) a. $\exists x[repr(x) \wedge \exists y[company(y) \wedge \exists z[sample(z) \wedge showed(x,y,z)]]]$

b. $\exists x[repr(x) \wedge \exists z[sample(z) \wedge \exists y[company(y) \wedge showed(x,y,z)]]]$

c. $\exists x[company(y) \wedge \exists x[repr(x) \wedge \exists z[sample(z) \wedge showed(x,y,z)]]]$

d. ...

In a sentence with n quantifiers, the number of alternative readings is $n!$. Such a proliferation of spurious interpretations could cause a significant overhead to any system. Koller and Thater (2006), for example, point out that the sentence "For travelers going to Finnmark there is a bus service from Oslo to Alta through Sweden" has 3960 distinct readings, all equivalent to each other.

Another possible problem is the case of scope asymmetries, as exemplified by the sentence "Every boy likes, and every girl detests, some saxophonist" (Geach, 1972). In such a case, a naïve account of quantification would result in a large number of possible readings, many of which are actually invalid. For example, the mixed reading:

(17) $\forall x[boy(x) \rightarrow \exists y[sax(y) \wedge likes(x,y)]] \wedge \exists v[sax(v) \wedge \forall z[girl(z) \rightarrow detests(z,v)]]$

presents a scope asymmetry, since the saxophonist is a different person for every boy, but the same person for all girls. In such a case, "some saxophonist" should in fact have retained the same scope for both the coordinated constituents.

Finally, in cases where a sentence contains more than two quantifiers, there is a possibility of "intermediate" scope readings, as in the following case:

(18) Some teacher showed every pupil every movie:
$*\forall x[movie(x) \rightarrow \exists y[teacher(y) \wedge \forall z[pupil(x) \rightarrow showed(x,y,z)]]]$

In the logical form of (18), the teacher depends only on the movies – in other words, this reading implies that there is a different teacher for every movie, which is not the intended meaning.

It seems then that the problem of quantifier scoping requires an even more flexible approach, one that would be able to properly handle every kind of complex dependency that can arise between multiple quantifiers. From the next section we will begin to work towards this direction, first presenting a comprehensive introduction to the most important approaches that have been applied so far in this regard.

2.4 Related work

A traditional approach to handle quantifier scope ambiguities is the creation of an *underspecified* representation that embodies all the possible readings without including specific information about the exact position of each quantifier. This idea was initially introduced by Kempson and Cormak (1981), and from then on has experienced a large number of implementations and variations. In this section we are going to discuss the most important approaches.

2.4.1 Storage-based approaches

Usually, underspecification is achieved through the notion of a *storage*, a special store that holds the core meaning of a node together with any quantified expressions that are gathered from the daughters of this node in the tree. For the case of the sentence "Every boy likes some girl", for example, the store would have the following form at the end of the derivation:

(19) $\langle loves(x_1,x_2),$
$(\lambda q.\forall x[boy(x) \rightarrow q(x)],1),$

$(\lambda q.\exists y[girl(y) \wedge q(y)], 2)\rangle$

Here, indices 1 and 2 indicate the position in the predicate from which each quantified expression is originated. Note that although these three expressions contain all the information we need, they remain neutral as to the order of the two quantifiers. After the end of the derivation, this underspecified representation is passed to a different component, which retrieves each quantified expression and merges it with the core meaning. The two alternative readings can be produced by retrieving the quantified expressions in different order. If we retrieve first the $\forall_{boy}$ expression, the narrow scope reading is created, where every boy likes a possibly different girl; by retrieving the $\exists_{girl}$ reading first, we produce the wide scope reading.

The approach described above was initially introduced by Cooper (1983), and in this form suffers an important deficiency: It cannot take in consideration the structure of a complex NP that contains other nested NPs, like in the case of "John likes every girl in the classroom"[2]. A solution to the problem was given by Bill Keller (1988), who essentially allowed every entry in the storage to be itself a storage. This *nested storage* concept was able to properly track abritrarily complex NPs by taking in account their hierarchical structure.

2.4.2 Constraint-based approaches

An important deficiency of storage methods described above is that they are too permissive – that is, they focus on representing all the available readings with an abstract way, but they do not have a means to filter efficiently these interpretations. As we saw in Section 2.3, this can lead to serious problems such as large numbers of spurious readings. Another problem of storage methods is that they can handle only a simple form of quantified noun phrases, similar to those we used in our previous examples. However, they are inadequate to face ambiguities that result in from more complex syntactic constructions, such as negation. For example, neither Cooper storage nor nested Cooper storage can properly address the sentence "Every boy does not like a girl".

Constraint-based approaches follow a different perspective: Each underspecified represenation is a set of constraints that govern the exact relations between the components of the sentence. Any first-order logic formula that satisfies these constraints

[2]We will not go into details about the exact nature of the problem here. The interested reader can find a detailed description in Blackburn and Bos (2005) pp. 122-125.

constitutes a valid logical form for the sentence. A representative of this area is the *hole semantics* approach (Bos, 1995), in which every λ-variable of a first-order formula is replaced by a *hole* that needs to be filled. Each sub-expression gets a distinct label, and which kinds of labels can fill which hole is governed by a set of what is called *dominance constraints*. With this approach, our example sentence gets the following underspecified representation:

(20) $l_1 : \forall x.[boy(y) \rightarrow h_1]$
$l_2 : \exists y.[girl(y) \wedge h_2]$
$l_3 : likes(y,x)$
$l_1 \leq h_0, l_2 \leq h_0, l_3 \leq h_1, l_3 \leq h_2$

The last line of the above representation provides the constraints. In general, an expression $l \leq h$ means that the expression with hole h dominates the expression with label l – that is, the expression with h must contain l as subexpression and not the other way around. In our example, h_0 is a hole that represents the whole expression, so it dominates every other expression. Holes h_1 and h_2 represent the two possible interpretations, so they outscope l_3. Furthermore, no constraint relates to l_1 and l_2 since we don't want to impose a specific ordering in the two quantifiers. Given the above underspecified form, the process of retrieving a fully specified formula corresponds to filling the holes with a way that satisfies every constraint. This is known as *plugging*. Finally, an important aspect of this method is that everything should be represented in a meta-language, which is responsible for associating labels and holes with the logical forms and imposing constrains like those in (20).

The notion of dominance constraints is also used by Koller and Thater (2006), this time in the form of *dominance graphs*. This work concerns the design of an algorithm that takes as input an underspecified represenation and reduces the available readings to distinct (non-equivalent) ones. The basic concept of their algorithm is that of permutable splits – that is, parts of the graph that are mutually exchangeable and thus equivalent. The algorithm proceeds by checking each subgraph for permutable splits; when it finds one, removes all the equivalent splits except the current.

Finally, we will close this section by mentioning the work of Koller et al. (1998) and Willis and Manandhar (1999), all of which used some aspect of dominance constraints for filtering the readings of underspecified represenations.

2.4.3 Other important contributions

There is a vast amount of work dedicated to underspecification and quantifier scope ambiguities, and it would be impossible to present everything in this short introduction. In this section we will provide some pointers to important contributions that can be used for further research. In particular, we should mention the work of Hobbs and Shieber (1987), Pereira and Shieber (1987), Copestake and Flickinger (2000), and Farkas (2001) all of which approached the quantification problems by applying some form of underspecification.

2.5 Skolemization

All the clever approaches described above can indeed provide a solution to the problem of quantifier scoping (up to a different degree each one), but all of them have a serious flaw: by decoupling the semantic derivation from the syntactic combinatorics, they allow more degrees of freedom on the way that the available semantic readings can be derived. This could give rise to a wide range of possible undesirable phenomena concerning the quantifiers, a sample of which we saw in Section 2.3.

Since all these side-effects have their roots in the complex dependencies that can arise between universal and existential quantifiers, an approach aiming to simplify these dependencies would seem reasonable. The process of *skolemization* (named after Thoralf Albert Skolem; 1887-1963) follows along these lines, aiming to relax the dependencies between quantifiers in formal logic statements by removing all existential quantifiers from them. More specifically, every existential quantifier is replaced by a function of all variables bound by a universal quantifier in whose scope the existential falls. In the example below

(21) $\forall x \exists y \forall z . P(x,y,z) \Longleftrightarrow \forall x \forall z . P(x, sk(x), z)$

the existential $\exists y$ has been replaced by the function $sk(x)$, since the only preceding universal quantifier was $\forall x$. In the case where the existential has the outer scope in an expression, it is replaced by a function of no arguments, that is, a constant:

(22) $\exists y \forall x \forall z . P(x,y,z) \Longleftrightarrow \forall x \forall z . P(x, sk(), z)$

Let us proceed and see the effect of skolemization in a more concrete example. Using skolem functions, the two readings of the sentence “Everyboy likes some girl” in (13) and (14) above now take the following form:

(23) a. $\forall x[boy(x) \rightarrow (girl(sk_1(x)) \wedge likes(x, sk_1(x)))]$

b. $\forall x[boy(x) \rightarrow (girl(sk_1()) \wedge likes(x, sk_1()))]$

In (23a) the skolem term is a function of x, yielding the meaning that every different boy x has their own specific favourite girl, $sk_1(x)$, whereas in (23b) the skolem term is a constant that has wide scope (the same very popular girl is liked by every boy). We can immediately see the advantage of skolemization in the representation of the logical forms: both readings now share an identical structure[3], where the only details that may differ is the set of the parameters in each skolem function. In the next section we are going to pursue further this important observation in order to apply a novel treatment of the quantifier scoping problem.

2.6 Generalized Skolem Terms

In his forthcoming work, "The Natural Semantics of Scope", Mark Steedman proposes a new semantic theory which replaces existential quantifiers in natural language with a generalized form of the skolem terms we introduced in the previous section. The main argument of Steedman's earlier and recent work on generalized skolem terms is that if this special form of skolemization is strictly guided by the combinatory rules of CCG, it can handle a large number of quantifier scope ambiguities like those described in Section 2.3 with a natural and elegant way. The key points are the following:

- The only determiners in English that should be represented by traditional generalized quantifiers are the universals *every*, *each*, and their relatives.
- Every other non-universal determiner should be associated with a generalized form of a skolem term, which can be freely interpreted at any step of the derivation process according to its environment – that is, the set of universal quantifiers in whose scope the generalized skolem term falls.

A *generalized skolem term* has the form $sk^{\mathcal{E}}_{n:p;c}$ where $\mathcal{E}$ is the environment, n the number of the originating noun phrase, p a nominal property, and c a cardinality condition. The index n is used only for disambiguation purposes, when the nominal property happens to be the same in two or more noun phrases – in all other cases is usually omitted. The nominal property p is a unary λ-abstraction that captures the semantics of the

[3] Technically, the process of skolemization produces an expression in *prenex normal form* (a sequence of quantifiers followed by a quantifier-free part) with only universal quantifiers.

specific NP, e.g. $\lambda x.man(x)$ for "a man", and can be arbitrarily complex. Finally, the cardinality condition c aims to capture quantifiers that denote subsets of a totality[4] like "most" or "at least 3", and can also be vacuous – case in which it will be also omitted.

When a skolem term is applied to a nominal property, say *girl*, it is still unspecified and has the form $skolem(girl)$. It will take its final generalized form by a process known as *skolem specification* that can freely occur at any point in the grammatical derivation based on the combinatory rules of the grammar. If at the time of specification $\mathcal{E}$ is empty, then the skolem term becomes a constant (a function of no arguments); if $\mathcal{E}$ is not empty, then the term becomes a function of the bound variables that the environment contains. In the light of the above description, the two formulas for our example sentence now become:

(24) a. $\forall x[boy(x) \rightarrow likes(x, sk_{girl}^{\{x\}})]$
b. $\forall x[boy(x) \rightarrow likes(x, sk_{girl}^{\{\}})]$

Although these formulas seem very similar to those in (23), there is an important difference: In this new form, we do not need to include a separate predicate *girl* over our skolem terms. This is because a generalized skolem term is a semantic element of its own right, where nominal properties like *girl* or *boy* are directly associated from the start. After the end of the derivation, these representations should be unpacked so the nominal properties would be able to be reinstated as predications over traditional skolem terms, producing the available alternative readings.

In order to apply the theory, we have to assign the following logical forms to universal and non-universal determiners:

(25) a. every, each, all, ...: $\lambda p.\lambda q.\forall x[p(x) \rightarrow q(x)]$
b. a, an, some, ...: $\lambda p.\lambda q.q(skolem(p))$

The form in (25a) is the traditional first-order logic formula for universals that we have already introduced in previous examples. The application of (25b) on a predicate of the form $\lambda x.girl(x)$ would consume the term λp yielding the expression $\lambda q.q(skolem(\lambda x.girl(x))$, in which the skolem term is still unspecified and can be specified at any step of the derivation. In this "self-application" form, the skolem term waits

[4] Actually, every quantifier in natural language denotes that a subset of some totality has a specific property. An expression of the form $\forall x\phi(x)$ denotes a set that contains every individual with property ϕ, while $\exists x\phi(x)$ asserts that the set of individuals with property ϕ has at least one member. The two traditional quantifiers of formal logic are nothing else but special cases of what is now known as *generalized quantifiers* (Mostowski, 1957).

for a predicate q that will be applied to itself, in order to bring itself to the scope of any quantifiers that could determine its specification.

We will now proceed to see how the use of generalized skolem terms together with the "anytime" process of specification can yield the two available readings showed in (24). In the analysis below the generalized skolem term for *some girl* is left unspecified until the last step of the derivation. Since during the specification is within the scope of *every boy*, it becomes a function of this quantified variable.

(26)

Every	boy	likes	some	girl
NP/N	N	$(S\backslash NP)/NP$	NP/N	N
$: \lambda p.\lambda q.\forall y[p(y) \rightarrow q(y)]$	$: \lambda y.boy(y)$	$: \lambda x.\lambda y.likes(y,x)$	$: \lambda p.\lambda q.q(skolem(p))$	$: \lambda x.girl(x)$
	$NP : \lambda q.\forall y[boy(y) \rightarrow q(y)]$ (>)		$NP : \lambda q.q(skolem(\lambda x.girl(x)))$ (>)	
		$S\backslash NP : \lambda y.likes(y, skolem(\lambda x.girl(x)))$ (>)		
	$S : \forall y[boy(y) \rightarrow likes(y, skolem(\lambda x.girl(x)))]$ (<)			
	$S : \forall y[boy(y) \rightarrow likes(y, sk^{\{x\}}_{girl})]$			

However, in (27) the specification takes places very early, where the skolem term is still unbounded. This results in a constant which is propagated to the final expression through the derivation.

(27)

Every	boy	likes	some	girl
NP/N	N	$(S\backslash NP)/NP$	NP/N	N
$: \lambda p.\lambda q.\forall y[p(y) \rightarrow q(y)]$	$: \lambda y.boy(y)$	$: \lambda x.\lambda y.likes(y,x)$	$: \lambda p.\lambda q.q(skolem(p))$	$: \lambda x.girl(x)$
	$NP : \lambda q.\forall y[boy(y) \rightarrow q(y)]$ (>)		$NP : \lambda q.q(skolem(\lambda x.girl(x)))$ (>)	
			$NP : \lambda q.q(sk^{\{\}}_{girl})$	
		$S\backslash NP : \lambda y.likes(y, sk^{\{\}}_{girl})$ (>)		
	$S : \forall y[boy(y) \rightarrow likes(y, sk^{\{\}}_{girl})]$ (<)			

Since the specification of a skolem term can happen at any step of the derivation, by the end of the parsing we will have both the required readings for the sentence. A subtle point is that the specification of a skolem term makes sense only in cases where the environment of this skolem term has been changed from the previous step – otherwise, the specification will yield an expression which is an exact duplicate of an already existing one. A new specification of the skolem term for *some girl*, for example, just after the combination of *likes* with *some girl*, will produce one more constant of the form $sk^{\{\}}_{girl}$, since the skolem term is still not bounded by any universals.

So now we are in position to propose an abstract algorithm that reflects the approach we are going to use:

(28) 1. Assign the appropriate categories to every determiner in the sentence, according to (25)
2. For every CCG rule that can be applied to a pair of categories $\langle B,C\rangle$ and produce a result A, combine the logical forms Λ_B and Λ_C to derive a new logical form Λ_A
3. For every skolem term sk in Λ_A, if the environment of sk has been changed from the previous step, specify a new generalized skolem term

The above steps are presented in pseudo-code form in Algorithm 2.1, and in Chapter 4 we will see how this can be placed in the context of the generic parsing process.

for all $\{DT_{\exists}|DT_{\exists} \in \{a, an, some, \ldots\}\}$ **do**
  $\Lambda_{DT_{\exists}} \leftarrow \lambda p.\lambda q.q(skolem(p))$
end for
for all $\{DT_{\forall}|DT_{\forall} \in \{every, each, all, \ldots\}\}$ **do**
  $\Lambda_{DT_{\forall}} \leftarrow \lambda p.\lambda q.\forall x[p(x) \rightarrow q(x)]$
end for
for all $\{A|A \rightarrow BC \in grammar\}$ **do**
  $\Lambda_A = app(\Lambda_B, \Lambda_C)$
  for all $\{sk^{\mathcal{E}_{OLD}}|sk^{\mathcal{E}_{OLD}} \in \Lambda_A\}$ **do**
    if $\mathcal{E}_{NEW} \neq \mathcal{E}_{OLD}$ **then**
      $specify(sk^{\mathcal{E}_{NEW}})$
    end if
  end for
end for

Algorithm 2.1: Using generalized skolem terms to handle quantifier scope ambiguities

This approach is in position to address quantifier scope ambiguities much more effectively than existing quantification methods such as storages, while at the same time delivers in the context of the grammar itself functionality that now is available through complex algorithmic solutions such as dominance graphs. For example, while a naïve quantification approach could return $3! = 6$ readings for a sentence with 3 quantifiers, like "some representative of some company showed a sample", it is clear that our method will return just one:

(29) $show(sk^{\{\}}_{\lambda x.repr(x)}, sk^{\{\}}_{company}, sk^{\{\}}_{sample})$

since there is no universal within whose scope any of the existentials can fall – and this will be solely based on the combinatory rules of the grammar, without the need of

using some kind of meta-language like in hole semantics or resorting to a dominance graph reduction method.

The Geach sentence with the scope asymmetries presents a much more challenging case for all quantification approaches. The diagrams below show how a CCG parser equipped with generalized skolem terms can deliver only the non-mixed readings (where the saxophonist retains the same scope for both boys and girls).

(30) a. Every boy admires and every girl detests — some saxophonist

$$\frac{\dfrac{\text{Every boy admires and every girl detests}}{S\backslash NP : \lambda x.\forall y[boy(y) \rightarrow admires(y,x)] \wedge \forall z[girl(z) \rightarrow detests(z,x)]} \qquad \dfrac{\dfrac{\text{some saxophonist}}{NP}}{S\backslash(S/NP) : \lambda q.q(skolem(sax))}{<\mathbf{T}}}{S : \forall y[boy(y) \rightarrow admires(y, skolem(sax))] \wedge \forall z[girl(z) \rightarrow detests(z, skolem(sax))]}<$$

$$\cdots\cdots\cdots\cdots\cdots\cdots$$

$$S : \forall y[boy(y) \rightarrow admires(y, sk_{sax}^{\{y\}})] \wedge \forall z[girl(z) \rightarrow detests(z, sk_{sax}^{\{z\}})]$$

b. Every boy admires and every girl detests — some saxophonist

$$\frac{\dfrac{\text{Every boy admires and every girl detests}}{S\backslash NP : \lambda x.\forall y[boy(y) \rightarrow admires(y,x)] \wedge \forall z[girl(z) \rightarrow detests(z,x)]} \qquad \dfrac{\dfrac{\text{some saxophonist}}{NP}}{S\backslash(S/NP) : \lambda q.q(skolem(sax)) \;\cdots\; : \lambda q.q(sk_{sax}^{\{\}})}{<\mathbf{T}}}{S : \forall y[boy(y) \rightarrow admires(y, sk_{sax}^{\{\}})] \wedge \forall z[girl(z) \rightarrow detests(z, sk_{sax}^{\{\}})]}<$$

In (30a), the specification of the skolem term takes places at the end of the derivation. However, since the skolem term is the *same* for both coordinated constituents, the specification takes places in parallel, yielding just one reading, that of the narrow-scope saxophonist. In (30b) the skolem term is specified early, and then again is the same entity that substitutes λx in the logical form of the verb phrase for both boys and girls. At the end of the derivation we have a single reading, this time the one in which the saxophonist is the same person (say, John Coltrane) for everybody.

As we will see in Chapter 5, this approach can indeed address a wide range of other quantifier scoping phenomena with a natural way. Furthermore, given the transparent interface between syntax and semantics that a lexicalized grammar as CCG provides, it can be easily incorporated into the syntactic combinatorics without requiring extensive modification of the underlying mechanism or raising the theoretical power of a CCG system. In fact, it can be shown that the power of a *nested push-down automaton* (NPDA) needed for the syntactic combinatorics of CCG is also sufficient for accommodating the semantic manipulation. Finally, the most important point is that

the whole process of semantic derivation will be performed in parallel with the syntactic analysis, solely based on the combinatory rules of CCG – this close integration between syntax and semantics provides a limit to the degree of freedom in which the available readings are derived, and ensures that all non-attested readings have been excluded from the results.

Chapter 3

Wide-coverage Parsing

The term wide-coverage parser in the NLP literature is used to describe a tool that can be syntactically or semantically analyze unrestricted text of the kind we meet in newspapers and magazines. The biggest problem with such an endeavor is that enumerating all possible parses and choosing the "best" possible parse according to some criterion is usually prohibitively expensive from a computational perspective. So, in order for a parser to be able to complete its task in a reasonable amount of time, an efficient way is needed for restricting the search space and focusing on the most promising parses. This is the role of the *oracle*, one of the three parts of a parser (with the other two being a *grammar* and an *algorithm*), and in the following sections we will see how statistical models can be used for CCG towards this goal. Ironically, the expressiveness of CCG adds to this problem, since even a simple sentence can have many different (but semantically equivalent) analyses, a phenomenon that sometimes called *spurious ambiguity* (Wittenburg, 1986). An example is shown in (1) below[1].

(1) a.

$$\dfrac{\dfrac{\text{John}}{NP : john} \quad \dfrac{\dfrac{\text{likes}}{(S\backslash NP)/NP : \lambda x.\lambda y.likes(y,x)} \quad \dfrac{\text{peanuts}}{NP : peanuts}}{S\backslash NP : \lambda y.likes(y,peanuts)}\,>}{S : likes(john,peanuts)}\,<$$

b.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{John}}{NP : john}}{S/(S\backslash NP) : \lambda q.q(john)}\,>\mathbf{T} \quad \dfrac{\text{likes}}{(S\backslash NP)/NP : \lambda x.\lambda y.likes(y,x)}}{S/NP : \lambda x.likes(john,x)}\,>\mathbf{B} \quad \dfrac{\text{peanuts}}{NP : peanuts}}{S : likes(john,peanuts)}\,>$$

[1]Note that this is different from the case presented by diagrams 15a and 15b in Chapter 2. In that case, the alternative derivations produced *different* logical forms, so they cannot be considered spurious.

Since these equivalent parses can be up to a Catalan number, that is $\frac{(2n)!}{(n+1)!n!}$ for a sentence with n words, this problem threatens to undermine the nearly context-free complexity of CCG. One solution has been proposed by Eisner (1996), involving the restriction of type-raising and composition only in cases in which these operations are necessary. More specifically, Eisner proposed that any constituent which is the result of a forward composition cannot serve as a functor in another forward composition or application. Similarly, any constituent produced by a backward composition cannot serve as a functor in another backward composition or application. This technique creates *normal form* derivations, and can drastically reduce the CCG parsing space.

3.1 Probabilistic models

In modern parsers the role of the oracle is fulfilled by statistical models, either generative or discriminative. A parser that uses a *generative model* makes every decision by taking in account probabilities calculated by the training data; the overall probability of an analysis (parse tree) is the joint probability of all these individual decisions that have been taken along the way. On the other hand, *discriminative models* attempt to calculate the probability of a parse tree given a sentence directly, by combining linearly a set of features associated with specific weights; in this case, the goal of learning is to set the feature weights in a way that favours the most probable analysis for the sentence. The next sections describe in some detail how both of these approaches have been applied to CCG parsing with very satisfying results.

3.1.1 Generative models

The first important work on generative models for CCG parsing (and one very revelant to our case, since in Chapter 4 we are going to reproduce a part of it) was conducted by Hockenmaier (2003). Using as her primary resource CCGbank (a CCG version of Penn Treebank – for more details, see Hockenmaier and Steedman, 2002), the author describes a number of models based on a simple probabilistic model (Hockenmaier, 2001) which served as a baseline for the evaluation of the performance in all other experiments.

In Hockenmaier's models a CCG derivation is handled in a top-down manner, starting from the root of a tree and proceeding to the leafs. The overall probability of a parse tree is calculated as the product of the probabilities of all local trees (trees with depth

1) that comprise the specific analysis. The probability of a local tree with root N then is defined in terms of a number of individual probabilities (described here for the baseline model): First, the expansion probability $P(expansion|N)$ is used to decide how the node N should be expanded. There are four different possibilities:

1. It is a leaf node, so it must be expanded to an appropriate word
2. It is a unary node, that is, a node that expands to just one daughter
3. It is a binary node (it has two daughters) with the head daughter being at the left
4. It is a binary node with the head daughter being at the right

If N is a leaf node, then the lexical probability $P(w|N, expansion = leaf)$ is used to estimate the most probable word w for the specific leaf category. If N is not a leaf (i.e. it has either one or two daughters) then the most probable category H of the head daughter (the daughter from which the parent inherits its lexical head) is calculated given the parent N and the way it has been expanded, using the head probability $P(H|N, expansion)$. In the case that the number of daughters is two, the non-head probability $P(S|N, expansion, H)$ estimates the most probable category S for the non-head daughter, given the category of the parent node, the way it has been expanded, and the category of the head daughter. Finally, the daughters are expanded using the same process, until every constituent has been expanded to a word.

Extending the baseline model

The most serious problem of a probabilistic model for parsing is the independence assumption that embodies: the category of each individual node is estimated independently of the position and the context of this node in the tree, possibly leading to wrong analyses (for example, the word "book" can be a noun or a verb, but if we notice that the previous word is a determiner then we can rule out the latter case). Collins (1999) showed that one of the most efficient ways to solve this problem is to use head dependencies, that is, to take in account not only the category of each individual constituent but also its head word (the most important word in the constituent from a grammatical perspective).

However, for a grammar as expressive as CCG the independence assumption seems a lesser problem, since every CCG category encodes a great amount of contextual information. Despite of that, Hockenmaier tried to further lessen possible problems by

applying a number of additional features to the baseline model, both non-lexical and lexical. The non-lexical features extensions were as follows:

- A boolean feature which was true for every constituent expanded to some kind of coordination and false for all other cases.
- The addition of information not only for the parent node of a local tree, but also for the grandparent node.
- The inclusion of distance measures, produced by calculating the distance of a head word from the left and the right frontier of its constituent.

Each extension modifies the probabilities of the baseline model with some way. In the case of the grandparent feature, for example, the expansion probability of a node N is not only based on the category of the node itself, but also on the category of the grandparent, GP: $P(expansion|N, GP)$.

More importantly, the author extended her models using lexical dependencies between the heads of the nodes. According to her top-down approach, the lexical head of a node (both the lexical category of the head word and the head word itself) is created probabilistically before of anything else, and then this information is used for expanding the node and generating the local tree. So, in this model, the daughters are conditioned both on the parent node and its lexical head. Furthermore, the author incorporated word-word dependencies, since the head of the one daughter is generated by taking in account the head of the other.

Table 3.1 provides a comparison of the results for the individual models, making clear that the word-word dependency model (HWDep) provides the best performance. The column *LexCat* shows the accuracy in the assignment of lexical categories, columns *LP*, *LR*, *BP*, *BR* refer to the standard PARSEVAL measures, and the two rightmost columns correspond to labeled and unlabeled dependencies, a measure that is more appropriate for CCG parsers since does not penalize the binary nature of the parses. For more details about the measures used for parsing evaluation, refer to Section 5.1. The HWDep model is described with details in Section 4.1.3.

3.1.2 Discriminative models

An important deficiency of generative probabilistic models is that they cannot easily incorporate arbitrary information – for example, there is no easy way to define specific

Model	**NoParse**	**LexCat**	**LP**	**LR**	**BP**	**BR**	$\langle \mathbf{P},\mathbf{H},\mathbf{S} \rangle$	$\langle\rangle$
Baseline	6	87.7	72.8	72.4	78.3	77.9	75.7	84.3
Conj	9	87.8	73.8	73.9	79.3	79.3	76.7	85.1
Grandparent	91	88.8	77.1	77.6	82.4	82.9	79.9	87.9
LexCat	9	88.5	75.8	76.0	81.3	81.5	78.6	86.8
LexCatDep	9	88.5	75.7	75.9	81.2	81.4	78.4	86.6
HeadWord	8	89.6	77.9	78.0	83.0	83.1	80.5	88.3
HWDep	8	92.0	81.6	81.9	85.5	85.9	84.0	90.1

Table 3.1: Performance of CCG generative models (Hockenmaier, 2003)

cases of long-range dependencies or distance measures. This greater degree of flexibility is given through the use of discriminative models, which is the approach adopted by Clark and Curran (2004b).

Taking ideas from the previous successful works in CCG parsing (Hockenmaier, 2003; Clark et al., 2002), the authors compare two different log-linear models: The first uses normal-form derivations, while the second exploits predicate-argument dependencies in which each derived structure is related to the complete set of possible derivations, including the non-standard ones. As the authors note, one of their goals was to test in what degree non-standard derivations can affect the analysis of a sentence. Since these are discriminative models, the probability of a parse ω given a sentence S is calculated by combining linearly a set of features associated with appropriate weights. More concisely:

$$P(\omega|S) = \frac{1}{Z_S} e^{\Sigma_i \lambda_i f_i(\omega)} \tag{3.1}$$

where λ_i is the weight of the ith feature, Z_S a normalizing constant ensuring that $P(\omega|S)$ is a valid probability, and $f_i(\omega)$ is a function that defines how many times the ith feature occurs in the context of ω. For the normal-form model, where ω is a single derivation, the above equation can directly provide a distribution for our probabilities. However, for the "all-derivations" model, a parse ω is actually a single derivation d (standard or non-standard) paired with the dependency structure π to which it leads. So, in this model, the probability of a dependency structure π given a sentence S is defined as the sum of the probabilities of all possible derivations d that lead to the specific dependency structure. That is,

$$P(\pi|S) = \sum_{d \in \Delta(\pi)} P(d, \pi|S) \tag{3.2}$$

where $\Delta(\pi)$ is the set of all derivations that lead to π.

A key component for the success of every discriminative model is the set of features that incorporates. For their dependency model, Clark and Curran chose to use features that were based on local or long-range predicate-argument dependencies. Each of these dependencies was represented by a tuple of the form $\langle h, c, s, h_a, l \rangle$, where h is the head word of the component, c is the lexical category, s is the specific slot for which the dependency holds, h_a is the head word of the argument, and l depends on the kind of the dependency: if the dependency is local, then l is null; if the dependency is long-range, then l is the lexical category of the item that mediates between the two tensed domains. For example, the tuple

(2) $\langle chased, (S\backslash NP_1)/NP_2, 2, cat, (NP\backslash NP)/(S/NP) \rangle$

represents the fact that the word "cat" can be the extracted object of "chased" (so in this case the number 2 indicates that we are referring to the second NP in the lexical category of "chased"), and can be connected with that verb using a relative pronoun with category $(NP\backslash NP)/(S/NP)$.

For the normal form model, the features were based on local rules instantiations, that is, local trees consisting of a parent and one or two daughters (pretty similar with the way that Hockenmaier created her lexicalized generative models). The authors enriched their feature sets by also including other information, for example distance measures and POS labels.

3.1.3 Comparing the two approaches

The experiments conducted by Clark and Curran showed that the normal-form model had almost the same performance with the dependency model, meaning that the impact of non-standard derivations in a dependency structure was very low. This was actually very good news, since the training of the dependency model was very computationally expensive, requiring about 17 hours in a 64-node cluster. On the other hand, the normal-form model was able to be trained in just 2 hours. Furthermore, as noted above, it also had significantly better parsing times than the dependency model. Compared with the best generative model of Hockenmaier, both of the discriminative

models had a slightly better performance. Table 3.2 presents the normal-form results in comparison with the two previous CCG parsing works.

	LP	LR	UP	UR	LexCat
Clark et al. (2002)	81.9	81.8	90.1	89.9	90.3
Hockenmaier (2003)	84.3	84.6	91.8	92.2	92.2
Clark and Curran (2004b)	86.6	86.3	92.5	92.1	93.6

Table 3.2: Comparison of the results for the CCG parsers (Clark and Curran, 2004b)

As in Hockenmaier's work, Clark and Curran found that word-word dependencies result in great improvements to their models. They also reported that the inclusion of distance features improved further the performance of the parser (this time in contrast to Hockenmaier), something that they attributed to the discriminative nature of their models.

3.2 Wide-coverage semantic parsing

The creation of robust and accurate parsers for CCG like the ones described in the previous sections made possible for researchers to pursue a more ambitious step, that of wide-coverage semantic parsing. Despite the large number of efficient wide-coverage parsers that exist, most of them provide syntactic phrase-structure trees that are not very helpful for reconstructing predicate-argument relationships or deriving logical forms. Exploiting the surface compositional semantics of CCG and its lexalized nature, Bos et al. (2004) used Clark and Curran's parser in order to create a system capable of providing semantic representations for newspaper text. The system operated on the output of the parser, which was a CCG derivation annotated with the exact rules applied in every step. Their overall approach consisted of the following steps:

1. Assigning semantic representation to the lexical items
2. Mapping each combinatory rule to the appropriate semantic transformation
3. Dealing with unaru rules, such as type-raising and type-changing
4. Applying β-conversion[2] to the derivation

[2]The basic process of a λ-abstraction, where λ-terms such λx are substituted by other expressions – a detailed description is provided in Section 4.2.2.

The authors annotated their lexicon with semantic representations with a semi-automatic way. For most open-class items they used the lemma to instantiate the semantics. As an example, the logical form of intransitive verbs was the following (in the Neo-Davidsonial analysis of events adopted by the authors):

(3) $\lambda q.\lambda u.q(\lambda x.\exists e(\texttt{lemma}(e) \wedge agent(e,x) \wedge u(e)))$

where `lemma` was replaced every time by the appropriate lemma, for example "walk" or "talk". For closed-class lexical items, the semantics were defined for each lemma individually. The mapping between syntactic rules and semantic reformulation is explicitely stated for each CCG rule in Section 2.2, and we will not go into further details here.

Since there is no some globally accepted evaluation metric for semantic analysis, the authors evaluated their work by testing the coverage and the well-formedness of the semantic representations, and they found that the system was able to cover 92.3% of the test corpus, providing logical forms all of which were well-formed. Below we can see an example of the system's output for a CCGbank sentence:

```
some A ((school-board[A] & hearing[A]) & some B (female[B] & some C
(dismiss[C] & (patient[C,B] & (at[A,C] & some D (crowd[D] &
(patient[D,A] & ((some E (student[E] & with[D,E]) & some F
(teacher[F] & with[D,F])) & event[D])))))))))
```

Figure 3.1: Output of Bos et al. (2004) system for the sentence "*The school-board hearing at which she was dismissed was crowded with students and teachers*"

Of course, as authors note, the well-formedness of an epxression does not necessarily guarantee its correctness – furthermore, the system in its original form did not attempt to handle complex semantic phenomena such as pronominal anaphora resolution or quantifier scope ambiguities. In any case, this work consisted a proof that wide-coverage semantic interpretation is indeed feasible, and it clearly demonstrated the attractions of CCG for such a purpose.

Chapter 4

Methodology

The purpose of this project is the creation of a wide-coverage parser able to use generalized skolem terms in order to handle quantifier scope ambiguities. This chapter describes with details all stages of the development and explains the courses of action that have been followed along the way. Before we step into implementation issues, however, we need to provide an answer to some questions regarding our fundamental decisions, the first of which was the selection of a grammar formalism and a logical form for the semantic representations.

Regarding the selected grammar, it should be clear by our previous discussion in Chapter 2 that CCG was a very appropriate choice for our purposes. The benefits of this formalism for a semantic parser can be summarized to the following points:

1. The transparent interface of CCG between syntax and semantics provides the surface compositionality that is necessary for such a goal

2. The lexicalized nature of the formalism makes the task of adding semantic representations to surface forms trivial, by simply augmenting the existing lexicon

3. The small number of combinatory rules means that little effort is required for the modification of each rule in a way that allows the correct semantic combinations to take place.

For our semantic representations, we decided to create our own Java library that allowed us to use first-order logic combined with λ-calculus as a "glue" language. This method was introduced by Montague (1970b) in its "Universal Grammar" paper, and from then on constitutes the way of preference among linguists for representing

semantics in natural languages. There is a reason for that: First-order logic is flexible enough to capture in a great extend the complexity of a natural language, while at the same time it can present the meaning of text with an elegant and intuitive way. It is true that there were many other alternative solutions, with the most appealing of them to be hybrid logic, since this was the semantic representation supported by the OpenCCG framework we used for the creation of the parser. The basic problem of such an approach was simply that hybrid logic is not as expressive as first-order logic – for example, it needs to be extended in certain ways in order to allow quantification (Blackburn and Marx, 2002). This limited expressiveness might be adequate for specialized tasks (Blackburn and Bos, 2005, p. 46), but eventually it will pose problems in the context of a wide-coverage parser. Even if we have decided to follow that way, the incorporation of generalized skolem terms into the existing logical framework would have required a great amount of ad-hoc manipulation with consequences in the clarity of the approach and possibly the robustness of the tool (for example, how can we embed the notion of the environment in a hybrid logic graph?). After all, the purpose of the project was to provide a proof of concept for the theory being tested, and not to be expended in obscure technical details about transformations between first-order and hybrid logic.

We need to address one more question: Why we chose to create our own probabilistic parser instead of using some of the existing state-of-the-art parsing tools that are now available for CCG? The basic motivation was that since none of those parsers carried some built-in functionality for semantic support, we would need in any case to closely integrate some semantic module with the existing parsing machinery. This would have required extensive modifications of the parser's existing code, a task that (as any software developer can confirm) can be proved very hard even under the most propitious circumstances (which basically are well-written code, consistent application of software engineering principles, and, even more importantly, very detailed documentation). Unfortunately, most of the code written in the context of some research project does not reach this level of readability, which means that by following such an approach (and given the limited time) we would put ourselves in the danger to spend a lot of time trying to control something that it was simply out of our control in the first place.

Of course, another possibility for us would be to apply an approach similar with Bos et al. (2004) (briefly discussed in Section 3.2), whose semantic component acts on the output of Clark and Curran's parser with very good results. The basic problem

with this method was that for our purposes we needed access to every possible full parse, not just to the single "best" one that is returned by the parser's statistical model. The necessity of examining all the alternative semantic readings corresponding to a full parse originates from two reasons: First, we needed this in order to thoroughly test the behaviour of semantic component; second, and even more importantly, this strategy actually seems to be integral to the method we apply, since there are cases where the full set of available readings can be derived only by scanning every alterative semantic derivation, not just the one that is linked to the best syntactic analysis (we will encounter such situations during the semantic evaluation stage, in Section 5.2). A modification of an existing parser to allow something like this could again lead to complications.

With all the above said, however, we have to admit that a great motivation for creating a parsing tool was the educational value of such an endeavor. The ultimate goal of an MSc thesis, after all, is to offer an opportunity for applying the taught material of the course in practical conditions. To this respect, the creation of a probabilistic parser was a necessary and invaluable addition, since it combined almost every aspect of a Natural Language Processing programme of studies.

4.1 Creating a wide-coverage parser

4.1.1 The OpenCCG framework

Since the creation of an efficient statistical parser from scratch is far beyond the scope of a project like this, our tool will be based on the *OpenCCG* framework. OpenCCG is an open-source Java framework for text parsing and realization purposes based on the CCG formalism. It has been developed by Jason Baldridge as a successor of the Grok system, while current development efforts, led by Michael White, are focused on practical applications of the realizer in dialogue systems. Despite the fact that the system has been used so far mostly for realization and natural language generation purposes, it provides a rich and well-tested API for parsing purposes which allows us to get a quick and robust result in the available time. Moreover, it fully supports almost every aspect of CCG.

OpenCCG parsing is based on the *Cocke-Kasami-Younger (CKY)* algorithm, a popular instance of *dynamic programming* for parsing purposes which uses a chart (a table) to store the intermediate results. The basic intuition of dynamic programming is that

a complex problem can be solved by providing solutions to smaller sub-problems. In the parsing case, by storing every intermediate step in our derivation (every partial derivation), we are able to re-use it for our larger calculations (derivations that capture larger spans), saving a lot of time. The algorithm scans the chart left-to-right and bottom-up, combining categories and storing new intermediate results, so that at the end of the derivation the top-right cell of the chart contains one or more full parses of the sentence. This process is summarized in Figure 4.1, where solid lines represent a conventional CCG derivation for a simple sentence, and dotted lines show a derivation using type-raising and composition. Algorithm 4.1 provides the pseudo-code.

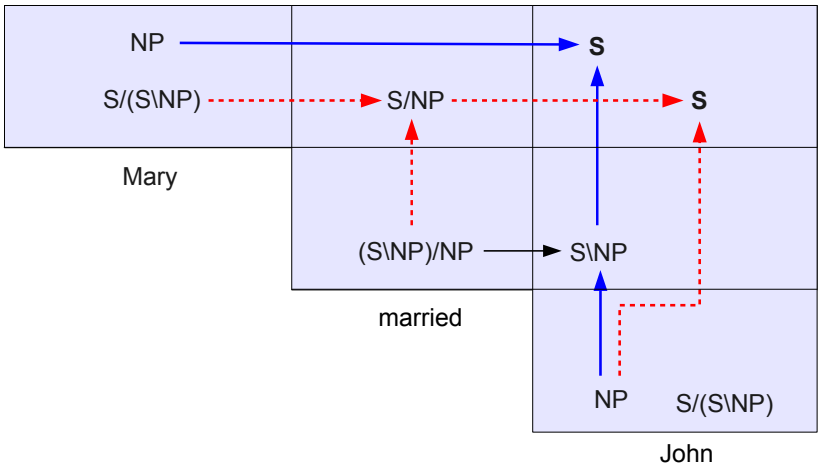


Figure 4.1: The chart-parsing process

function CKY-PARSE($words, grammar$) **returns** $table$
for $j \leftarrow 1$ **to** LENGTH($words$) **do**
  $table[j-1, j] \leftarrow \{A | A \rightarrow words[j] \in grammar\}$
  for $i \leftarrow j-2$ **downto** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      $table[i, j] \leftarrow table[i, j] \cup \{A | A \rightarrow BC \in grammar,$
        $B \in table[i,k],$
        $C \in table[k, j]\}$
    end for
  end for
end for

Algorithm 4.1: The Cocke-Kasami-Younger (CKY) algorithm (adapted by Jurafsky and Martin, 2000)

4.1.2 Supertagging

The first step of the parsing process, as demonstrated in line 3 of Algorithm 4.1, deals with the assignment of lexical categories to each word in the input sentence. The parser reads each word in the row, scans the lexicon (which has been extracted from some corpus – in our case CCGbank), and adds every category from the word's entry into the corresponding chart cell. The problem here is that in lexicalized grammars like CCG a word can correspond to a large number of categories, depending on the different contexts in which the word has been seen in the corpus used for the creation of the lexicon. For example, the number of categories for the verb *is* in CCGbank is 45. This fact, combined with the spurious ambiguity phenomenon of CCG, could cause a high computational cost to any parsing system.

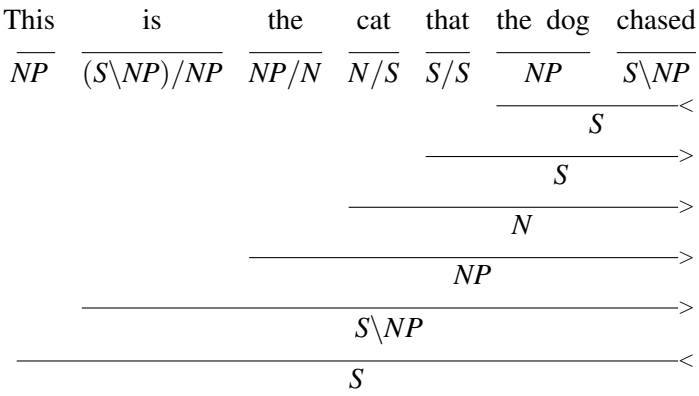
A more sensible approach therefore would be, instead of blindly assigning every category to the word, to assign only the lexical categories that are more likely given the context of the word, by using a probabilistic model. This technique, known as *supertagging* (Bangalore and Joshi, 1999) has been applied by Clark and Curran (2004b) on their parser with excellent results, and provably can drastically reduce the parsing speed without noticeable loss in the quality of the results (Table 4.1). The supertagger of Clark and Curran uses a log-linear model trained from CCGbank, with probabilities of the form:

$$p(c|h) = \frac{1}{Z(h)} e^{\sum_i \lambda_i f_i(c,h)} \tag{4.1}$$

Here, c is a lexical category, h is a context, f_i a feature, λ_i the corresponding weight, and $Z(h)$ a normalization constant. The context h is a 5-word window surrounding the target word, and the features are the words in the window as well as their POS tags.

To provide a simple example of how beneficial the use of a supertagger can be, we used Clark and Curran's supertagger in order to assign categories to the sentence in (1) below. The interesting point here is that, by taking into account the context of word *cat*, the supertagger has assigned to it the category N/S, instead of the presumed N. With this way, the supertagger allows *cat* to be nicely combined with the relative pronoun *that*, simplifying the rest of the derivation. It is exactly this quality of supertagging that made Bangalore and Joshi to call the technique *almost parsing* in their work on the LTAG formalism.

(1)



The supertagger of Clark and Curran works as follows: first, it uses the probabilistic model in order to find the most likely category for a specific word, given its context. Next, it assigns all the other categories from the word's entry in the lexicon whose probabilities fall within a factor β of the highest probability. If the word has been seen fewer than k times in the training data, the supertagger assigns categories based on the POS tag of the word, instead on the word itself. So the performance of the supertagger is governed by these two parameters, β and k, and Table 4.1 provides some results under different configurations. The last row ($\beta = 0$) represents the "no restrictions" case, where each word gets all the categories from its entry in the lexicon. Note that despite the drastic reduction in the number of categories assigned to each word in average, the accuracy remains in very high levels.

β	**Categories/ word**	**Accuracy**	**Sentence accuracy**	**Accuracy (POS tagger)**	**Sent. Accuracy (POS tagger)**
0.1	1.4	97.0	62.6	96.4	57.4
0.075	1.5	97.4	65.9	96.8	60.6
0.05	1.7	97.8	70.2	97.3	64.4
0.01	2.9	98.5	78.4	98.2	74.2
$0.01_{k=100}$	3.5	98.9	83.6	98.6	78.9
0	21.9	99.1	84.8	99.0	83.0

Table 4.1: Results of the supertagger in Section 00 of CCGbank. Sentence accuracy refers to sentences with all words correctly tagged. The two last columns present results for automatically assigned POS tags (Clark and Curran, 2004a)

Due to the importance of the supertagging process in the creation of an efficient parser, and given the time limitations of this project, we decided to use the supertagger of Clark and Curran as a front end to our parser, instead of trying to implement our own

custom-made version. In order to do that, we created a wrapper class which encapsulates all the technical details of the interaction between the two programs. The wrapper passes the sentence to be parsed to the external program by defining appropriate β and *k* parameters, and then gets the output of the supertagger and processes it in order to create a data structure readable by our parser. From this point on, the parsing process proceeds with the usual way.

We should note that our use of the supertagger as a "black box" imposes an important restriction compared with the parser of Clark and Curran. In their case, the supertagger initially provides to the parser a minimum set of lexical categories for the words of the sentence; if the parser is not able to end up with a derivation, it then requests more categories from the supertagger. The process is repeated until the parser gets a full parse. This kind of close interaction between the parser and supertagger (called by the authors *adaptive supertagging*) has been found to provide much higher parsing speed (by a factor of 3) without any loss in accuracy. In our case, however, this degree of integration between the parser and the supertagger is not possible, since the overhead of repeatedly calling an external program from within our code would have eventually compromise the overall efficiency.

4.1.3 Probabilistic model

After the supertagger has assigned the lexical categories to the words of the sentence, the parsing process proceeds left-to-right and bottom-up, as shown in Figure 4.1. For every cell of the column that is currently processed, the algorithm attempts to combine every partial parse that can be found in a previous cell of the same row with every partial parse in a previous cell of the same column. As the derivation process proceeds to cover larger spans of the sentence, this produces an exponentially increasing number of entries in the cells, making eventually the parsing infeasible.

The role of a probabilistic model is to provide a score for each possible result, so the program will be able to keep only the most likely of them and to finish in a finite time. For our parser, we chose to implement the head-words dependency model (HWDep) of Hockenmaier (2003), shortly described in Section 3.1.1. The motivation for this decision is simple: Although the project has no ambition to compete the state-of-the-art parsers that are now available for CCG (something like that would be unrealistic for the scope of this work), we still need to take in consideration that our semantic analysis is syntax-driven and depends on the way with which syntactic derivation is proceeding.

So, by using a powerful model for guiding the syntactic derivation we can expect an at least acceptable performance for our parser, despite the fact that the limited time does not permit subtle parameter refinement or fine-grain adjustments of the model.

HWDep model

As we saw in Section 3.1.1, all Hockenmaier's models are based to a simple baseline model that uses four kinds of probabilities: expansion, head, non-head, and lexical. The difference in HWDep model is that in this case the head word and the category of the head word for a local tree are generated at the maximal projection (either at the root of the tree or when generating a non-head daughter S), and then these pieces of information are used as conditioning variables for generating the other elements. If exp is the way that the parent node is expanded, taking values from $\{left, right, unary, lexical\}$, P is the parent category (the category for the root of the tree), H the category of the head daughter, S the category of the non-head daughter, $\langle c_P, w_P \rangle$ the pair of category and word for the head of the tree, and $\langle c_S, w_S \rangle$ the pair of category and word for the head of the sister node, the new model uses the conditional probabilities shown in Table 4.2.

Description	**Conditional probability**
Expansion	$P(exp \mid P, c_P \# w_P)$
Head	$P(H \mid P, exp, c_P \# w_P)$
Non-head	$P(S \mid P, exp, H \# c_P \# w_P)$
Lexical category	$P(c_S \mid S \# H, exp, P)$ or $P(c_{TOP} \mid P = TOP)$
Head word	$P(w_S \mid c_S \# P, H, S, w_P)$ or $P(w_{TOP} \mid c_P)$

Table 4.2: The conditional probabilities of the HWDep model (Hockenmaier, 2003). The # symbols indicate interpolation levels (explained in the next section).

So, in this model, expansion probability, head probability, and non-head probability are now also conditioned on the lexical head $\langle c_P, w_P \rangle$. These additions to the model has been found by Hockenmaier to increase the labeled dependencies[1] performance compared with the baseline model by 4.8%. The last row of the above table indicates that the head word of the sister node is generated by conditioning on the head word of the local tree, among other information. This kind of dependency that holds between the head words of the sibling nodes improves the overall labeled dependencies

[1] Briefly explained in Section 5.1.

performance by a further 3.5%.

Applying beam search

Equipped with our model, we are now in position to assign scores to the results. In this step essentially we are dealing with line 6 of the CKY pseudo-code (Algorithm 4.1), where a list of results is returned with the categories produced by every possible combination $\langle B, C \rangle$ that is permitted by some CCG rule. Instead of adding all these results to the chart cell, we first apply a simple beam-search approach, adding only those results that fall within a specific factor of the result with the highest probability. This factor depends on an initial parameter γ, known as *base beam factor*, and the size of the list with the following way:

$$f(\gamma, |l|) = \gamma |l|^2 \tag{4.2}$$

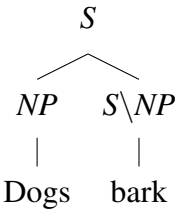
Essentially, what this technique achieves is that when the length of the list is small then the beam width f is very wide, so the majority of the results will be accepted; on the other hand, when the number of the returned results is high, the beam width is relatively narrow, and only a small portion of them will be accepted (Roark, 2001).

We will close this section with some notes about the implementation of the probabilistic model: We trained our models on Sections 02-21 of CCGbank using a Python script, which prepared one text file for each model (each conditional probability). From the parser's side, we collected all the necessary functionality of the probabilistic manipulation in a single Java class, in order to introduce the minimum possible amount of modifications to the existing OpenCCG code. From a software engineering perspective, our design aimed to impose the lowest possible *coupling* (the degree in which one class depends on other classes) and the highest *cohesion* (how strongly correlated is the content of a class). This will be a typical strategy for every aspect of this project.

4.1.4 Dealing with sparse data

The most important problem of any probabilistic model is that no matter how large our training corpus is, there will always be cases that they were not contained in the training data, and so the model is not able to rank them properly. For our model to be in position to assign a score in the following local tree,

(2) Dogs bark:



there should be at least one identical instance in the training corpus (Section 02-21), otherwise the model will return 0. Actually, this is exactly the number we are going to get, since this sentence, despite its simplicity, cannot be found in Sections 02-21 of CCGbank. The zero score will be multiplied with scores of other partial derivations, and eventually will invalidate (by zeroing its overall score) a possibly legitimate parse.

One way to avoid such problems is to *linearly interpolate* the empirical estimates we have calculated from our training data with (possibly smoothed) estimates coming from less specific distributions. If our empirical estimate is $\hat{e}_i$ and the less specific estimate is $\tilde{e}_{i-1}$, then the linear interpolation has the form:

$$\tilde{e}_i = \lambda \hat{e}_i + (1-\lambda)\tilde{e}_{i-1} \tag{4.3}$$

where λ is a weight that depends on the particular distribution (we will see in a minute how this weight is calculated). Note that the above definition is recursive: with 3 levels of interpolation, for example, Equation 4.3 becomes:

$$\tilde{e}_3 = \lambda_3 \hat{e}_3 + (1-\lambda_3)[\lambda_2 \tilde{e}_2 + (1-\lambda_2)\tilde{e}_1] \tag{4.4}$$

In her models, Hockenmaier indicates levels of interpolation with the symbol # (Table 4.2). For example, the non-head probability $P(S|P,exp,H\#c_P\#w_P)$ contains two levels of interpolation: $P(S|P,exp,H,c_P,w_P)$ is interpolated with $P(S|P,exp,H,c_P)$, which in turn is an interpolation of $P(S|P,exp,H,c_P)$ and $P(S|P,exp,H)$.

For our model, we decided to introduce some additional levels of interpolation, trying to avoid as much as possible the problems caused by the sparseness of data. More specifically, we augmented the expansion probability to $P(exp|P*c_P\#w_P)$ and the head probability to $P(H|P,exp*c_P\#w_P)$, with the $*$ symbol indicating the new levels of backoff. This addition indeed caused a 2% improvement to the parsing results.

The last thing that remains to be seen is how we calculate the interpolation weights. We followed Collins (1999, p. 185), who suggests that if the more specific estimate is equal to $\frac{n_i}{f_i}$ (so n_i is the partial count and f_i the total), the value of λ_i can be found using the following formula:

$$\lambda_i = \frac{f_i}{f_i + 5u_i} \tag{4.5}$$

Here, u_i is the *diversity* of f_i, that is, the number of different outcomes that have been seen in the context of f_i in the training corpus. This treatment of weights is indeed very reasonable: each interpolation weight λ_i depends on the sample size f_i on which the specific distribution is based. As this sample increases, the distribution becomes more reliable, and this is reflected by a higher weight. The smaller the sample, the closer λ_i is to 0, so the contribution of the specific backoff level to the overall result will be minimal. The inclusion of the diversity term u_i in the denominator makes also sense: a high diversity means that there is high probability for a new outcome to be observed, so this penalizes λ_i.

4.1.5 Treatment of coordination and punctuation

Coordination and punctuation constitute special cases for every parser, and usually require ad-hoc machinery for their treatment. However, as a lexicalized grammar, CCG provides us the necessary tools to handle this issues in the context of the formalism itself. For coordination, we can imagine that a proper category for a conjunction would have the form $(X\backslash X)/X$, allowing derivations of the following form:

(3)

John	likes	peanuts	and	spinach
NP	$(S\backslash NP)/NP$	NP	$(NP\backslash NP)/NP$	NP
			$NP\backslash NP$ (>)	
		NP (<)		
	$S\backslash NP$ (>)			
S (<)				

However, such a naïve treatment can also lead to violations of the across the board condition (ATB; Ross, 1967) and over-generation, like in following case (adapted by Baldridge, 2002):

(4) ∗

player	that	shoots	and	he misses
N	$(N\backslash N)/(S\backslash NP)$	$S\backslash NP$	$(S\backslash S)/S$	S
			$S\backslash S$ (>)	
		$S\backslash NP$ (<B)		
	$N\backslash N$ (>)			
N (<)				

This was the reason for which in the original form of CCG Mark Steedman opted to address coordination by introducing a ternary rule, which consumes both coordinated arguments simultaneously:

(5) *Coordination*:
$X \quad CONJ \quad X \quad \Rightarrow \quad X \quad (> \Phi^n)$

The problem here was that since the ternary rule did not correspond to some CCG combinator, the solution seemed rather ad-hoc and ill-fitting in the generic CCG framework[2]. A better approach to coordination was eventually made possible by the work of Jason Baldridge, who in 2002 introduced as part of his PhD thesis an extension of CCG, incorporating category modalities (Baldridge, 2002). In this multi-modal version, every slash in a category can be decorated with one of four types $\{\star, \times, \diamond, \cdot\}$, imposing certain restrictions to the way that the categories can be combined. The $\star$ type is the most restrictive, allowing only the use of functional application. In the light of this new feature, we can now avoid the over-generation observed in (4) by using for our conjunctions the category $(X\backslash_\star X)/_\star X$.

OpenCCG provides us the tools to apply such a solution, since it supports modalities and also incorporates the notion of dollar variables, that is, variables that can range over a stack of arguments (the concept of dollar variables is based on the \$ convention, as explained in Section 2.2.1). For our parser, then, we assign to every conjunction the following set of lexical categories:

(6) a. $(S\$_1\backslash_\star S\$_1)/_\star S\$_1$
b. $(NP\$_1\backslash_\star NP\$_1)/_\star NP\$_1$
c. $(N\$_1\backslash_\star N\$_1)/_\star N\$_1$

The first category allows the coordination over a wide range of clausal categories, while the other two aim to capture coordination cases between various forms of noun phrases and nouns. Equipped with this feature, our parser gives the derivation showed in Figure 4.2 for the rather complicated case of the sentence "I like music and cinema but I know you don't and nothing can changes that".

For the case of punctuation, we decided to provide a very simple solution. More specifically, we introduced a new OpenCCG punctuation rule that combines every punctuation mark to the previous constituent, essentially forcing the parser to ignore

[2]More specifically, the ternary rule did not conform to the *Principle of Combinatory Type Transparency*, as stated in Steedman (2000).

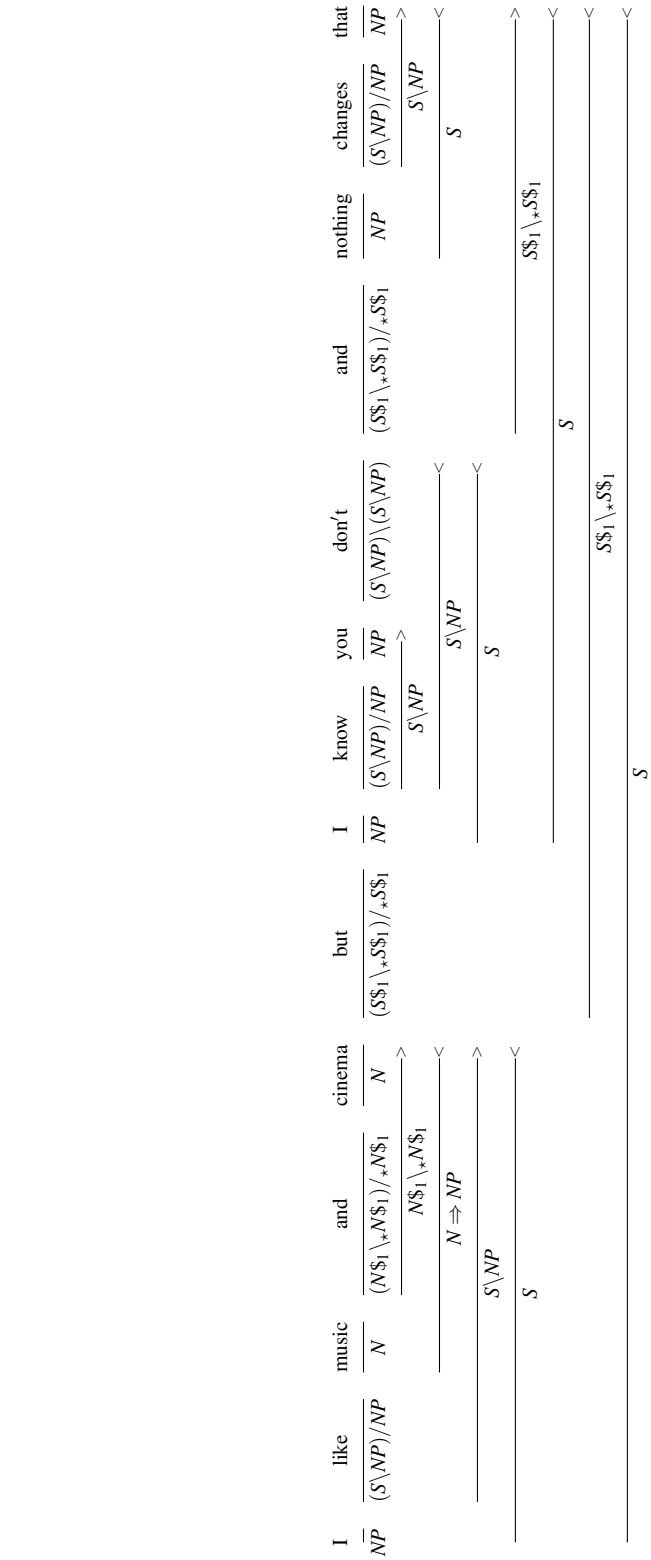


Figure 4.2: A CCG derivation for a sentence with multiple conjunctions

any punctuation marks. Of course, such an account is an oversimplification. In a proper treatment, the absorption of punctuation should be performed according to the type of the punctuation mark and the categories of adjacent constituents. Clark and Curran's parser, for example, includes a large number of punctuation rules defining in every detail what category can absorb what kind of punctuation mark according to its relative position (right or left of the constituent). Some cases are even more complicated, like the sentence "I like movies, books and music", where comma has essentially the role of a conjunction. Despite these obvious shortcomings, our solution was a decent trade-off between the available time and efficiency, and has been found to work well in practice.

4.1.6 Type-changing rules

Beyond the standard combinatory rules presented in Section 2.2.1, CCGbank includes a number of unary rules that convert a CCG category to some other category. These unary rules usually serve to convert a verb phrase to a modifier. Such a case, for example, is the following:

(7) $$\dfrac{\dfrac{\text{the millions of dollars}}{NP} \quad \dfrac{\text{it generates}}{S/NP \Rightarrow NP\backslash NP}}{NP}\,{<}$$

Sections 02-21 of CCGbank (our training corpus) contain about 190 distinct unary rules that cover a wide range of cases. However, most of them (about 150) are quite rare, with less than 10 occurences (on the other hand, the most frequent one was $N \Rightarrow NP$, with 115,541 occurences). For our parser we use a set of about 25 unary rules, which was derived by taking in consideration the frequency of the rules, the suggestions of Clark and Curran (2007, pp. 542-543), and the results of extensive experimentation in Section 00 of CCGbank, our development corpus.

4.1.7 Form of the syntactic derivations

The parser provides its output in text form through a series of lines, each of which corresponds to an assignment of a lexical category or an application of a CCG rule (binary or unary). The type of rule is showed inside parentheses at the beginning of the line. Figure 4.3 shows the output of the parser for a simple CCGbank sentence.

```
(lex)   A :- NP/N
(lex)   Lorillard :- N/N
(lex)   spokewoman :- N
(>)     Lorillard spokewoman :- N
(>)     A Lorillard spokewoman :- NP
(lex)   said :- (S\NP)/S
(lex)   , :- ,
(punct) said , :- (S\NP)/S
(lex)   This :- NP
(lex)   is :- (S\NP)/NP
(lex)   an :- NP/N
(lex)   old :- N/N
(lex)   story :- N
(lex)   . :- .
(punct) story . :- N
(>)     old story . :- N
(>)     an old story . :- NP
(>)     is an old story . :- S\NP
(<)     This is an old story . :- S
(>)     said , This is an old story . :- S\NP
(<)     A Lorillard spokewoman said , This is an old story . :- S
```

Figure 4.3: The output of the parser for a simple sentence

4.2 Adding semantics

The second part of this project (and the most relevant towards our purpose) was the implementation of the semantic aspect of the parser and the integration of this aspect with the probabilistic parser. At the time of this writing, no Java library was available for the logical form of this project (λ-calculus/first-order logic) to serve as the basis for an extended version with generalized skolem terms. The only option, therefore, was the creation of a new framework capable of performing the logical manipulation we have introduced in the previous sections of this dissertation, based on the λ-calculus and first-order combination introduced by Richard Montague.

We believe that this approach has many advantages: First, it offers us the necessary flexibility so that our design can focus solely on representation and efficiency issues, instead of expending our time trying to fit a new concept in something that it has not designed for this purpose in the first place; second, our library constitutes a useful tool for students and researchers that can be used as it is or be extended further in order

to incorporate other more advanced aspects of semantics (such negation or anaphora resolution) that we wouldn't be able to address here due to time limitations; and third, once again, the importance of such an effort from an educational perspective it was sufficient to justify the extra time and work.

4.2.1 An object-oriented design

The first challenge in creating a calculus software system is to find an efficient way to represent arbitrarily complex expressions. Let us examine for now a rather mild case (taken from Gabsdil and Striegnitz, 2000):

(8) Every owner of a hash bar gives every criminal a big kahuna burger:
$\forall x[(\exists y[hashbar(y) \wedge of(x,y)] \wedge owner(x)) \rightarrow$
$\forall z[criminal(z) \rightarrow \exists u[bigburger(u) \wedge gives(x,z,u)]]]$

Despite the apparent complexity of such an expression, we can immediately notice that in fact it is nothing else but an aggregation of other simpler expressions like $\exists y[hashbar(y) \wedge of(x,y)]$ or $\forall z[criminal(z) \rightarrow \exists u[bigburger(u) \wedge gives(x,z,u)]]$. Each sub-expression may, in turn, consists of other sub-expressions, and this structure ends to atomic formulas such as the predicates $criminal(z)$, $owner(x)$, or $gives(x,z,u)$.

In fact, the definition of a *well-formed formula (wff)* in predicate logic provides us all the necessary information for how a first-order formula should be constructed:

(9)
1. If R is a n-ary relation symbol, and $t_1, \ldots t_n$ are terms (variables, constants, and other relation symbols), then $R(t_1, \ldots t_n)$ is an atomic formula.
2. All atomics formulas are wffs.
3. If ϕ and ψ are wffs, then the same is true for $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, and $(\phi \rightarrow \psi)$.
4. If ϕ is a wff, and x is a variable, then both $\exists x\phi$ and $\forall x\phi$ are wffs.
5. Nothing else is a wff.

So we can see that no matter how complex a first-order formula is must follow one of the forms defined in 9(3) (or be an atomic formula), and the same must holds for all of its sub-expressions (one for negation, and two for conjunction, disjunction, and implication). This creates a disciplined nested structure that might not be immediately apparent because of the infix notation of first-order logic, where operators are written between the operands they act on. However, nothing in principle prevent us to write our

formulas using prefix notation, where operators are placed to the left of their operands. With this notation, the relations of 9(3) can be rewritten as $not(\phi)$, $and(\phi,\psi)$, $or(\phi,\psi)$, and $imp(\phi,\psi)$. Accordingly, the quantified expressions of 9(4) become $all(x,\phi)$ and $exists(x,\phi)$[3]. By using this new way of notation, the expression

(10) $\forall x[man(x) \rightarrow walks(x)]$

can now be written as:

(11) `all(x,imp(man(x),walks(x)))`

The above observations provide us the insight for the exact form of the data structure we should use for the representation of our logical forms. We adopt an object-oriented approach, where each expression is an object that contains other (possibly complex) sub-objects, representing the sub-expressions. For the formula of (11), for example, we can imagine a nested object of the following form:

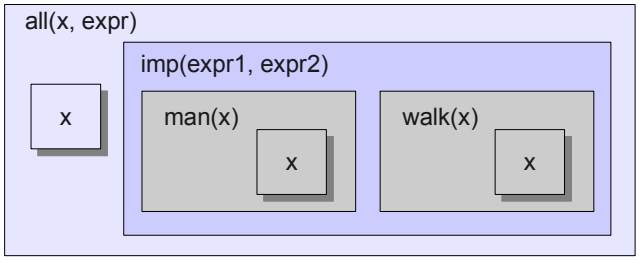


Figure 4.4: A nested object structure for a logical form

There are few more details we need to specify. The first is to decide the form of a λ-expression, that is, an expression that contains one or more λ-terms. This is easy: we will introduce a relation $lam(x,\phi)$ and we will adopt a format similar to that of the quantified expressions. The formula $\lambda x.\lambda y.loves(y,x)$, for example, can be now written as `lam(x,lam(y,loves(y,x)))` – a λ-expression having as an argument another λ-expression. We can use the same approach for an inclusion relation, which we are going to use for handling plurar nouns and generalized quantifiers like "most". For this purpose, we represent relations of the form $x \in y$ by a binary relation $inc(x,y)$.

Finally, we need a notation for functional application, the process of combining two logical forms in order to derive a new one. Again, we can simply introduce a binary

[3]Of course, our new theory does not require existentials. We include this relation just for demonstration purposes.

relation of the form $app(\phi, \psi)$. This will be an instruction to our system for applying the λ-calculus manipulation on the two expressions and producing a new logical form.

Figure 4.5 presents a diagram with the class hierarchy of our design. We can see that every relation introduced in the previous paragraphs is represented by a separate class. The basic point here is that all objects have the same superclass `Expression`, so for every object holds an *is-a* relationship with this class. For our purposes, this simply means that every object can be used as an argument (sub-expression) to every other object. This *object composition* technique allows us to represent arbitrarily complex expressions.

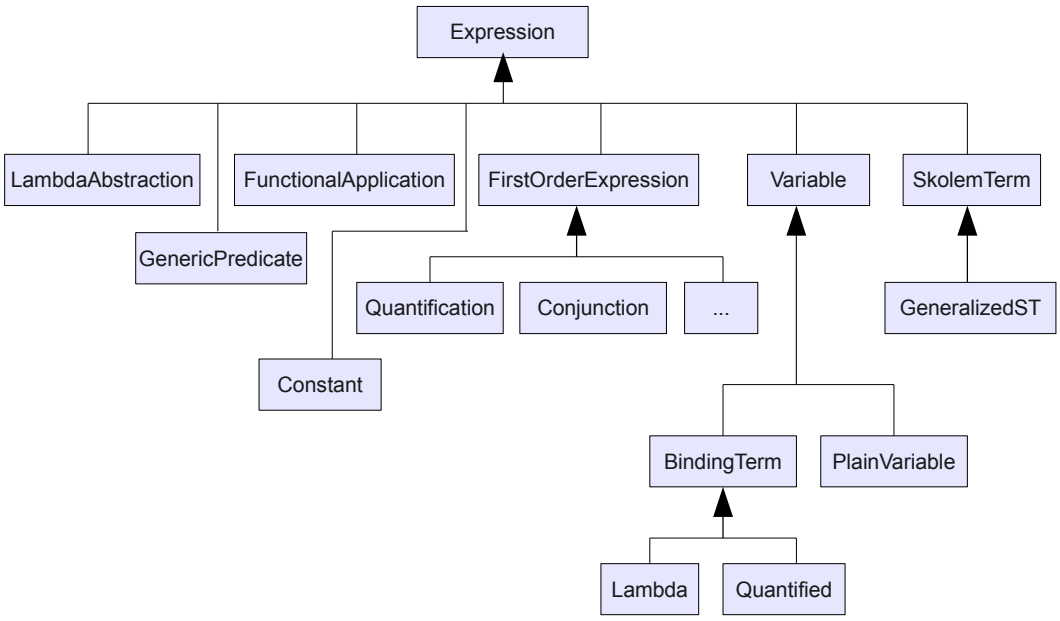


Figure 4.5: Class diagram for the λ-calculus Java library

4.2.2 Beta-conversion

The previous section described in some detail the basic data structures for our library, but it did not attempt any description of the underlying algorithmic mechanism. So, before we proceed to the discussion for the exact form of the generalized skolem terms objects, we need to address this important issue.

λ-calculus works with the following way: Every time that a λ-expression of the form $\lambda x.\phi$ has to be combined with some other expression ψ using functional application, the λx term at the left is thrown away and every occurrence of x in ϕ is replaced by ψ. This process is called β-*conversion* and lies at the core of a λ-calculus system.

In order to implement such a mechanism for our library, we follow the stack-based method described in Blackburn and Bos (2005) for their Prolog system. Our approach can be described as follows:

- When the expression that is currently been processed is an application, we push its argument to the stack, and we discard the outermost application object.
- If the expression is a λ-abstraction, we throw away the λ-term, and we pop the item at the top of the stack and substitute it for every occurrence of the correlated variable.
- If the expression we are working with is neither an application nor a λ-abstraction, we attempt to first β-convert its sub-expressions.

As an example[4], suppose that we need to get the logical form for the sentence "John loves Mary". In our prefix notation, and with the appropriate application relations added, our expression has the following form before β-conversion:

(12) `app(app(lam(x,lam(y,loves(y,x))),mary),john)`

In order to β-convert (12), we begin by pushing the expression in the stack. The rest of the process is described in Table 4.3.

	Expression	**Stack**
1	`app(app(lam(x,lam(y,loves(y,x))),mary),john)`	`[]`
2	`app(lam(x,lam(y,loves(y,x))),mary)`	`[john]`
3	`lam(x,lam(y,loves(y,x)))`	`[mary,john]`
4	`lam(y,loves(y,mary))`	`[john]`
5	`loves(john,mary)`	`[]`

Table 4.3: β-conversion for the sentence "John loves Mary"

In Step 1, the argument of the application, `john`, is pushed in the stack and we throw away the outer object. Our new expression is again an application, so in Step 2 we again push its argument, `mary`, to the stack and throw away the outer object. Now in Step 3 we have a λ-abstraction, so we pop the first entry of the stack (`mary`) and substitute for every occurrence of *x* (only one here, in predicate *loves*). By discarding the outer object, we get another λ-abstraction in Step 4. We repeat the same procedure for `john`, and this leaves us with an empty stack and the β-converted form of Step 5[5].

[4] Adapted by Blackburn and Bos (2005), p. 76.

[5] A more complicated case is demonstrated in Table 4.4.

4.2.3 Alpha-conversion

The mechanism introduced in the previous section requires an important addition in order to function properly. Specifically, there are certain problems that can arise from careless use of variable names in the expressions that participate in an application. Imagine, for example, that we wish to combine the logical form for "every man" $\lambda q.\forall x[man(x) \rightarrow q(x)]$ with the logical form of "loves a woman" $\lambda y.\exists x[woman(x) \wedge love(y,x)]$[6]. The first steps of β-reduction we will produce the intermediate form

(13) $\forall x[man(x) \rightarrow \lambda y.\exists x[woman(x) \wedge love(y,x)](x)]$

where the notation $\lambda y.\exists x[woman(x) \wedge love(y,x)](x)$ is used to denote application between the λ-abstraction at the left and the variable x. The problem now is obvious: such an application will substitute x for every occurrence of y in the λ-abstraction, producing the following wrong result:

(14) $*\forall x[man(x) \rightarrow \exists x[woman(x) \wedge love(x,x)]]$

The solution to this problem is to replace the expression that acts as a functor, in this example, $\lambda y.\exists x[woman(x) \wedge love(y,x)]$, with some other α-*equivalent* expression in which the variables bound by some quantifier are not in conflict with the variable that acts as an argument. One such expression in the previous case would be $\lambda y.\exists z[woman(z) \wedge love(y,z)]$. In that case, the application of this expression to x would provide the result:

(15) $\forall x[man(x) \rightarrow \exists z[woman(z) \wedge love(x,z)]]$

which is a correct representation of the meaning of the sentence "Every man loves a woman".

In our design, the responsibility of α-conversion has been delegated to λ-terms (represented by objects of type `LambdaVariable`). Before the λ-term object proceeds to substitute its bounded variable with some argument, it scans the overall object structure to verify if there are any α-conversions that should be performed first. If such a case is located, the λ-term selects a name not used anywhere else and renames accordingly the problematic variable.

[6] Again, we will use a non-skolemized form of a formula since it serves better our purposes for this example.

4.2.4 Self-application and other transformations

In the context of the λ-calculus operations, some non-typical cases of manipulation can emerge that need special treatment by our library. Perhaps the most frequent of them is the case of self-application, where a λ-expression waits for a predicate that it then applies on itself. This is, for example, the typical logical form for NPs:

(16) a. Mary $\vdash NP : \lambda q.q(mary)$

b. a book $\vdash NP : \lambda q.q(skolem(book))$

Suppose we want to combine the logical form of a transitive verb, $\lambda x.\lambda y.loves(y,x)$, with the first of the above expressions. If we just follow the stack-based procedure described in Section 4.2.2, the term λx will be consumed by $\lambda q.q(mary)$ and we will end up with something like $\lambda y.loves(y,\lambda q.q(mary))$, which of course is not the expected outcome. An easy way to overcome the problem is to simply swap the positions of the two operands in the application before the β-conversion, that is, use $\lambda q.q(mary)$ as the left operand and $\lambda x.\lambda y.loves(y,x)$ as the right operand. This simple work-around will indeed give us the required result, but works only with very simple cases such those in (16). Let's imagine, for example, that our parser needs to combine the logical forms below:

(17) a. loves $\vdash (S\backslash NP)/NP : \lambda x.\lambda y.loves(y,x)$

b. every woman $\vdash NP : \lambda q.\forall y[woman(y) \rightarrow q(y)]$

The β-conversion between these two constituents will yield the following transformations: first, q is substituted by $\lambda x.\lambda y.loves(y,x)$, so the new expression (after the appropriate α-conversion) is $\forall y[woman(y) \rightarrow \lambda x.\lambda z.loves(z,x)(y)]$. This involves a further application for the sub-expression $\lambda x.\lambda z.loves(z,x)(y)$, with our final expression to be

(18) $\forall y[woman(y) \rightarrow \lambda z.loves(z,y)]$

However, in this form $\lambda z.loves(z,y)$ is not further reducible, since the remaining slot in the predicate *loves*, represented by term λz, will not have a chance to be substituted by any expression. In such cases, the λ-terms representing the unfilled slots should be moved outside of the brackets:

(19) $\lambda z.\forall y[woman(y) \rightarrow loves(z,y)]$

In order to capture such cases we apply a certain amount of pre-processing to the operands of an application. Instead of attempting to swap their positions, like in the case of self-application, we transform the left operand by introducing an additional λ-term that can be used for guiding the proper application of the right term. In our previous case, for example, the logical form of the transitive verb will be first transformed to

(20) $\lambda w.\lambda y.w(\lambda x.loves(y,x))$

It is instructive to see how this allows the right kind of β-conversion. Table 4.4 shows the detailed states of our β-conversion stack, but this time using a mixed notation in order to not clutter things (prefix for application, infix for everything else).

	Expression	**Stack**
1	$\texttt{app}\,(\lambda w.\lambda y.\texttt{app}\,(w,\lambda x.loves(y,x))\,),$ $\hookrightarrow \lambda q.\forall z[woman(z) \rightarrow \texttt{app}\,(q,z)\,])$	[ ]
2	$\lambda w.\lambda y.\texttt{app}\,(w,\lambda x.loves(y,x))$	$[\lambda q.\forall z[woman(z) \rightarrow \texttt{app}\,(q,z)\,]]$
3	$\lambda y.\texttt{app}\,(\lambda q.\forall z[woman(z) \rightarrow \texttt{app}\,(q,z)\,],\lambda x.loves(y,x))$	[ ]
3.1	$\texttt{app}\,(\lambda q.\forall z[woman(z) \rightarrow \texttt{app}\,(q,z)\,],\lambda x.loves(y,x))$	[ ]
3.2	$\lambda q.\forall z[woman(z) \rightarrow \texttt{app}\,(q,z)\,]$	$[\lambda x.loves(y,x)]$
3.3	$\forall z[woman(z) \rightarrow \texttt{app}\,(\lambda x.loves(y,x),z)\,]$	[ ]
3.3.1	$\texttt{app}\,(\lambda x.loves(y,x),z)$	[ ]
3.3.2	$\lambda x.loves(y,x)$	[z]
3.3.3	$loves(y,z)$	[ ]
3.4	$\forall z[woman(z) \rightarrow loves(y,z)]$	[ ]
4	$\lambda y.\forall z[woman(z) \rightarrow loves(y,z)]$	[ ]

Table 4.4: β-conversion for a more complicated case

The lines in which the numbering is getting more specific (e.g. from 3 to 3.1) indicate points where the outer expression at the left is not further reducible and the process has to β-convert its sub-expressions. Of course, this transformation technique has to be generalized for predicates of an arbitrary arity. So, for every application where one term has the form $\lambda q.(\ldots q(x)\ldots)$ and the other the form $\lambda x.\lambda y.\ldots pred(x,y,\ldots)$, the latter is transformed to:

(21) $\lambda w.\lambda y.\ldots w(\lambda x.pred(x,y,\ldots))$

4.2.5 Implementation of Generalized Skolem Terms

As we can see in the class diagram of Figure 4.5, our design supports the skolemization process by two different objects, `SkolemTerm` and `GeneralizedSkolemTerm`. The first object corresponds to the unspecified form of a skolem term, as discussed in Section 2.6. Such an object includes a list of specifications, which in the beginning of the derivation is empty. Every time the environment (that is, the set of quantifiers within whose scope the skolem term falls) changes, a new generalized skolem term object is created, and this new object is added to the list of the specifications. By the end of the derivation, the generic "unspecified" form will be linked with all possible alternative readings, as shown in Figure 4.6.

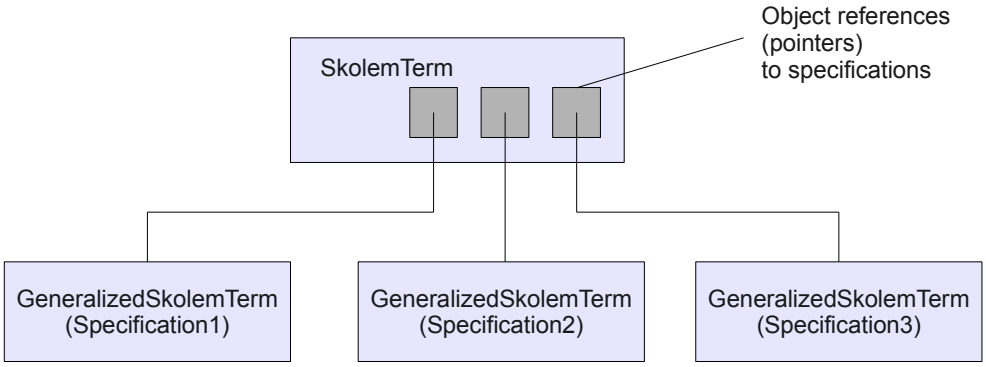


Figure 4.6: A skolem term object with its specifications

It is important at this point to clarify that the contents of the specification list are not complete objects but *object references*, that is, the Java equivalent of pointers in other languages (numbers that indicate the memory address in which the actual object is stored). In other words, the multiple specified readings are efficiently stored as a single linked packed structure that is shared among the other parts of the logical expression. For the sentence "A boy ate a pizza", this packed structure would have the following form at the end of the derivation (the brackets represent disjunctive packing of the chart):

$$(22) \quad ate(\left\{ \begin{array}{c} skolem' \\ sk' \end{array} \right\} boy', \left\{ \begin{array}{c} skolem' \\ sk' \end{array} \right\} pizza')$$

4.2.6 CKY modifications

Now we have finally set almost every detail that is required in order to modify the core of our parser, the CKY algorithm. Actually, this is a rather trivial task given all the

above preparation. The only thing we have to do is to introduce an additional step in the inner loop of the algorithm, which we call skolem term specification[7]. If Λ_A is the logical form of a result category A that has been produced by the application of some CCG rule, then this step will proceed as follows:

(23) *Skolem term specification:*

1. For each skolem term ST in the logical form Λ, collect the new environment of ST.
2. If the new environment is different than the old environment, specify a new Generalized Skolem Term and add it to the specifications list of ST.

The collection of the environment is actually trivial due to our design, and is shown in Figure 4.7. Since the skolem term forms part of a nested structure, the environment is always implicit and readily available, without requiring from us to pass some list of bound variables from the daughters to the parent, introducing further computational cost. The skolem term can simply backtrack from parent to parent, following the pointers, and collect each bound variable every time it reaches a quantified expression.

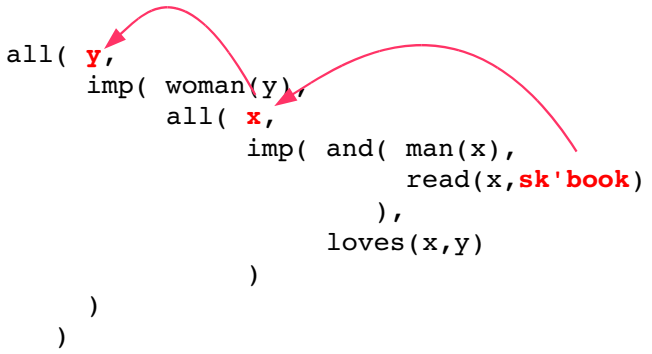


Figure 4.7: Collecting the environment of a skolem term

To make things clearer, we have included an example of a derivation for the simple sentence "Everybody needs somebody", that can be found in Figure 4.8. The figure at the top left (a) depicts the chart in its initial condition, after the assignment of the logical forms to every word. Note that word *somebody* has been assigned to the category $\lambda q.q(skolem(person))$, following our new semantic theory. Since this is the first step in the derivation, and there is no old environment for the skolem term $skolem(person)$, the system proceeds to a specification creating the "constant" skolem form $sk^{\{\}}_{person}$ and linking it with the unspecified form (part b). Next, the system combines *needs*

[7] See also Algorithm 2.1 in page 19.

with *somebody*, leaving the chart at the state showed in part c. In this case, there is no change to the environment, so the specification step does not alter the current state of the skolem term. Now the system proceeds to the combination of *everybody* with *needs somebody*, yielding the logical form of part d at the top-right cell of the chart. However, now the environment has been changed, since the skolem term in 4.8d falls within the scope of the quantified variable x. Therefore, the specification step yields a new generalized skolem term $sk^{\{x\}}_{person}$. So at the end of the derivation (part e) we have the required result: two specifications corresponding to all possible quantifier scopes (the final state also includes the unspecified form, which essentially is a "container" for all the specifications).

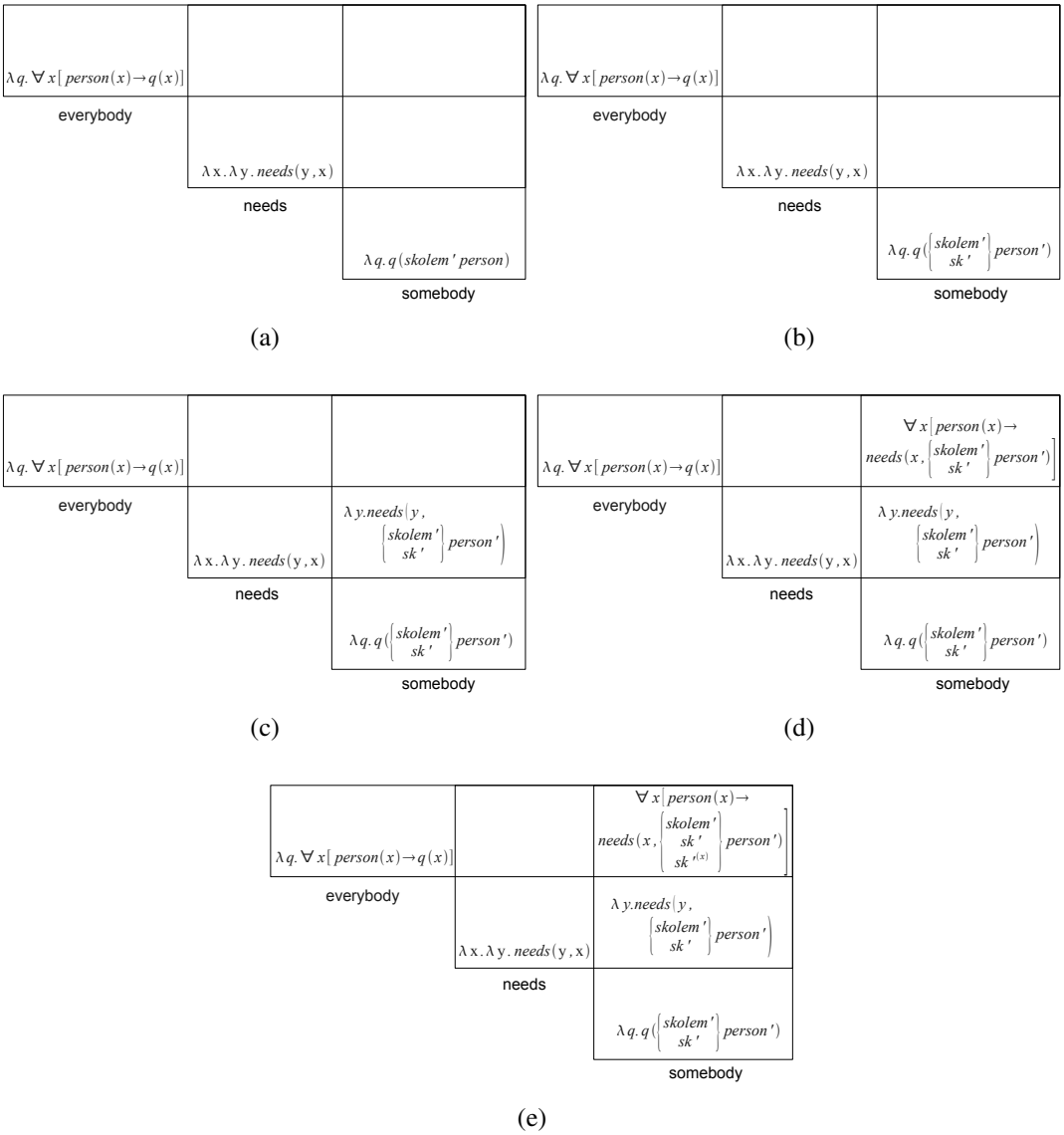


Figure 4.8: A sample derivation for the sentence "Everybody needs somebody"

4.2.7 The semantic lexicon

In this section we will describe the semantic lexicon of our parser, the repository for storing the semantic representations. We adopt a simple form that allows various degrees of grouping between categories and words. Each entry is comprised by a descriptive title, a list of CCG categories, a list of surface forms, and a logical expression in the prefix notation introduced in Section 4.2.1[8]. The entry for universal quantifiers, for example, could have the following form:

(24)
```
[universal]
categories: (S/(S\NP))/N|NP/N
words: every|each|all
LF: lam(p,lam(q,all(x,impl(app(p,x),app(q,x)))))
```

The lists of categories and words can be empty (but not at the same time for the same group): An empty category list means that every word in the word list will take the specified logical form, regardless its assigned CCG category; respectively, an empty word list means that all lexical entries that has been assigned to one of the specified categories should get the given logical form. This second case is useful for open-class lexical items. For such cases, the logical form contains the placeholder `<word>`, which is an instruction for the system to instantiate the lexical semantics using the surface form. The group for adjectives, for example, is the following:

(25)
```
[adjectives]
categories: N/N
LF: lam(p,lam(x,and(<word>(x),app(p,x))))
```

This instructs the parser that every word `w` with the category N/N should get the logical form $\lambda p.\lambda x.\mathtt{w}(x) \wedge p(x)$. On the other hand, groupings with both a category and a word list (like in 24), or with just a word list, represent closed-class lexical items, where the semantics should be defined for each lemma individually. In such cases, it is not unusual for the word list to contain just one lexical item, as in the case of the word *not*:

(26)
```
[not]
categories: (S\NP)\(S\NP)
```

[8]The only reason we use prefix notation in the lexicon is that it simplifies the translation of an expression to the internal nested object structure that our system uses. In principle, we could also use infix notation or some other representation compatible with our formalism.

```
words: not
LF: lam(v,lam(q,lam(f,not(app(app(v,q),f)))))
```

4.2.8 Integration of semantics to the system

Having all the necessary information, we are now in position to provide a general workflow of our system concerning the semantic manipulation: After the assignment of lexical categories by the supertagger, the parser consults the semantic lexicon in order to allocate appropriate semantic forms to every word. Then, control is passed to the CKY algorithm, which starts to produce new results according to CCG rules. Now, for every new result A that is accepted by the probabilistic model and has been produced by the application of a rule R to a pair of input categories $\langle B,C\rangle$, with logical forms $\langle \Lambda_B, \Lambda_C\rangle$, the semantic component creates a new λ-abstraction for $\langle \Lambda_B, \Lambda_C\rangle$ and attemps to β-convert it by taking in consideration the originating combinatory rule R. This produces a new semantic form Λ_A that will be assigned to result A. More specifically, we apply semantic support to forward and backward application (FAPP and BAPP, respectively) and all the versions of forward and backward composition (FCOMP and BCOMP, respectively)[9]. Table 4.5 presents the exact way in which the logical forms are combined.

Rule	**λ-abstraction**
FAPP(Λ_B, Λ_C)	$\Lambda_A = \texttt{app}(\Lambda_B, \Lambda_C)$
BAPP(Λ_B, Λ_C)	$\Lambda_A = \texttt{app}(\Lambda_C, \Lambda_B)$
FCOMP(Λ_B, Λ_C)	$\Lambda_A = \lambda\bar{x}.\texttt{app}(\Lambda_B, \texttt{app}(\bar{x}, \Lambda_C))$
BCOMP(Λ_B, Λ_C)	$\Lambda_A = \lambda\bar{x}.\texttt{app}(\Lambda_C, \texttt{app}(\Lambda_B, \bar{x}))$

Table 4.5: Mapping of syntactic rules to semantic transformations

The treatment of forward and backward application is straightforward: we just change the order of logical forms in the case of backward application. However, composition is more involved. In order for this rule to work properly, the λ-terms of the predicate should be substituted in reverse order, that is, the inner term first. So, in the above table, $\bar{x}$ represents a vector containing the outer λ-terms of the predicate that remain to be filled after the composition. Suppose, for example that we have the following derivation:

[9]Currently the system is not supporting semantics for substitution.

(27)
$$\dfrac{\dfrac{\dfrac{\dfrac{\text{I}}{NP}}{\dfrac{NP}{(S/NP)/NP}\,{:}\,\lambda q.q(me)}\,{>}\mathbf{T}\quad\dfrac{\text{gave}}{((S\backslash NP)/NP)/NP\,{:}\,\lambda x.\lambda y.\lambda z.give(z,x,y)}\quad\dfrac{\text{Mary}}{NP\,{:}\,\lambda q.q(Mary)}}{\dfrac{(S/NP)/NP}{:\,\lambda x.\lambda y.give(me,x,y)}\,{>}\mathbf{B}^2\quad\dfrac{\text{ten dollars}}{NP\,{:}\,\lambda q.q(dollars)}}}{S/NP\,{:}\,\lambda y.give(me,Mary,y)}\,{>}}{S\,{:}\,give(me,Mary,dollars)}\,{>}$$

In the forward composition betweeb I and $gave$, $\bar{x}$ is $(\lambda x, \lambda y)$. So, the semantic reformulation, as presented in the third row of Table 4.5, follows these steps:

(28) 1. $\lambda x.\lambda y.\texttt{app}\,(\lambda q.\texttt{app}\,(q, me)\,, \lambda z.give(z,x,y))$

2. $\lambda x.\lambda y.\,(\texttt{app}\,(\lambda z.give(z,x,y), me)$

3. $\lambda x.\lambda y.give(me,x,y)$

giving the right result. Finally, we apply semantic transformations for a small number of unary rules, including the type-raising case. For example, the type-change from a category N with logical form Λ_N to NP will result in a new logical form $\lambda q.q(skolem(\Lambda_N))$. For the type-raising case, a category C with logical form Λ_C will get the new form $\lambda q.q(\Lambda_C)$.

4.2.9 Semantic output of the program

With the addition of the semantic component, each step of a derivation is now augmented by the appropriate logical form. The logical forms are presented in a text infix notation, as shown in Figure 4.9.

Most of the forms should be self-explanatory. However, we need to describe the way in which the parser represents the new semantic elements, the skolem terms. We use a packed form to represent all the readings that are available by the end of the derivation. For example, the result for the sentence of Figure 4.9 in a more readable format is the following:

(29) $\forall x[theorem(x) \rightarrow proved(sk^{\{\}\{x\}}_{logician}, x)]$

The different environments superscripting a skolem term, here $sk^{\{\}\{x\}}_{logician}$, represent all the different specifications (i.e generalized skolem terms) that are linked with this skolem term by the end of the derivation. This notation represents disjunctive packing: A sentence with two skolem terms, for example, of the form $\forall x[\ldots sk^{\{\}\{x\}}_{nom_1} \ldots sk^{\{\}\{x\}}_{nom_2} \ldots]$ has four different readings:

```
(lex)       Some :- NP/N : lam:p.lam:q.q(skolem(p))
(lex)       logician :- N : lam:x.logician(x)
(>)         Some logician :- NP : lam:q.q(sk{lam:x.logician(x)}_{})
(lex)       proved :- (S\NP)/NP : lam:x.lam:y.proved(y,x)
(lex)       every :- NP/N : lam:p.lam:q.all:x[p(x)->q(x)]
(lex)       theorem :- N : lam:x.theorem(x)
(>)         every theorem :- NP : lam:q.all:x[theorem(x)->q(x)]
(>)         proved every theorem :- S\NP : lam:y.all:x[theorem(x)->proved(y,x)]
(gram)      type-changing3: S\NP => NP\NP
(tchange3) proved every theorem :- NP\NP : lam:y.all:x[theorem(x)->proved(y,x)]
(<)         Some logician proved every theorem :- NP :
                all:x[theorem(x)->proved(sk{lam:x.logician(x)}_{}_{x},x)]
```

Figure 4.9: The output of the parser with logical forms

(30) a. $\forall x[\ldots sk^{\{\}}_{nom_1}\ldots sk^{\{\}}_{nom_2}\ldots]$
b. $\forall x[\ldots sk^{\{\}}_{nom_1}\ldots sk^{\{x\}}_{nom_2}\ldots]$
c. $\forall x[\ldots sk^{\{x\}}_{nom_1}\ldots sk^{\{\}}_{nom_2}\ldots]$
d. $\forall x[\ldots sk^{\{x\}}_{nom_1}\ldots sk^{\{x\}}_{nom_2}\ldots]$

For the above example, then, our parser delivers two readings: One in which the same ingenious logicial proved all theorems, and one in which every theorem has been proved by some possibly different logicial.

(31) a. $\forall x[theorem(x) \rightarrow proved(sk^{\{\}}_{logician}, x)]$
b. $\forall x[theorem(x) \rightarrow proved(sk^{\{x\}}_{logician}, x)]$

There are some cases, however, where a parallel reading is enforced, as we saw in (30) in Chapter 2 for the Geach sentence. For such cases, the parser uses an index to indicate the parallelization, as follows:

(32) Every boy likes, and every girl detests, some saxophonist:

a.
```
all:y[boy(y)->likes(y,sk{lam:x.saxophonist(x)}_{}_{y}<1>)]/\
  all:y[girl(y)->detests(y,sk{lam:x.saxophonist(x)}_{}_{y}<1>)]
```

b. $\forall y[boy(y) \rightarrow likes(y, sk^{\{\}\{y\}}_{\langle 1\rangle:\lambda x.saxophonist(x)})] \wedge$
$\forall y[girl(y) \rightarrow detests(y, sk^{\{\}\{y\}}_{\langle 1\rangle:\lambda x.saxophonist(x)})]$

This form corresponds to the following two readings:

(33) a. $\forall x[boy(x) \rightarrow likes(x, sk^{\{\}}_{\lambda saxo})] \wedge \forall y[girl(y) \rightarrow detests(y, sk^{\{\}}_{saxo})]$

b. $\forall x[boy(x) \rightarrow likes(x, sk^{\{x\}}_{\lambda saxo})] \wedge \forall y[girl(y) \rightarrow detests(y, sk^{\{y\}}_{saxo})]$

In general, a sentence with logical form $\forall x[\ldots sk^{\mathcal{E}_1}_{\langle 1\rangle:nom_1} \ldots sk^{\mathcal{E}_n}_{\langle 1\rangle:nom_1} \ldots]$, where n is the number of different environments associated with the skolem term, has n distinct readings:

(34) a. $\forall x[\ldots sk^{\mathcal{E}_1}_{nom_1} \ldots sk^{\mathcal{E}_1}_{nom_1} \ldots]$

b. $\forall x[\ldots sk^{\mathcal{E}_2}_{nom_1} \ldots sk^{\mathcal{E}_2}_{nom_1} \ldots]$

...

c. $\forall x[\ldots sk^{\mathcal{E}_n}_{nom_1} \ldots sk^{\mathcal{E}_n}_{nom_1} \ldots]$

Chapter 5

Results

Our work for this project was essentially divided in two large parts: The construction of the probabilistic parser and the subsequent development and integration of the semantic component. We evaluate these two parts separately, mainly because despite their strong association a good performance for one component does not necessarily implies a similar performance for the other. Another practical reason is that although there are many well-established measures for the syntactic parsing, the same is not true for semantics. In fact, the evaluation of semantic analysis is usually performed in an ad-hoc fashion, producing results that are not interpretable according to a globally accepted standard. In the case of our work, the purpose was to provide a proof of concept that the theory we presented in Section 2.6 can offer an elegant solution to specific problems which can rise from quantifier scope ambiguities, and this is exactly the point in which we focus our evaluation efforts.

5.1 Syntactic parsing results

Traditionally, syntactic parsers are evaluated using the PARSEVAL metric suite (Black et al., 1991), which is based on the information retrieval notions of precision and recall. In the parsing context, *precision* shows us what fraction of the constituents returned by the parser is correct, according to some "gold standard". More formally:

$$P = \frac{correct\ constituents\ in\ parsing\ result}{total\ constituents\ in\ parsing\ result} \tag{5.1}$$

On the other hand, *recall* gives the fraction of the correct constituents in the gold standard that was eventually returned by the parser – in other words, it provides an indication of how many correct constituents the parser has missed.

$$R = \frac{correct\ constituents\ in\ parsing\ result}{total\ constituents\ in\ gold\ standard} \tag{5.2}$$

PARSEVAL provides two types of precision-recall metrics: one for unlabeled constituents, where the above equations are applied to correct *yields* (groupings of adjucent words) that the parser retrieves, and one for labeled constituents, where the metric takes also in consideration the category of each constituent. The problem of this method (and of any method that is based on the number of crossing brackets) is that it heavily penalizes the binary trees produced by a CCG parser compared with the flat trees of other CFG parsers. The problem is purely arithmetical: a binary tree contains many more bracketings than a non-binary, so the number of mismatches tends to be higher. As Hockenmaier (2003) notes, there is an even more important problem: in many cases, CCG produces many different but equivalent bracketings. Since the reference corpus will contain just one of them as the "gold standard" parse, the exact score assigned to the tree is a matter of chance.

For the above reasons, we also evaluate our model by examining the recovery of word-word dependencies on local trees. For a tree $\langle P,H,S\rangle$, where P, H, and D are the labels of the parent node, the head daugher, and the sister daughter in the local tree, respectively, $rel(w_H,w_S)$ holds if w_H is the head word of the head daughter and w_S the head word of the sister node. Again, there are two variations: the labeled dependencies measure takes in account the labels P, H, and S, while the unlabeled dependencies examines only the word-word relations.

Table 5.1 presents the results of our parser using the metrics discussed above, and provides a comparison with all the state-of-the-art parsers for CCG that have been developed so far. The table includes two more columns: *Coverage* shows the percentage of the test corpus for which our parser was able to get a parse, and *LexCat* is the percentage of the correcty assigned lexical categories.

Parser	**Cov.**	**LexCat**	**LP**	**LR**	**BP**	**BR**	$\langle$**P**,**H**,**S**$\rangle$	$\langle\rangle$
Clark et al. (2002)	95.0	90.3	–	–	–	–	81.8	90.0
Hockenmaier (2003)	99.8	92.2	82.2	82.4	86.2	86.4	85.1	91.4
Clark and Curran (2004b)	99.6	93.6	–	–	–	–	86.4	92.3
SemCCG parser	96.6	92.4	68.9	70.6	76.2	75.8	71.8	78.8

Table 5.1: Parsing results of SemCCG parser on Section 23 of CCGbank

The use of Clark and Curran's supertagger provides to our parser a lexical cate-

gory accuracy of 92.4%, and the performance of the probabilistic model is sufficiently acceptable to guide the derivations in a way that offers a coverage of 96.6%, making the parser really "wide-coverage". Of course, compared to state-of-the-art parsers, the results for PARSEVAL and head dependencies are less than optimal. We should note though that the fine-tuning of a propabilistic parser is a task worthing months of experimentation by its own right, and certainly this amount of time is not available in the context of an MSc thesis. Our intention was to get an acceptable performance that would support and not undermine the semantic manipulation which was the central point of this work. We think we achieved that.

5.2 Semantic evaluation

We evaluate the semantic aspect of our parser on about 30 selected sentences that present a wide range of linguistic challenges, most of which are related to the quantifier scope ambiguity problem. Additionally, we provide the logical forms as delivered by the parser for about 20 sentences from the quantifier section of FraCaS test suit (Cooper et al., 1996), as an independent set of test cases created by some other source.

One important problem we faced during this task was that in many cases the probabilistic model was simply too weak to provide us the expected kind of syntactic derivation that would allow the proper manipulation in the level of the logical form. To present an example, in the analysis of the sentence "Every man who reads a book loves a woman" the parser always opted to combine first *man* and *every* instead of creating the full form of the subject, *man who reads a book*. This blocked the semantic derivation, since the two constituents *every man* and *who reads a book* cannot be subsequently combined in a semantics level. This is shown in the following (partial) diagram:

(1)

Every	man	who	reads	a book
NP : $\lambda p.\lambda q.\forall x[p(x) \rightarrow q(x)]$	N : $\lambda x.man(x)$	$(NP\backslash NP)/(S\backslash NP)$: $\lambda q.\lambda n.\lambda y.n(y) \wedge q(y)$	$(S\backslash NP)/NP$: $\lambda x.\lambda y.reads(y,x)$	NP : $skolem(book)$
	NP : $\lambda q.\forall x[man(x) \rightarrow q(x)]$ (>)		$S\backslash NP$: $\lambda y.reads(y, skolem(book))$ (>)	
		$NP\backslash NP$: $\lambda n.\lambda y.n(y) \wedge reads(y, skolem(book))$ (>)		
		NP : $\times$ (<)		

The nature of such problems is purely statistical: The model disfavours the constituent *man who reads a book* simply because there is no data in the training corpus to support it, so at some point is pruned by the beam-search process[1]. Since our goal here

[1] In order to ensure that this was not some flaw of our model, we cross-checked the sentence by

was to test the semantic theory and the capacity of our library under very specific linguistic configurations, we needed a higher level of control on the way that constituents can be composed. So, in order to overcome such situations, we created a testing environment which allows us to control the exact way in which syntactic derivation takes place. The input of this environment is bracketed sentences, like the following:

(2) `(((every (man (who (reads (a book)>)>)>)<)> loves)F (every woman)>)>`

where the symbols `>`, `<` , `F`, and `B` state the rule that should apply to the specific bracketing (forward and backward application, and forward and backward composition, respectively).

All the logical forms provided below have been derived by the `toTex()` method of our library, in the form that is explained in detail in Section 4.2.9. Here, we briefly repeat for convenience the basic skolem term conventions: the different environments superscripting a skolem term, for example, $sk^{\{\}\{x\}}_{donkey}$, represent all the disjunctive readings that are available by the end of the derivation. On the other hand, a subscript of the form $\langle n \rangle$ denotes a parallelization between two or more skolem terms with the same nominal property.

5.2.1 Generic evaluation

The sentences have been divided in some general (and in some cases, rather arbitrary and overlapping) categories – when appropriate, we have included comments explaining the aspect being tested and how our program is handling the case.

General

Some simple sentences demonstrating the basic concept. In (2) we can also see a scope invertion case.

1. Everybody loves somebody:
   ▶ $\forall y[person(y) \rightarrow loves(y, sk^{\{\}\{y\}}_{\lambda x.person(x)})]$
2. Somebody loves everybody:
   ▶ $\forall x[person(x) \rightarrow loves(sk^{\{\}\{x\}}_{\lambda x.person(x)}, x)]$

getting a derivation from Clark and Curran's parser, which is also trained using CCGbank. We got exactly the same syntactic analysis.

Donkey sentences

Donkey sentences present a kind of quantifier problem first observed by Geach (1962). In cases like (3a), the pronoun seems to be bound in an existential quantifier associated with the NP *a donkey*. However, this is not allowed by any kind of syntactic analysis. The interpretation of the NP as a skolem term provides a solution, although a complete treatment would require some mechanism of pronominal anaphora resolution, binding the pro-term *pro* to the skolem term before the beginning of the derivation. Our tool does not carry such a mechanism, so the logical form below provides just an indication for the feasibility of this account.

3. (a) Every farmer who owns a donkey feeds it:
   ▶ $\forall x[farmer(x) \wedge owns(x, sk^{\{\}\{x\}}_{donkey}) \rightarrow feeds(x, pro)]$

   (b) Every farmer who owns a fat donkey feeds it:
   ▶ $\forall x[farmer(x) \wedge owns(x, sk^{\{\}\{x\}}_{\lambda x.fat(x) \wedge donkey(x)}) \rightarrow feeds(x, pro)]$

   The basic "donkey" concept is demonstrated in (3a). The interesting point of the variation in (3b) is that the nominal property of a skolem term can be an arbitrarily complex expression.

4. Some farmer owns every donkey:
   ▶ $\forall x[donkey(x) \rightarrow owns(sk^{\{\}\{x\}}_{\lambda x.farmer(x)}, x)]$

   Another case of scope inversion.

Scope asymmetries

In this category we examine cases involving possible asymmetries in the scope of noun phrases, as exemplified by the Geach sentence (Geach, 1972) in (5).

5. Every boy likes, and every girl detests, some saxophonist:
   ▶ $\forall y[boy(y) \rightarrow likes(y, sk^{\{\}\{y\}}_{\langle 3\rangle:\lambda x.saxophonist(x)})] \wedge$
   $\forall y[girl(y) \rightarrow detests(y, sk^{\{\}\{y\}}_{\langle 3\rangle:\lambda x.saxophonist(x)})]$

   The subscript $\langle 3\rangle$ in the above logical form[2] implies that the readings are not disjunctive – that is, the system returns only the two predicted readings, where the saxophonist retains the same scope for both constituents:

[2] In the output of the system in (5) we can see that the quantified variable is the same (y) for both boys and girls. This is actually not a problem, since the two quantified expressions are totally separate and there is no need for the parser to α-convert them.

(3) a. $\forall x[boy(x) \rightarrow likes(x, sk^{\{\}}_{saxo})] \wedge \forall y[girl(y) \rightarrow detests(y, sk^{\{\}}_{saxo})]$

b. $\forall x[boy(x) \rightarrow likes(x, sk^{\{x\}}_{saxo})] \wedge \forall y[girl(y) \rightarrow detests(y, sk^{\{y\}}_{saxo})]$

6. Some woman detests every saxophonist and every pianist:

   ▶ $\forall z[saxophonist(z) \rightarrow detests(sk^{\{\}\{z\}}_{\langle 3\rangle:\lambda x.woman(x)}, z)]$
   $\wedge \forall a[pianist(a) \rightarrow detests(sk^{\{\}\{a\}}_{\langle 3\rangle:\lambda x.woman(x)}, a)]$

   ▶ $\forall x[saxophonist(x) \rightarrow \forall z[pianist(z) \rightarrow detests(sk^{\{\}\{z,x\}}_{\langle 3\rangle:\lambda x.woman(x)}, x)$
   $\wedge detests(sk^{\{\}\{z,x\}}_{\langle 3\rangle:\lambda x.woman(x)}, z)]]$

   In both logical forms delivered for this sentence there are only two distinct readings (as denoted by the subscripts of skolem terms), one in which *some woman* has wide scope, and one in which it has narrow scope and is dependent on both universals. Again, there is no mixed reading.

7. Some woman detests every saxophonist and likes every pianist:

   ▶ $\forall z[saxophonist(z) \rightarrow detests(sk^{\{\}\{z\}}_{\langle 3\rangle:\lambda x.woman(x)}, z)]$
   $\wedge \forall z[pianist(z) \rightarrow likes(sk^{\{\}\{z\}}_{\langle 3\rangle:\lambda x.woman(x)}, z)]$

   Another variation of a sentence that could result in a possible scope asymmetry. The system once again delivers only two distinct readings.

8. (a) Some woman likes and John detests every saxophonist:

   ▶ $\forall x[saxophonist(x) \rightarrow likes(sk^{\{\}\{x\}}_{\lambda x.woman(x)}, x) \wedge detests(john, x)]$

   (b) Some woman likes and every man detests every saxophonist:

   ▶ $\forall x[saxophonist(x) \rightarrow likes(sk^{\{\}\{x\}}_{\lambda x.woman(x)}, x) \wedge \forall y[man(y) \rightarrow detests(y, x)]]$

   The above sentences are interesting cases where a non-paraller reading is allowed. Here, *some woman* can be a constant or a function of *every saxophonist* irrespectively of the state of the other constituent. In (8a) this other constituent is a constant (*John*), while in (8b) is a universal. In both cases the system returns the correct interpretations.

Intermediate scope

The following sentences present complex dependencies between the so-called existential and universal quantifiers that can lead to intermediate scope readings.

9. Every man who reads a book loves every woman:

   ▶ $\forall x[man(x) \wedge reads(x, sk^{\{\}\{x\}}_{book}) \rightarrow \forall z[woman(z) \rightarrow loves(x, z)]]$

   ▶ $\forall x[woman(x) \rightarrow \forall y[man(y) \wedge reads(y, sk^{\{\}\{y\}\{y,x\}}_{book}) \rightarrow loves(y, x)]]$

This case is exhaustively examined in Steedman (2010). If we unpack the above logical forms (each one of which corresponds to a different syntactic derivation), we get the following five readings:

(4) a. $\forall x[man(x) \wedge reads(x, sk^{\{\}}_{book}) \rightarrow \forall z[woman(z) \rightarrow loves(x,z)]]$
b. $\forall x[man(x) \wedge reads(x, sk^{\{x\}}_{book}) \rightarrow \forall z[woman(z) \rightarrow loves(x,z)]]$
c. $\forall x[woman(x) \rightarrow \forall y[man(y) \wedge reads(y, sk^{\{\}}_{book}) \rightarrow loves(y,x)]]$
d. $\forall x[woman(x) \rightarrow \forall y[man(y) \wedge reads(y, sk^{\{y\}}_{book}) \rightarrow loves(y,x)]]$
e. $\forall x[woman(x) \rightarrow \forall y[man(y) \wedge reads(y, sk^{\{y,x\}}_{book}) \rightarrow loves(y,x)]]$

We can see that readings (4a)-(4c) and (4b)-(4d) differ only in the order of the two universal quantifiers, so we can consider them model-theoretically equivalent: The first pair corresponds to the wide-scope reading for *a book*, while the second pair is the narrow scope reading, where *a book* depends on men. There is also a third reading (4e), where the skolem term depends on both men and women. However, the fact that there is no reading in which *a book* is dependent solely on women means that (4e) is again model-theoretically equivalent to the narrow scope reading, where books depends only on men. So, equipped with a rather trivial filtering mechanism, our parser would be able to provide the following output:

(5) a. $\forall y[man(y) \wedge reads(y, sk^{\{\}}_{book}) \rightarrow \forall x[woman(x) \rightarrow loves(y,x)]]$
b. $\forall y[man(y) \wedge reads(y, sk^{\{y\}}_{book}) \rightarrow \forall x[woman(x) \rightarrow loves(y,x)]]$

As Steedman notes, this constitutes an important difference from underspecification and storage approaches, where the decoupling between syntactic and semantic derivation allows the problematic reading where *a book* depends solely on women (Hobbs and Shieber, 1987; Cooper, 1983; Copestake and Flickinger, 2000).

10. (a) Some teacher showed every pupil every movie:
► $\forall x[movie(x) \rightarrow \forall y[pupil(y) \rightarrow show(sk^{\{\}\{y\}\{y,x\}}_{\lambda x.teacher(x)}, y, x)]]$
(b) Every student studied every paper by some author:
► $\forall x[student(x) \rightarrow \forall z[paper(z) \wedge by(z, sk^{\{\}\{z\}\{z,x\}}_{\lambda x.author(x)}) \rightarrow study(x,z)]]$

Some variations of the previous case. In (10a) there is no reading where the teacher depends solely on movies, while in (10b) again an author is not dependent solely on students.

Spurious readings

The benefit of eliminating existential quantifiers from a logical form is the simplification of the form and the restriction of spurious (equivalent) readings that can be produced during the derivation. The following group of sentences demonstrates that in every case the system derives only the necessary readings.

11. (a) Some representative showed some company some sample:
    ▶ $show(sk^{\{\}}_{\lambda x.repr(x)}, sk^{\{\}}_{\lambda x.company(x)}, sk^{\{\}}_{\lambda x.sample(x)})$
    (b) Some representative of some company saw some sample:
    ▶ $see(sk^{\{\}}_{\lambda x.repr(x) \wedge of(x, sk^{\{\}}_{\lambda x.company(x)})}, sk^{\{\}}_{\lambda x.sample(x)})$

    The output for these sentences shows just one reading, since there is no universal quantifier within the scope of which any skolem term can fall. The second sentence also presents a more interesting case of a complex nominal property, embedding another skolem term.

12. (a) Every representative of some company saw some sample:
    ▶ $\forall y[repr(y) \wedge of(y, sk^{\{\}\{y\}}_{\lambda x.company(x)}) \rightarrow see(y, sk^{\{\}\{y\}}_{\lambda x.sample(x)})]$
    (b) Every representative showed some company some sample:
    ▶ $\forall z[repr(z) \rightarrow show(z, sk^{\{\}\{z\}}_{\lambda x.company(x)}, sk^{\{\}\{z\}}_{\lambda x.sample(x)})]$

    Both of the above sentences have four readings, since the *sample* and the *company* skolem terms can depend or not on the representative, based on their exact specification time.

Coordination

The purpose of this group is to examine a wide range of coordination cases and how the semantic compontent addresses them.

13. John likes peanuts and spinach:
    ▶ $likes(john, peanuts \wedge spinach)$

    Simple coordination over NPs.

14. I like and you detest music
    ▶ $like(me, music) \wedge detest(you, music)$

    A simple case of coordination between verb phrases.

15. (a) Every man walks and talks:
    ▶ $\forall x[man(x) \rightarrow walks(x) \wedge talks(x)]$

    (b) Every man walks or talks:
    ▶ $\forall x[man(x) \rightarrow walks(x) \vee talks(x)]$

    The above sentences test the distribution of universal quantifiers over conjunction and disjunction, with the expected results.

16. (a) Some man walks and talks:
    ▶ $walks(sk^{\{\}}_{\langle 3\rangle:\lambda x.man(x)}) \wedge talks(sk^{\{\}}_{\langle 3\rangle:\lambda x.man(x)})$

    (b) Some man walks or talks:
    ▶ $walks(sk^{\{\}}_{\langle 3\rangle:\lambda x.man(x)}) \vee talks(sk^{\{\}}_{\langle 3\rangle:\lambda x.man(x)})$

    Distributivity of skolem terms over conjunction and disjunction. Although redundant in this case, since there is only one available reading, we can see that the identical index $\langle 3\rangle$ enforces a paraller interpretation of the coordinated parts.

17. (a) Every man and every woman:
    ▶ $\lambda x.\forall y[man(y) \rightarrow x(y)] \wedge \forall y[woman(y) \rightarrow x(y)]$

    (b) Every man and every woman walks:
    ▶ $\forall y[man(y) \rightarrow walks(y)] \wedge \forall y[woman(y) \rightarrow walks(y)]$

    (c) Every man and every woman likes chocolate:
    ▶ $\forall y[man(y) \rightarrow likes(y, chocolate)] \wedge \forall y[woman(y) \rightarrow likes(y, chocolate)]$

    (d) Every man and every woman walks or talks:
    ▶ $\forall y[man(y) \rightarrow walks(y) \vee talks(y)] \wedge \forall y[woman(y) \rightarrow walks(y) \vee talks(y)]$

    Coordination of universally quantified NPs for a range of cases. (17a) demonstrates the basic logical form, (17b) and (17c) show the behaviour on intransitive and transitive verbs, and (17d) presents a more complicated case including an additional level of distribution over disjunction.

18. (a) Some man and some woman:
    ▶ $\lambda r.r(sk^{\{\}}_{\lambda x.man(x)}) \wedge r(sk^{\{\}}_{\lambda x.woman(x)})$

    (b) Some man and some woman walks:
    ▶ $walks(sk^{\{\}}_{\lambda x.man(x)}) \wedge walks(sk^{\{\}}_{\lambda x.woman(x)})$

    (c) Some man and some woman likes chocolate:
    ▶ $likes(sk^{\{\}}_{\lambda x.man(x)}, chocolate) \wedge likes(sk^{\{\}}_{\lambda x.woman(x)}, chocolate)$

(d) Some man and some woman walks or talks:

$\blacktriangleright$ $(walks(sk^{\{\}}_{\langle 3\rangle:\lambda x.man(x)}) \vee talks(sk^{\{\}}_{\langle 3\rangle:\lambda x.man(x)}))$
$\wedge (walks(sk^{\{\}}_{\langle 4\rangle:\lambda x.woman(x)}) \vee talks(sk^{\{\}}_{\langle 4\rangle:\lambda x.woman(x)}))$

Coordination of indefinite NPs, in a setting similar to the one for universals in the previous sentence group. In (18a) we can see the basic form of a such a sentence, which is derived by using the following lexicon entry for the conjunction:

(6) and $\vdash (NP\$_1 \backslash NP\$_1)/NP\$_1 : \lambda x.\lambda y.\lambda r.r(y) \wedge r(x)$

Especially interesting is case (18d), where the indefinite NPs are distributed over an additional level of disjunction. The indices $\langle 3\rangle$ and $\langle 4\rangle$ enforce the parallel readings.

Generalized quantifiers

The implementation of generalized quantifiers like "most" or "few" in our system is still basic, since the cardinality properties of generalized skolem terms are simple strings that essentially are used as labels. Despite this, the parser can deliver the intendent meaning in most of the cases. In our experiments, every verb with plular agreement, say *read*, gets a category of the following form:

(7) read $\vdash (S\backslash NP)/NP : \lambda x.\lambda y.\forall z[z \in y \rightarrow read(z,x)]$

In sentences like "Four students read a book", this category allows both a collective reading, where all students read the same book (8a), and a distributive reading where each student reads a possibly different book (8b) – each reading corresponds to a different scoping possibility for the skolem term *a book*:

(8) a. $\forall z[z \in sk^{\{\}}_{students;four} \rightarrow read(z, sk^{\{\}}_{book})]$
b. $\forall z[z \in sk^{\{\}}_{students;four} \rightarrow read(z, sk^{\{z\}}_{book})]$

Note that if we just continue to use the conventional logical form $\lambda x.\lambda y.read(y,x)$, the system will be able to delive only the collective reading:

(9) $read(sk^{\{\}}_{students;four}, sk^{\{\}}_{book})$

A more extensive set of generalized quantifier cases, taken from FraCaS test suite, can be found in Section 5.2.2.

19. Most farmers own a donkey:
    ▶ $\forall z[z \in sk^{\{\}}_{farmers;most} \rightarrow own(z, sk^{\{\}\{z\}}_{donkey})]$

    A simple case of using *most*. The system delivers both a collective and a distributive reading.

20. Most farmers who own a donkey feed it:
    ▶ $\forall z[z \in sk^{\{\}}_{\lambda y.farmers(y) \wedge owns(y, sk^{\{\}}_{donkey});most} \rightarrow feed(z, pro)]$

    A slightly more complicated case. This differs from (19) in that the cardinality property *most* is now attached to the whole phrase "farmers who owns a donkey". However, this example reveals also a flaw of the system. The inner skolem term sk_{donkey} fails to specify a new generalized form $sk^{\{x\}}_{donkey}$ when it falls into the scope of the universal.

21. Three boys ate a pizza:
    ▶ $\forall z[z \in sk^{\{\}}_{\lambda x.boys(x);three} \rightarrow ate(z, sk^{\{\}\{z\}}_{\lambda x.pizza(x)})]$

    Again, the parser returns both readings.

5.2.2 Evaluation on Fracas framework

In this section we test our parser in a set of sentences taken by the FraCaS framework, a test suite created for textual entailment. The part of FraCaS that is devoted to generalized quantifiers contains 80 cases, each of which consists of one or two premises and a question. A textual entailment system must provide an answer to the question of the form "yes", "no", or "don't know", given the specific premises. What we did was to use the premises of the first set of cases as a test suite for our tool. The reason we did not test a larger number of sentences was purely practical, since for every sentence we had to prepare an appropriate lexicon for our testing environment manually. On the other hand, since the framework aims to test not quantifier scope ambiguities but entailment, most of the sentences were very similar to each other, so the usefulness of a larger test set from this source would be limited.

Most of the following sentences contain generalized quantifiers, an aspect in which our parser in its current form provides limited support. In order to get consice forms, we use the following logical form for the verb "are":

(10) are $\vdash (S\backslash NP)/NP : \lambda p.\lambda x.\forall z[z \in x \rightarrow p(z)]$

Cases where the logical form is wrong are marked with an asterisk ($*$).

1. An Italian became the greatest tenor:
   ▶ $became(sk^{\{\}}_{\lambda x.Italian(x)}, \lambda x.greatest(x) \wedge tenor(x))$

2. Every Italian man wants to be a great tenor:
   ▶ $\forall y[Italian(y) \wedge man(y) \rightarrow wants(y, sk^{\{\}\{y\}}_{\lambda x.great(x) \wedge tenor(x)})]$

3. Some Italian men are great tenors:
   ▶ $\forall z[z \in sk^{\{\}}_{\lambda x.Italian(x) \wedge men(x);some} \rightarrow great(z) \wedge tenors(z)]$

4. All Italian men want to be a great tenor:
   ▶ $\forall y[Italian(y) \wedge men(y) \rightarrow wants(y, sk^{\{\}\{y\}}_{\lambda x.great(x) \wedge tenor(x)})]$

5. $*$ Each Italian tenor wants to be great:
   ▶ $\forall x[Italian(x) \wedge tenor(x) \rightarrow wants(x, \lambda z.great(z))]$

6. No great tenors are modest:
   ▶ $\neg(\forall z[z \in \lambda x.great(x) \wedge tenors(x) \rightarrow modest(z)])$

7. Some Italian tenors are great:
   ▶ $\forall z[z \in sk^{\{\}}_{Italian(\lambda x.tenors(x));some} \rightarrow great(z)]$

8. The really ambitious tenors are Italian:
   ▶ $\forall z[z \in \lambda x.ambitious(x) \wedge tenors(x) \rightarrow Italian(z)]$

9. Some great tenors are Swedish:
   ▶ $\forall z[z \in sk^{\{\}}_{\lambda x.great(x) \wedge tenors(x);some} \rightarrow Swedish(z)]$

10. Many great tenors are German:
    ▶ $\forall z[z \in sk^{\{\}}_{\lambda x.great(x) \wedge tenors(x);many} \rightarrow German(z)]$

11. Several great tenors are British:
    ▶ $\forall z[z \in sk^{\{\}}_{\lambda x.great(x) \wedge tenors(x);several} \rightarrow British(z)]$

12. Most great tenors are Italian:
    ▶ $\forall z[z \in sk^{\{\}}_{\lambda x.great(x) \wedge tenors(x);most} \rightarrow Italian(z)]$

13. Some great tenors like popular music:
    ▶ $\forall z[z \in sk^{\{\}}_{\lambda x.great(x) \wedge tenors(x);some} \rightarrow like(z, music)]$

14. Few tenors are poor:
    ▶ $\forall z[z \in sk^{\{\}}_{\lambda x.tenors(x);few} \rightarrow poor(z)]$

15. Both leading tenors are excellent:
    ▶ $\forall z[z \in sk^{\{\}}_{\lambda x.leading(x) \wedge tenors(x);both} \rightarrow excellent(z)]$

16. $*$ Tenors who are excellent are indispensable:
    ▶ $\forall z[z \in \lambda y.tenors(y) \wedge \forall z[z \in y \rightarrow excellent(z)] \rightarrow indispensable(z)]$

17. An Irishman won the Nobel prize:
    ▶ $won(sk^{\{\}}_{\lambda x.Irishman(x)}, nobel)$

18. Most Europeans are residents in Europe:
    ▶ $\forall z[z \in sk^{\{\}}_{\lambda x.Europeans(x);most} \rightarrow residents(z) \wedge in(z, Europe)]$

5.3 Computational complexity issues

One of the arguments we used towards our approach in quantifier scoping in this dissertation was that the incorporation of the generalized skolem term concept in a CCG parser does not raise the theorerical power of the system, which has been found by Vijay-Shanker and Weir (1994) to be equivalent to that of a nested pushdown automaton. We would like to attempt to provide an informal explanation why this is actually true.

If we simplify things a little, we can imagine that at the core of a CCG parser lies a stack. At the beginning of the derivation, the parser starts by pushing in the stack the lexical categories for each word. In every case in which a CCG rule can be applied to the top categories in the stack, the parser does a *reduction*, that is, replaces these top categories with their result. In Figure 5.1, we can see that the first reduction for the sentence "I like swimming" happens in part d, where *like* and *swimming* are combined to a verb phrase. This new constituent can be further reduced with the next item in the stack, producing the final state that is shown in part e.

Of course, this description is an oversimplification. In a real CCG parser, every item of the stack could also have a stack-like form, accommodating more than one categories. In any case, it provides us a chance to show that this conceptual stack which lies at the core of the parsing process can actually also accommodate the generalized skolem term mechanism and the semantic manipulation.

Indeed, it is obvious that since the semantic manipulation takes places during the reduction stage, the stack at that time is neutral and can be used for the β-conversion process described in Section 4.2.2. We can imagine then that during the reduction

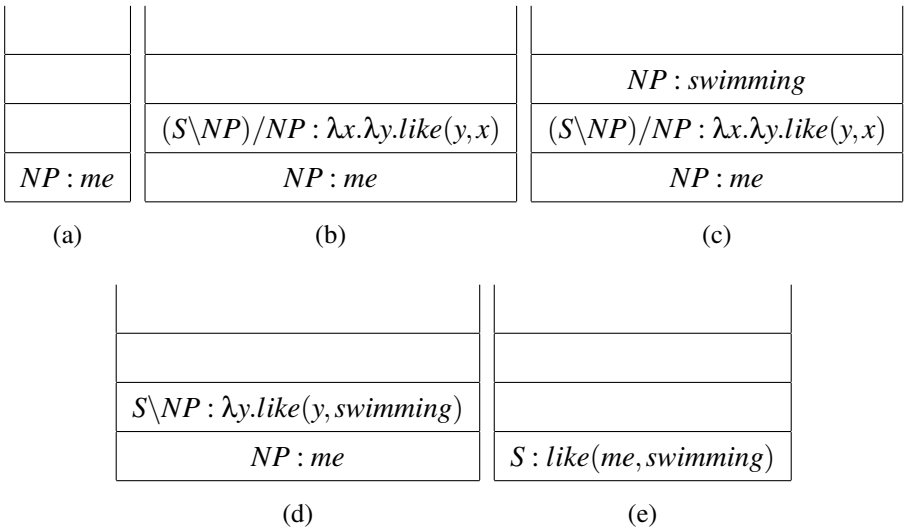


Figure 5.1: The CCG mechanism as a stack

in step c, the semantic component pushes and pops from the stack the logical form *swimming*, following the steps explained in detail in 4.2.2. As we demonstrated there, β-conversion ends up with an empty stack, which means that our semantic component will leave the single CCG stack in its original condition, as it was before the start of the β-conversion process. In this way, the syntactic derivation can continue from the point in which has stopped, moving on to possible subsequent reductions.

This argument can be also extented for the case of the different environments collected for the skolem term objects, which might resemble a stack of bounded variables passed between the nodes of the syntactic tree. However, as we have already noted in Section 4.2.6, in our design the environment is always implicit in the nested object structure that is created for every logical form. This means that collecting the environment for an object with n levels of nesting brings a complexity of $O(n)$, that is, insignificant. What about the construction of logical forms then? This process indeed takes place with a recursive way, exhibiting stack-like behaviour. However, the construction of logical forms happens when again the main CCG stack is neutral, before of each reduction stage. So the system once more does not need to resort to a second stack.

The fact that the whole process can take place in the context of just one stack is very important, since it means that CCG keeps its nearly-context free complexity[3]; the addition of a second stack would in principle give to the mechanism the expressive

[3]See Section 2.2.2.

power of a universal Turing machine, possibly raising its theoretical power to exponential time.

Chapter 6

Discussion

In this dissertation we described in detail the construction of a wide-coverage parsing system capable of handling quantifier scope ambiguities with a novel way. In the present chapter we attempt to provide a summary of our work, critically evaluating our contributions and the extend to which we achieved our purpose. In Section 6.2, we also provide specific suggestions on possible ways with which this work might be further extended.

6.1 General remarks

Before of anything else, it would be helpful to recall our initial intentions. Our purpose was to provide a proof of concept that a semantic CCG parser equipped with generalized skolem terms can naturally and efficiently handle a wide range of quantifier scoping phenomena, which until now were covered in various (and, in many cases, unsatisfactory) degrees by many different ad-hoc and incompatible to each other approaches. This can be achieved in the context of the grammar itself, providing a linguistically appealing "global" solution to the problem. We believe that the results presented in Chapter 5 prove that we have accomplished this. More specifically, the contributions of this paper were the following:

- We provided one of the few implementations[1] of the generalized skolem term notion and showed how this can be placed in the context of a λ-calculus manipulation system.

[1] Actually, the only other "hands-on" work using generalized skolem terms we are aware of is that of Otake and Yoshimoto (2006) on quantified relative clauses in Japanese.

- We showed how this concept can be applied in the context of wide-coverage parsing, describing in details the integration process between a probabilistic parser and the semantic component.

- We provided a proof of concept for the applicability of the approach under practical conditions and the benefits it brings regarding the quantifier scoping problem compared with the current approaches, by presenting results that indeed conform to the theory.

A by-product of this work is the Java implementation of a λ-calculus system, the only one at this level of completeness in the open-source domain we are aware of by the time of this writing. This library is self-contained and designed in a way that favours extensibility, so it can be used in other contexts with minimal modifications[2].

The probabilistic parser we developed is able to parse 96.6% of unseen newspaper text and to achieve 92.4% accuracy on the assignment of lexical categories. Despite the less than optimal performance in PARSEVAL measures and head-word dependencies, we consider this part of the project successful. After all, and given that the parser assigns the right categories to the lexical entries at the beginning of the parsing process, it is mathematically certain that the application of the combinatory rules will lead to a derivation.

Although the syntactic component was really "wide-coverage", time did not permit us to achieve a similar result for the semantic part. Something like that would require the creation of a detailed lexicon of semantic forms able to cover any possible syntactic structure in the test corpus. This was very difficult to be achieved in the available time (in Section 6.2 we present a methodology of how this part can be fulfilled). We can certainly expect that the application of the semantic component to unrestricted text would create further requirements from our implementation, which for now remained unseen behind the carefully selected test set we used.

Despite this fact, our semantic evaluation covered a wide range of quantifier scope ambiguities, focusing on the most problematic and the less succesfully addressed by current approaches cases: spurious readings, scope assymetries, intermediate readings, distribution over conjunction and disjunction, generalized quantifiers. Most of the test cases we used are long-studied examples that are constantly turn up in linguistic papers related to quantifier scoping. In almost every case we got the right result – some minor problems were due to flaws of the implementation and can be easily fixed. This is

[2]The code of the project will be available in due time at `http://code.google.com/p/semccg/`.

a proof that our approach can indeed provide a global and a natural treatment to the problem, and that its application in a wide-coverage setting is also possible.

During our evaluation we met some problems, since in many cases the probabilistic model could not produce the right constituents that would allow the proper semantic manipulation. This problem is not specific to our parser, but constitutes a real weakness of every statistical approach: No matter how simple or common a constituent is, the model will disfavour it if it was not originally contained in the training data[3].

Interestingly, wide-coverage parsers have their way to overcome such problems, by stretching the syntactic flexibility provided by the grammar. They will eventually end up to some derivation, even if this is not the optimal or even not the right one. In the short discussion of this issue in Chapter 5, we mentioned an example that falls to the "not optimal" case. It is instructive to also see a "wrong" analysis. For the remarkably simple sentence "Every man walks and talks", both our tool and Clark and Curran's parser assigned the wrong category N to the verb *talks*[4]. The interesting point is how both tools eventually managed to provide an analysis, despite this fact (the C&C derivation is provided in 11a, while our derivation in 11b):

(11) a.

Every man	walks	and	talks
NP	$(S\backslash NP)/NP$	$conj$	N
		$\xrightarrow{>}$ N	
		$N \Rightarrow NP$	
	$\xrightarrow{>}$ $S\backslash NP$		
$\xrightarrow{<}$ S			

b.

Every man	walks	and	talks
NP	$S\backslash NP$	$(S\$_1\backslash_\star S\$_1)/_\star S\$_1$	N
$\xrightarrow{<}$ S			$N \Rightarrow NP \Rightarrow S$
		$\xrightarrow{>}$ $S\$_1\backslash_\star S\$_1$	
$\xrightarrow{<}$ S			

The fact that logical forms cannot follow this kind of "workarounds" constitutes a real problem for every wide-coverage semantic system, and might be a strong indication that we simply follow a wrong perspective: Instead of running a semantic component on top of a wide-coverage parser, perhaps we should work the other way around. After all, in our case syntactic analysis is just a means to derive correct logical forms, not an end by itself.

[3] Zipf's law says that this will be the case for the majority of text fragments.

[4] This should not be surprising: both tools use the same supertagger, although with a slightly different way – see Section 4.1.2 for details.

6.2 Future work

The work presented in this dissertation is by no means complete. For the syntactic component, further work is needed in order to achieve better performance according to the standard parsing measures. This work has to be concentrated on the fine-tuning of the probabilistic model, where extensive experimentation is required with every possible configuration between the various parameters: the supertagger's k and β parameters, the beam-search width, and the set of unary rules that the system should use. Furthermore, an error analysis we attempted showed that our approach for treating conjunction and punctuation has some differences with the way in which these structures were handled during the initial development of CCGbank, and are now represented in our "gold standard". Our parser had also difficulties in recognizing correctly many cases of numbers or financial symbols, as well as dashes, parentheses, and other delimiters. All these problems might have resulted some reduction in the performance. It seems then that a systematic examination of these small details and refining can bring some considerable improvements to our numbers.

The semantic component needs also considerable amount of work. First, in its current form does not incorporate polarities or negation, while the support of generalized guantifiers like "most" or "at least 3" and plural nouns is rather sketchy. Together with the extention of the semantic lexicon (which is described in the next section) we consider the addition of this functionality as the next step towards a really wide-coverage tool. This task is simplified by the fact that the appropriate design and the required data structures are already present. There is also a number of other issues that time did not permit us to properly address. The next sections provide a short introduction to the most important of them:

6.2.1 Extending the semantic lexicon

The extension of the semantic lexicon in order to provide logical forms for every case that can be encountered in a wide-coverage setting is an involved task, but we believe that our design simplifies it to a great extend. We propose a semi-automatic procedure similar to that of Bos et al. (2004): the semantics for each open-class word can be instantiated by using the lemma of the word – this in our system is achieved through the keyword `<word>` that can be included in a logical form, such us $\lambda x.\lambda y.$<word>(y,x) for transitive verbs. On the other hand, the logical forms for closed-class words (e.g. prepositions) can be assigned manually. The form of our lexicon helps again for this

purpose, since it allows arbitrary levels of granularity in the word groupings, and even supports the assignment of logical forms to individual words. More details about the semantic lexicon can be found in Section 4.2.7.

6.2.2 Unpacking of the results

The unpacking and enumeration of the available semantic readings is another issue not adequately addressed in this project due to time constraints. For the moment, the parser delivers packed forms of the kind described in Section 4.2.9 and used in Section 5.2 for the presentation of the evaluation. This output should eventually be unpacked, reinstating the predications that currently are represented as nominal properties of skolem terms, and enumerating the readings in a conventional form. For example, the output for the sentence "Everybody loves somebody" is the following:

(12) $\forall x[person(x) \rightarrow loves(x, sk_{person}^{\{\}\{x\}})]$

After the end of the derivation, this should be unpacked in the two prenex normal form conventional formulas:

(13) a. $\forall x[person(x) \rightarrow (person(sk_1()) \wedge loves(x, sk_1()))]$

b. $\forall x[person(x) \rightarrow (person(sk_1(x)) \wedge loves(x, sk_1(x)))]$

This will provide compatibility with the current conventions and will allow the use of the tool in many other contexts and applications. Although this specific part is trivial, the whole process of collecting all the available readings can be a little more involved, since, as we saw in Section 5.2, in some cases the parser needs access to all full parses that managed to reach the upper-right cell of the chart. To this respect, Algorithm 6.1 presents the steps that should be performed after the end of the parsing process. In the pseudo-code, P corresponds to a full parse and R is the set of prenex normal form formulas returned to the user. After the collection of the results, we have added an extra step of filtering. This can be done along the lines of the discussion in Section 5.2 for the sentence "Every man who reads a book loves a woman" (p. 66).

Finally, we should admit that the code of our semantic component at its current state is by no means optimal – in software engineering terms, we would consider it as an *alpha* version. We will continue working on this and we will constantly publish updates in the address given in footnote 2, page 76.

```
1: function EnumerateResults(table, words) returns R
2: R ← {}
3: for all {P|P ∈ table[0, Length(words)]} do
4:     Q ←Enumerate(P)
5:     Q ←Normalize(Q)
6:     R ← R ∪ Q
7: end for
8: R =Filter(R)
```

Algorithm 6.1: An enumeration algorithm

Bibliography

Ades, A. and Steedman, M. (1982). On the order of words. *Linguistics and Philosophy*, 4:517–558.

Ajdukiewicz, K. (1935). Die syntaktische konnexität. In McCall, S., editor, *Polish Logic 1920-1939*, pages 207–231. Oxford University Press, Oxford. Translated from *Studia Philosophica*, 1, 1-27.

Bach, E. (1976). An extension of classical transformational grammar. In *Problems in Linguistic Metatheory: Proceedings of the 1976 Conference at Michigan State University*, pages 183–224, Lansing. Michigan State University.

Baldridge, J. (2002). *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh.

Bangalore, S. and Joshi, A. K. (1999). Supertagging: An approach to alsmost parsing. *Computational Linguistics*, 25(2):237–265.

Bar-Hillel, Y. (1953). A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58.

Black, E., Abney, S. P., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J. L., Liberman, M., Marcus, M. P., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of english grammars. In *HLT*. Morgan Kaufmann.

Blackburn, P. and Bos, J. (2005). *Representation and Inference for Natural Language: A First Course in Computational Semantics*. CSLI Publications, Stanford, California.

Blackburn, P. and Marx, M. (2002). Tableaux for quantified hybrid logic. In Egly, U. and Fermüller, C. G., editors, *Automated Reasoning with Analytic Tableaux and*

Related Methods, volume 2381 of *Lecture Notes in Computer Science*, pages 38–52. Springer-Verlag.

Bos, J. (1995). Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143, University of Amsterdam.

Bos, J., Clark, S., Steedman, M., Curran, J., and Hockenmaier, J. (2004). Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 1240–1246, Geneva.

Clark, S. and Curran, J. R. (2004a). The importance of supertagging for wide-coverage ccg parsing. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 282–288, Geneva.

Clark, S. and Curran, J. R. (2004b). Parsing the WSJ using CCG and log-linear models. In *ACL*, pages 103–110.

Clark, S. and Curran, J. R. (2007). Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Clark, S., Hockenmaier, J., and Steedman, M. (2002). Building deep dependency structures using a wide-coverage CCG parser. In *ACL*, pages 327–334.

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania.

Cooper, R. (1983). *Quantification and Syntactic Theory*. D. Reidel, Dordrecht, The Netherlands.

Cooper, R., Crouch, D., Eijck, J. V., Fox, C., Genabith, J. V., Jaspars, J., Kamp, H., Milward, D., Pinkal, M., Poesio, M., Pulman, S., Briscoe, T., Maier, H., and Konrad, K. (1996). Using the framework.

Copestake, A. and Flickinger, D. (2000). An open-source grammar development environment and broad-coverage English grammar using hpsg. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC-2000*. ACL.

Curry, H. B. and Feys, R. (1958). *Combinatory Logic: Vol. I.* North Holland, Amsterdam.

Eisner, J. (1996). Efficient normal-form parsing for combinatory categorial grammar. In Joshi, A. and Palmer, M., editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 79–86, San Francisco. Association for Computational Linguistics, Morgan Kaufmann Publishers.

Farkas, D. (2001). Dependent indefinites and direct scope. In Condoravdi, C. and de Lavalette, R., editors, *Logical Perspectives on Language and Information*, pages 41–72. CSLI Publications, Stanford, CA.

Gabsdil, M. and Striegnitz, K. (2000). Classifying scope ambiguities. *Journal of Language and Computation*, 1(2):291–297.

Geach, P. T. (1972). A program for syntax. In Davidson, D. and Harman, G. H., editors, *Semantics of Natural Language*, pages 483–497. D. Reidel Publishing Co., Dordrecht.

Hobbs, J. and Shieber, S. (1987). An algorithm for generating quantifier scopings. *Computational Linguistics*, 13:47–63.

Hockenmaier, J. (2001). Statistical parsing for CCG with simple generative models. In *ACL (Companion Volume)*, pages 7–12.

Hockenmaier, J. (2003). *Data and models for statistical parsing with Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh.

Hockenmaier, J. and Steedman, M. (2002). Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, pages 1974–1981, Las Palmas, Spain.

Jannsen, T. (1997). Compositionality. In Van Benthem, J. and Ter Meulen, A., editors, *Handbook of Logic and Language*, chapter 7, pages 417–473. Elsevier.

Joshi, A., Levy, L., and Takahashi, M. (1975). Tree-adjunct grammars. *Journal of Computer Systems Science*, 10:136–163.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, Englewood Cliffs, New Jersey.

Keller, W. R. (1988). Nested cooper storage: the proper treatment of quantification in ordinary noun phrases. In Reyle, U. and Rohrer, C., editors, *Natural Language Parsing and Linguistic Theories*, pages 432–447. D. Reidel, Dordrecht.

Kempson, R. M. and Cormak, A. (1981). Ambiguity and quantification. *Linguistics and Philosophy*, 4(2):259–309.

Koller, A., Niehren, J., and Treinen, R. (1998). Dominance constraints: Algorithms and complexity. In *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics (LACL '98)*, Grenoble, France. To appear in LNCS.

Koller, A. and Thater, S. (2006). An improved redundancy elimination algorithm for underspecified representations. In *ACL*. The Association for Computer Linguistics.

Montague, R. (1970a). English as a formal language. In *Linguaggi nella Società e nella Tecnica*, pages 189–224. Edizioni di Comunità, Milan.

Montague, R. (1970b). Universal grammar. *Theoria*, 36:373–398.

Montague, R. (1973). The proper treatment of quantification in ordinary english. In e.a., J. H., editor, *Approaches to Natural Language*, pages 221–242. Reidel.

Mostowski, A. (1957). On a generalization of quantifiers. *Fundamenta Mathematica*, 44:12–36.

Otake, R. and Yoshimoto, K. (2006). Multiply quantified internally headed relative clause in japanese: A skolem term based approach. In *19th Asia-Pacific Conference on Language, Information and Computation*.

Pereira, F. and Shieber, S. (1987). *Prolog and Natural Language Analysis*. CSLI Publications, Stanford, CA.

Roark, B. (2001). Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27:249–276.

Ross, J. R. (1967). *Constraints on Variables in Syntax*. PhD thesis, MIT. Published as *Infinite Syntax!*, Ablex, Norton, NJ, 1986.

Steedman, M. (1999). Quantifier scope alternation in CCG. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, College Park, MD*, pages 301–308, San Francisco, CA. Morgan Kaufmann.

Steedman, M. (2000). *The syntactic process*. MIT Press, Cambridge, Massachusetts.

Steedman, M. (2010). *The Natural Semantics of Scope*. Currently in publication by MIT Press.

Vijay-Shanker, K. and Weir, D. J. (1994). The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27(6):511–546.

Willis, A. and Manandhar, S. (1999). Two accounts of scope availability and semantic underspecification. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, College Park MD, June*, pages 293–300, San Francisco, CA. Morgan Kaufmann.

Wittenburg, K. B. (1986). *Natural Language Parsing with Combinatory Categorial Grammar in a Graph-Unification Based Formalism*. PhD thesis, University of Texas, Austin.