Probabilistic Grammar Induction from Sentences and Structured Meanings

Tom Kwiatkowski

Doctor of Philosophy
Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh
2011

Abstract

The meanings of natural language sentences may be represented as compositional logical-forms. Each word or lexicalised multiword-element has an associated logical-form representing its meaning. Full sentential logical-forms are then composed from these word logical-forms via a syntactic parse of the sentence.

This thesis develops two computational systems that learn both the word-meanings and parsing model required to map sentences onto logical-forms from an example corpus of (sentence, logical-form) pairs. One of these systems is designed to provide a general purpose method of inducing semantic parsers for multiple languages and logical meaning representations. Semantic parsers map sentences onto logical representations of their meanings and may form an important part of any computational task that needs to interpret the meanings of sentences. The other system is designed to model the way in which a child learns the semantics and syntax of their first language. Here, logical-forms are used to represent the potentially ambiguous context in which child-directed utterances are spoken and a psycholinguistically plausible training algorithm learns a probabilistic grammar that describes the target language. This computational modelling task is important as it can provide evidence for or against competing theories of how children learn their first language.

Both of the systems presented here are based upon two working hypotheses. First, that the correct parse of any sentence in any language is contained in a set of possible parses defined in terms of the sentence itself, the sentence's logical-form and a small set of combinatory rule schemata. The second working hypothesis is that, given a corpus of (sentence, logical-form) pairs that each support a large number of possible parses according to the schemata mentioned above, it is possible to learn a probabilistic parsing model that accurately describes the target language.

The algorithm for semantic parser induction learns Combinatory Categorial Grammar (CCG) lexicons and discriminative probabilistic parsing models from corpora of (sentence, logical-form) pairs. This system is shown to achieve at or near state of the art performance across multiple languages, logical meaning representations and domains. As the approach is not tied to any single natural or logical language, this system represents an important step towards widely applicable black-box methods for semantic parser induction. This thesis also develops an efficient representation of the CCG lexicon that separately stores language specific syntactic regularities and domain specific semantic knowledge. This factorised lexical representation improves the performance of CCG based semantic parsers in sparse domains and also provides a potential basis for lexical expansion and domain adaptation for semantic parsers.

The algorithm for modelling child language acquisition learns a generative probabilistic model of CCG parses from sentences paired with a context set of potential logical-forms containing one correct entry and a number of distractors. The online learning algorithm used is intended to be psycholinguistically plausible and to assume as little information specific to the task of language learning as is possible. It is shown that this algorithm learns an accurate parsing model despite making very few initial assumptions. It is also shown that the manner in which both word-meanings and syntactic rules are learnt is in accordance with observations of both of these learning tasks in children, supporting a theory of language acquisition that builds upon the two working hypotheses stated above.

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.

(*Tom Kwiatkowski*)

Acknowledgements

This thesis would not exist without the input of Mark Steedman, Sharon Goldwater and Luke Zettlemoyer. Meetings with Mark have been an adventure that shaped my thinking both within and far beyond the scope of this thesis. Sharon has been an invaluable guide in focusing and grounding the project and has always been ready with a new direction when needed. Luke has been a great collaborator from whom I have learnt a great about the nature of the problems that we are trying to solve and the best way to go about solving them.

This thesis has benefited from useful and interesting discussions with far too many people to even start to enumerate but I owe thanks, in particular, to Frank Keller and Trevor Cohn for their detailed feedback on my early attempts at this work.

Informatics in Edinburgh has been a great place to work and spend time and, again, I cannot thank everyone responsible but special thanks should go to Emily T, Kira, Mark W, Arm, Christos, Aciel, Moreno, Stella, Sasa and Dave for making it a nice place to be.

Outside of work and Informatics, thanks to Emma, Ian and Frances for being great flatmates and friends. Thanks to Emily W for all the food (and wine). Thanks to Erin for all the listening. Thanks to all other friends for keeping me sane and sociable—often forcibly. Thanks to Mum, Dad and the gang for everything.

Table of Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is concerned with learning probabilistic grammars that can parse sentences onto compositional logical representations of their meanings. These *logical-forms* are composed from the logical-forms of the words in the sentence via a *syntactic parse*. The problem of learning how to parse sentences onto their logical-forms can be broken down into learning a lexicon that maps words onto logical representations of their meanings, and learning a parsing model that describes how these should be combined to generate sentential logical-forms. These are both tasks have been performed successfully by all linguistically competent humans for at least one language and the computational exploration of how this is done is crucial to our understanding of how human language works. A reliable methodology for learning parsers that parse word-strings onto logical-forms is also necessary if we wish to build computational systems that can interpret the compositional meanings of sentences.

This thesis develops systems to address both of the separate but related tasks motivated above: the task of *modelling language acquisition*, and the task of *semantic parser induction*[1]. This thesis states that the lexicons and probabilistic parsing models required by each of these two tasks can be learnt from a set of example (sentence, logical-form) pairs via a language independent combinatorial function that defines the set of possible parses consistent with each of these pairs. By defining this combinatorial function once and in terms of a few general combinatory schemata, the approaches presented here can work across all natural languages and multiple logical meaning representations. This is important when modelling language acquisition as we do not

[1]In the context of this thesis, semantic parsers are parsers that map sentences onto compositional logical expressions which are interpretable by computer. The term semantic parser may be somewhat confusing as the parser in question does not parse semantics. Rather it parses sentences onto representations of their semantics.

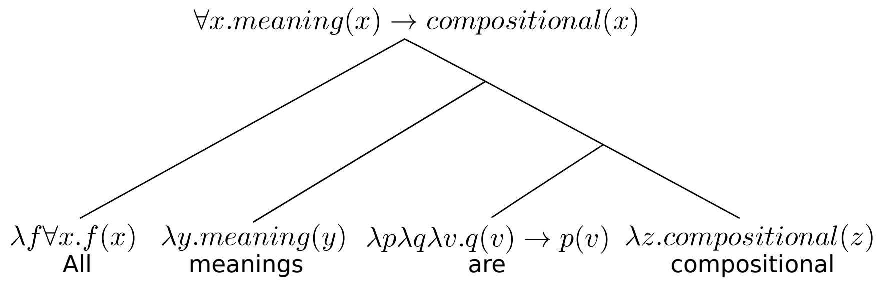
wish to make any assumptions about the language being learnt, nor do we wish to make excessive assumptions about the manner in which the semantics should be psychologically correctly represented. Independence from language and logical representation is also important if we wish to induce multiple different semantic parsers for different domains in which both the language and target meaning representation may differ.

Below I set out the key assumptions and working hypotheses behind the grammar induction systems presented. Then I describe and motivate in greater detail the two tasks, of semantic parser induction and of modelling language acquisition, that this thesis addresses. While related, these two tasks require different approaches. When inducing semantic parsers, the accuracy and efficiency of the resulting parser are paramount and the semantic parser induction algorithms are designed accordingly. When modelling language acquisition, it is important that the grammar learnt does represent the target language accurately but, equally importantly, the manner in which this grammar is learnt must be *psycholinguistically plausible*. Subsequently, the model of language acquisition calls for a different learning approach than semantic parser induction.

Having introduced the key novel ideas behind the work presented here and the two tasks that are addressed, I present a detailed overview of the rest of this document.

1.1 Underlying Assumptions and Hypotheses

I shall assume that the literal meanings of natural language sentences can be represented with compositional logical forms. For example, the meaning of the sentence 'All meanings are compositional.' can be represented logically with the compositional logical-form $\forall x.meaning(x) \rightarrow compositional(x)$. This logical-form is retrieved from the words in the surface string via a parse that first maps the words onto the logical-forms representing their individual meanings and then composes these into more complex logical-forms representing the meanings of phrases. An example parse is:


$\forall x.meaning(x) \rightarrow compositional(x)$
$\lambda f \forall x.f(x)$
All
$\lambda y.meaning(y)$
meanings
$\lambda p \lambda q \lambda v.q(v) \rightarrow p(v)$
are
$\lambda z.compositional(z)$
compositional

Each of the words has been mapped onto a word logical-form. The full sentence logical-form is built from word logical-forms via an intermediary *parse tree* that defines the functors and arguments in the process of composing word and phrasal logical-forms. In our example the order of composition is:

$$\lambda f \forall x.f(x) \ \ (\ \lambda p \lambda q \lambda v.q(v) \rightarrow p(v) \ \ (\ \lambda z.compositional(z) \ \ \lambda y.meaning(y) \ ) \ ) \, .$$

The aim of both of the systems presented in this thesis is to learn a parser $\mathcal{P}$ that maps sentences s onto logical representations of their meanings m via a parse t of the type illustrated above.

$$m \ = \ \mathcal{P}(s) \tag{1.1}$$

My hypothesis is that $\mathcal{P}$ can be represented as a probabilistic grammar and that this probabilistic grammar can be learnt from an example set of (sentence, logical-form) pairs $D = \{(s_i, m_i)|i = 1, \ldots, N\}$ even though the correct parse t_i for each of these pairs is unknown. In the work to follow, $\mathcal{P}$ will have access to a language specific lexicon; a language independent set of combinatory rules; and a probabilistic model that chooses between parses. In this thesis, I propose that both the language specific lexicon and probabilistic parsing model can be learnt from D via a mapping $\mathcal{T}$ from (sentence, logical-form) pairs to all combinatorially possible parses $\{t\}$ that could be used to map the words of each sentence onto its logical form[2].

$$\{t\}_i \ = \ \mathcal{T}(s_i, m_i) \tag{1.2}$$

The lexicon can be learnt by accumulating the lexical items used at the leaves of each parse t. The parsing model can be learnt by aggregating statistics over the set $\{t\}_i$ for $i = 1, \ldots, N$.

This thesis states that the function $\mathcal{T}$ can be defined once for all natural languages in terms of a small set of combinatory rules and a simple mapping from semantic type to possible syntactic categories in all languages. Furthermore, this thesis states that the very general syntactic information encoded in the combinators of $\mathcal{T}$ and the semantic typing system provides enough structure for $\mathcal{P}$ to be learnt from a set of sentences paired with their logical-forms via general statistical techniques— obviating the need for more specific syntactic specifications or acquisitive operations.

[2]All parses possible according to a small set of CCG combinators that cover the syntax of all natural languages.

1.2 Semantic Parser Induction

Semantic parsers build computationally interpretable logical-forms from natural language sentences. Recent work has focused on learning such parsers directly from corpora made up of sentences paired with logical meaning representations (Kate et al., 2005; Kate and Mooney, 2006; Wong and Mooney, 2006, 2007; Zettlemoyer and Collins, 2005, 2007; Lu et al., 2008). However, these systems have tended to use grammars that have been specifically tailored to either the natural language of interest or the formal language in which the meanings are represented.

In this thesis I present a system for learning semantic parsers that works across a range of formal and natural languages. The motivation for language independence is obvious - there are many languages that we wish to model. The motivation for meaning representation independence is less obvious. While there is a general consensus between linguists that the correct representation of meaning should be universal there is no consensus on what this universal representation looks like. Consequently most of the meaning representations on offer are ad-hoc and tailored to the application of interest. A general purpose semantic parsing framework that works across all common types of compositional logical meaning representation is desirable.

The semantic parsing framework that I present can be adapted to any new natural language or compositional logical representation of meaning simply training it on a set of example (sentence, logical-form) pairs of the correct type. I show that, despite being motivated by the need for a generally applicable system, this approach is capable of learning a state of the art semantic parser for multiple natural languages and logical meaning representations in multiple semantic parsing domains.

1.3 Modelling Language Acquisition

The “cognitive theory of language acquisition” (Pinker, 1979) assumes that children learn the syntax and semantics of a language through mapping the words heard in an utterance onto a contextually afforded representation of the utterance’s meaning. The process itself is unclear but widely theorised about. We can test a theory of how the child learns language by implementing it computationally, training it on data that represents the learning input available to the child, and comparing its progress in acquiring linguistic knowledge to that of the child. Siskind (1992); Villavicencio (2002) and Buttery (2006) have presented computational accounts of a child’s acquisition of syntax and semantics from child-directed utterances paired with compositional logical representations of their meaning. The input available to these approaches is in accordance with the cognitive theory of language acquisition stated above. The process by which they acquire linguistic knowledge requires predefinition of a special linguistic faculty. Predefined parameters are assumed by these authors to control specific linguistic alternations. This requires the full set of linguistic alternations to be listed prior to training.

In this thesis, I present a computational model of syntactic and semantic acquisition that does not require the enumeration and parameterisation of all linguistic alternatives. Rather, it uses the general function $\mathcal{T}$ to propose all possible parses of an utterance in a given context and learns a probabilistic grammar from a set of (utterance, context) pairs, where the context may support more than one candidate meaning. The design and training of this model is motivated by the a need for *psycholinguistic plausibility* — it does not have access to any language-specific information prior to training and the training algorithm is strictly online.

I show that it is possible to learn an accurate parsing model from an input that approximates the input available to a child learning her first language. I also show that this parsing model can learn word meanings on the basis of a single exposure, similar to the *fast mapping* phenomenon observed in children (Carey and Bartlett, 1978). I also show that the learner captures the step-like learning curves for word order regularities that Crain and Thornton (1998) and Thornton and Tesan (2007) claim children show. This result counters one of Thornton and Tesan's principle criticisms of statistical grammar learners—that they tend to exhibit gradual learning curves rather than the abrupt changes in linguistic competence observed in children.

1.4 Overview of Document

The rest of this document proceeds as follows. **Chapter 2** reviews formal representations used for compositional meanings of sentences, along with Combinatory Categorial Grammar (CCG), the grammatical framework used to model natural language syntax.

Chapter 3 then defines the function $\mathcal{T}$ from Equation 1.2. This function forms the basis for the learning algorithms in Chapters 4 and 5 (for semantic parsing) and Chapter 6 (for modelling language acquisition).

Chapter 4 presents a system that is capable of learning a state of the art semantic parser from corpora of (sentence, logical-form) pairs. This system can learn parsers for multiple languages and multiple meaning representations making it the most general semantic parser induction framework to date. In order to maintain scalability, the system presented in Chapter 4 does not explore the full set of parses proposed by $\mathcal{T}$ for the entire training corpus. Rather it searches the space of possible parses incrementally guided by its probabilistic parsing model. This work has been published as Kwiatkowski et al. (2010).

Despite its generality, the semantic parsing framework presented in Chapter 4 suffers from data sparsity in more complex domains. The work presented in **Chapter 5** combats this problem by introducing a factored representation of the CCG's lexicon. This allows the system to learn separate syntactic and semantic abstractions yielding a lexical representation that is more compact than that used in Chapter 5 but which also has greater coverage. This work has been published as Kwiatkowski et al. (2011).

Chapter 6 presents a model of child language acquisition from child directed utterances paired with logical representations of several possible meanings. These logical representations are used as a proxy for the inferences that the child makes about the utterance's potential meanings on the basis of the context in which the utterance is heard. In practice, it is likely that the child will be unable to identify which of a number of possible meanings is the one intended by the speaker early in learning. This *propositional uncertainty* is modelled in by pairing each utterance with a *context set* of potential logical-forms, only one of which represents the speaker's intended meaning. The system presented in Chapter 6 is designed to be language independent and psycholinguistically plausible, trained with a single online pass over the data. I show that this system is capable of learning a grammar and parser that models the language to which the child has been exposed. I also show that the manner in which both the word-meanings and word-order rules are learnt correlate with some observations of the progress of language acquisition in children. This work has been submitted for publication as Kwiatkowski et al. (2012).

Chapter 2

Representing the Semantics and Syntax of Natural Language

Human language is *productive*. We all have the ability to generate and interpret an unbounded set of novel sentences with novel meanings. Therefore any reasonable model of natural language semantics and syntax must also have this capability. It was this observation that led Frege (1892) to formulate the *principle of compositionality* [1]:

> **The Principle of Compositionality :** The meaning of a whole is a function of the meanings of its parts and their mode of syntactic composition (paraphrase drawn from Portner and Partee 2002) .

With the principle of compositionality, it is possible to define an infinitely productive model of natural language semantics from a finite set of atomic semantic expressions. The meaning of a sentence is built from its components by use of a lexical mapping from atomic morphological units onto their meanings and a syntactic parse that combines these meanings into a single compositional structure. This thesis is concerned with the development of systems that learn grammars containing both the lexical mappings and syntactic parse rules required to parse sentences onto their logical-forms. Before going on to describe how these grammars are learnt, I first introduce the formal representations of semantics and syntax used in later chapters.

Section 2.1 reviews predicate logic and the simply typed lambda-calculus. Together these are used to represent the meanings of words, phrases and sentences. The lambda-calculus is used both in the logical representation, to define functional logical-

[1]The key idea behind the principle of compositionality almost certainly predates Frege, yet it is he who generally credited with its current prevalence in linguistic semantics.

forms, and as a ‘glue language’ allowing logical-forms to be composed in a parse. Section 2.2 then reviews the Combinatory Categorial Grammar (CCG) used to build parses of a sentence. CCG explicitly models the concurrent operations of syntactic combination and semantic composition in the parse of a sentence. This is particularly important for the approach taken in this thesis. Subsequently, I focus in particular on the close relationship between the combinators of CCG and the *function application* and *function composition* operations of the lambda-calculus.

2.1 Representing Natural Language Meaning

Truth conditional theories of semantics, first proposed by Tarski (1933), state that the meaning of a sentence can be given by stating the conditions that must be satisfied for that sentence to be true. This notion of meaning paves the way for the application of formal logical languages to the task of describing natural language semantics. The model theoretic account of natural language semantics, first developed into a grammatical framework in Montague (1970) and applied to a subset of English in Montague (1973) (both reviewed in Dowty et al. 1980), provides a framework within which a truth-conditional semantics can be built. For this reason, model theoretic views of semantics are of great use in computational linguistics as they allow a system designer to write out well formed representations of a sentence’s meaning. All of the work in this thesis represents sentence meanings with compositional, symbolic, model-theoretic logical-forms for which a model theory is known.

2.1.1 Predicate Logic

The grammar of Montague (1973) is based upon higher-order predicate logic with the lambda-calculus. While it is true in some sense that natural language is capable of quantifying over higher-order terms, logical inference in higher-order logics is difficult. This led Blackburn and Bos (2003, 2005) to describe first-order predicate logic as “an attractive compromise between the conflicting demands of expressivity and inferential effectiveness”. All of the experiments presented in Chapters 4, 5 and 6 use first-order predicate logic in the representation of natural language meaning. However, all of the systems presented are also agnostic to the order of the logic used[2]. As these

[2]And as we shall see, all use the higher-order lambda calculus for sub-sentential meanings and questions.

systems do not, at any point, perform logical inference with the logical-forms they are blind to the problems of higher-order logics mentioned above.

First-order predicate logic builds logical expressions from a fixed set of *logical constants*, a fixed set of *non-logical constants* and a countably infinite collection of variables. The logical constants of a language can be separated into *boolean connectives* $\{\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $=$ (equality)$\}$, and *quantifiers* $\{\forall$ (universal quantifier), $\exists$ (existential quantifier)$\}$. The non-logical constants of the language are *entity identifiers*, *predicates* that describe relations and *functions* that describe mappings from one or more elements of a set to an element of another set.

First order predicate logic expressions can be *terms* that describe entities in the world or *formulae* that describe relationships between terms. Terms $\mathtt{t}$ are entity identifiers, variables v or functions $\mathtt{F}_n$ of arity n applied to n terms $\mathtt{F}_n(\mathtt{t}_1, \ldots, \mathtt{t}_n)$. Formulae $\mathtt{f}$ are built from terms, predicate symbols $\mathtt{P}_n$ of arity n, other formulae, variables and the logical connectives above. Formulae take of one of the following forms:

$$\{\mathtt{f}\} \;=\; \mathtt{P}_n(\mathtt{t}_1, \ldots, \mathtt{t}_n),\;\; \mathtt{t}_1 = \mathtt{t}_2,\;\; \neg\mathtt{f},\;\; \mathtt{f}_1 \wedge \mathtt{f}_2,\;\; \mathtt{f}_1 \vee \mathtt{f}_2,\;\; \forall x.\mathtt{f},\;\; \exists x.\mathtt{f}$$

As an example, the binary predicate $love$ is used in a well formed formula to describe a relationship between the terms $mary$ and $john$.

$$\text{Mary loves John.}$$
$$loves(mary, john)$$

Variables are terms which range over sets of entities. Quantifiers *bind* variables and take scope over formulae. Quantified formulae specify a domain-of-discourse that satisfies this formula. The following logical formula makes use of the existential quantifier $\exists$, a variable x and a conjunction $\wedge$.

$$\text{Some woman loves John.}$$
$$\exists x.woman(x) \wedge loves(x, john)$$

2.1.2 Davidsonian Events

Davidson (1967) expands the truth conditional theory of semantics with the theory of *event semantics* that uses events to reference actions in the world. In Davidson's initial work on event semantics, an *event argument* is added to each predicate representing an action verb. Adverbial modifiers may then be treated as predications over events. For example, in the following example the predicate *passionate* applies to the event argument of the main verb.

$$\text{Mary loves John passionately}$$
$$\exists ev.loves(mary, john, ev) \wedge passionate(ev)$$

Using predication over events to conjoin action verb phrases with their modifiers avoids the duplication of semantic content that occurs when new verbal predicates of ever greater arity are invented to cope with a (theoretically unbounded) set of adverbial modifiers.

2.1.3 The Lambda-Calculus

The lambda-calculus introduced by Church (1936) uses λ abstractions to define functions. The lambda-term $\lambda x.t$ represents a function in which the variable x ranges over all possible input expressions. When applied to a single input expression a, the function above returns the expression $t[a/x]$ in which all occurrences of x in t have been replaced with a. The lambda-calculus is useful in modelling natural language semantics for two reasons. Firstly, it can be used to define functional meanings of sentences and phrases that could not be expressed with predicate logic alone. Secondly, it can be used as a 'glue language' to combine logical expressions that represent the meanings of phrases.

Functional Meanings

The predicate logic described above in Section 2.1.1 can only be used to describe declarative relationships between entities. Many natural language sentences do not make such statements but rather query the state of the world. By augmenting predicate logic with the lambda-calculus it is possible to express functional meanings of sentences. These functions can then be applied over the model of the world in order to ascertain facts about the world state. The logical expression in the following example could be applied over the current world model to find the set of terms $\{a\}$ for which the proposition $loves(a, john)$ is true.

$$\text{Who loves John}$$
$$\lambda x.loves(x, john)$$

In other words, answering the question 'Who loves John'. This type of construction is particularly common in representations of sentence meaning for database queries as we shall see later.

Combining Phrasal Meanings

As well as increasing the expressive power of predicate logic, the lambda-calculus can be used as a glue language to combine the meanings of natural language phrases. Lambda-calculus expressions may be combined via *function application* or *function composition.*

Function Application applies a function $\lambda x.f(x)$ to an argument a.

$$\lambda x.f(x) \quad a \;\rightarrow\; f(a) \tag{2.1}$$

Function Composition applies a primary functor $\lambda x.f(x)$ to the results of a secondary functor $\lambda y.g(y)$. The resulting expression is a function with the same range as the secondary functor.

$$\lambda x.f(x) \quad \lambda y.g(y) \;\rightarrow\; \lambda y.f(g(y)) \tag{2.2}$$

2.1.4 The Simply Typed Lambda-Calculus

The standard lambda-calculus is type-free and allows the application of every possible function to every possible expression. The *typed lambda-calculus* introduced by Church (1940) uses a set of types to restrict the range of expressions that each function may be applied to. The simply typed lambda-calculus has a set of atomic *base types* **B** and a single type constructor $\rightarrow$ used to build function types. The base types used by Montague (1973) are *entity* e and *truth value* t. The set of types **T** is defined recursively as:

$$\mathbf{T} = \{\mathbf{B}\} \cup \{T_1 \rightarrow T_2 \mid T_1 \in \mathbf{T} \wedge T_2 \in \mathbf{T}\}$$

In Montague's grammar, logical terms are given the type e and formulae have type t. For example, the constant $john$ and variable x in the expression $\lambda x.loves(x, john)$ are assigned type e. The expression itself then has the functional type $e \rightarrow t$ which tells us that this function may only be applied to arguments of type e and that when this is done it will return a formula of type t. In the interests of a compact notation we shall, from now on, represent the functional type $T_1 \rightarrow T_2$ as $\langle T_1, T_2 \rangle$. It should be noted that in the first order predicate logic with the lambda-calculus, unlike first order predicate logic alone, variables bound by lambda-terms may have non-base types.

The choice of base types used in the simply typed lambda-calculus is open to interpretation. In later chapters, the base types e and t will be supplemented with ev (for Davidsonian events) and i (for natural numbers). However, the rich set of classifiers possible in natural language suggests that a much richer set of base types exists, they are just not used here. Also the typing system may be enriched with a domain specific *type hierarchy* can be used to rule out logical-forms that make no sense within the domain specific logical-language. We shall see an example of this in Chapter 4.

2.2 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG) (Steedman, 2000) is a strongly lexicalised grammatical formalism in which lexical items are triples of the form:

$$\text{word} \vdash \mathsf{syntactic\ category} : logical\ form$$

Examples of which are $\text{John} \vdash \mathsf{NP} : john$ which pairs the word 'John' with the logical-form entity $john$ and the noun-phrase syntactic category NP; and $\text{loves} \vdash \mathsf{S\backslash NP/NP} : \lambda x \lambda y.loves(y, x)$ which pairs the word 'loves' with the binary predicate logical-form representing its semantics and the CCG syntactic category for transitive verbs in SVO ordered languages.

One of the greatest motivations behind CCG is the need for a transparent interface between syntax and semantics. Each CCG category $\mathsf{X} : h$ has a syntactic category X and a semantic component h. CCG relates syntactic category to semantic type via the *Principle of Categorial Type Transparency*.

The principle of categorial type transparency: "For a given language, the semantic type of the interpretation together with a number of language-specific directional parameter settings uniquely determines the syntactic category of a category" (Steedman, 2000, pp.36).

The allowable values of the syntactic category X are defined by the type of the semantics h. CCG also relates syntactic and semantic combination via the *Principle of Combinatory Type Transparency*.

The principle of combinatory type transparency: "All syntactic combinatory rules are type-transparent versions of one of a small number of simple semantic operations over functions" (Steedman, 2000, pp.37).

CCG syntactic categories use the slash operators / and \ to define *syntactic functions* in much the same way that λ operators are used to define *semantic functions* in the lambda-calculus. The CCG category $\mathsf{X/Y} : \lambda y.h$ is a function looking for a single syntactic argument matching Y and a single semantic argument matching y. A CCG functor may be combined with an argument category through the use of one of the CCG combinatory rules. These, described later in Section 2.2.2, are the syntactic analogues of the function application and function composition combinators of the lambda-calculus.

2.2.1 CCG Syntactic Categories

CCG syntactic categories may be *basic* or *complex*. The full set of syntactic categories C available to a grammar contains all basic and complex categories. Basic CCG categories can only ever be syntactic arguments in a parse. The set of atomic categories is defined on a ad-hoc basis. A typical set would be $\{\mathsf{S}, \mathsf{NP}, \mathsf{N}, \mathsf{PP}, \ldots\} \in C$. Complex CCG categories contain CCG slash operators (/ and \ ) that define the syntactic categories as functors. Each complex CCG category is made up from a pair of CCG categories and a CCG slash operator. If $\mathsf{X}, \mathsf{Y} \in C$ then $\mathsf{X/Y} \in C$ and $\mathsf{X\backslash Y} \in C$. The direction of the slash used in a complex syntactic category describes the direction in which that category is looking to combine with a constituent of the correct type. The forward slash $/\mathsf{Y}$ signals that the category is looking to combine with a constituent of the type Y on its right. The backward slash $\backslash\mathsf{Y}$ signals that the category is looking to combine with a constituent of the type Y on its left. An example CCG complex syntactic category is the transitive verb category used in the lexical item for ‘loves’:

$$\text{loves} \vdash (\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} : \lambda x \lambda y.loves(y,x).$$

The slash operators in this syntactic category tell us that it is looking to combine first with the basic NP syntactic category on its right and then another NP on its left.

The order in which a syntactic category combines with its syntactic arguments is the same as the order in which the associated logical-expression combines with its semantic arguments. For example, the following CCG parse builds the logical expression $loves(john, mary)$ for the sentence ‘John loves Mary’ using the lexical entry given above.

$$\begin{array}{ccc}
\text{John} & \text{loves} & \text{Mary} \\
\hline
\mathsf{NP}: \ john & (\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} : \lambda x \lambda y.loves(y,x) & \mathsf{NP}: \ mary \\
\cline{2-3}
 & \multicolumn{2}{c}{\mathsf{S}\backslash\mathsf{NP} : \lambda y.loves(y, mary) \quad >} \\
\hline
\multicolumn{3}{c}{\mathsf{S} : loves(john, mary) \quad <}
\end{array}$$

This parse uses the CCG *forward application* rule once to combine the transitive verb category $(\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP}$ with the NP on its right, and the *backward application* rule to combine the intransitive verb category $\mathsf{S}\backslash\mathsf{NP}$ with the NP on its left. The semantic application of the logical expression $\lambda x \lambda y.loves(y,x)$ to first $mary$ and then $john$ is implicit in the CCG combinatory rules used to perform syntactic combination. These are introduced next.

2.2.2 CCG Combinatory Rules

CCG uses a small set of combinatory rule schemata to describe the ways in which CCG categories are allowed to combine in a parse. Each of the combinatory rules is defined purely in terms of the CCG slash operators and a type equivalence between CCG syntactic categories in the functor and argument. The simplest of the rules used by CCG are the rules of *function application* (introduced by the categorial grammars of Ajdukiewicz (1935) and Bar-Hillel (1953)). The function application rules can be seen as the syntactic extension of the function application operation of the lambda-calculus. The function application rules are illustrated below by use of a rule schemata in which variables X and Y stand in for actual categories of the grammar.

1. Forward Application
   $\mathsf{X/Y} : f \quad \mathsf{Y} : a \quad \Rightarrow \quad \mathsf{X} : f(a)$

2. Backward Application
   $\mathsf{Y} : a \quad \mathsf{X\backslash Y} : f \quad \Rightarrow \quad \mathsf{X} : f(a)$

As well as having a syntactic analogue to function application in the lambda-calculus, CCG has a similar analogue to function composition. The following CCG rules can combine a pair of complex categories provided that the range of the primary functor matches the domain of the secondary functor and the outermost slash operators of both match the direction of composition.

3. Forward Composition
   $\mathsf{X/Y} : f \quad \mathsf{Y/Z} : g \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X/Z} : \lambda x.f(g(x))$

4. Backward Composition
   $\mathsf{Y\backslash Z} : g \quad \mathsf{X\backslash Y} : f \quad \Rightarrow_{\mathbf{B}} \quad \mathsf{X\backslash Z} : \lambda x.f(g(x))$

As an example, forward composition is used in the following partial parse of the sentence ‘Mary might love John’.

$$\cfrac{\cfrac{\text{might}}{\mathsf{S\backslash NP/(S\backslash NP)} : \lambda f \lambda x.might(f(x))} \quad \cfrac{\text{love}}{\mathsf{S\backslash NP/NP} : \lambda y \lambda z.love(z,y))}}{\mathsf{S\backslash NP/NP} : \lambda y \lambda x.might(love(x,y))} {>}\mathbf{B}$$

The CCG rules of forward and backward composition perform function composition into a function ranging over the same (single) argument as the secondary functor. In general it may, however, be desirable to perform function composition into results with multiple arguments. For this reason Steedman (2000) defines *Generalized Composition* using a \$ convention to define complex categories of bounded complexity in which all of the slash operators align. The category $/\$_1$ is a functional category of small finite valency in which all of the slash operators are forward slashes. The category $\backslash\$_1$ is a functional category of small finite valency in which all of the slash operators are backward slashes. These are used to define the combinatory schemata for Generalized composition.

5. Generalized Forward Composition
   $\mathsf{X/Y} : f \quad \mathsf{(Y/Z)}/\$_1 : \ldots\lambda z.g(z,\ldots) \quad \Rightarrow_{>\mathbf{B^n}} \quad \mathsf{(X/Z)}/\$_1 : \ldots\lambda z.f(g(z,\ldots))$

6. Generalized Backward Composition

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

An example of which is the following partial parse of the sentence ‘Mary might love John in an aeroplane’.

$$\begin{array}{cccc}
\text{might} & \text{love} & \text{John} & \text{in an aeroplane} \\
\hline
\mathsf{S\backslash NP/(S\backslash NP)} & \mathsf{S\backslash NP/PP/NP} & \mathsf{NP} & \mathsf{PP} \\
\lambda f\lambda x.might(f(x)) & \lambda y\lambda z\lambda x.love(x,y,z)) & & \\
\hline
\multicolumn{2}{r}{>\mathbf{B}^2} & & \\
\multicolumn{2}{l}{\mathsf{S\backslash NP/NP/PP} : \lambda y\lambda z\lambda x.might(love(x,y,z))} & &
\end{array}$$

Steedman (2000) sets a bound of 4 on the allowed valency of the category $\$$ in rules 5 and 6. I do the same here.

All of the combinatory rules presented here perform semantic composition as well as syntactic combination. They will be used by the parsers presented in Chapters 4 to 6 to parse sentences onto compositional logical expressions. They will also be used to propose parses during learning as we shall see in Chapter 3. Along with the Rules 1-6 outlined above, CCG has access to rules of *Type Raising*, *Substitution*, *Crossing Composition* and *Crossing Substitution*. These are unneccessary for the experiments run in Chapters 4, 5 and 6. For this reason they are not used here. They could, however, easily be integrated into the present approach as they all obey the principles of categorial and combinatory type transparency.

2.2.3 Semantic Type and Syntactic Category

The close link between semantic type and syntactic category is used by the combinators of CCG to ensure that the semantic composition of two categories' semantic components is allowed whenever syntactic combination of their syntactic components are. In the following example, forward application is used to combine two categories:

$$\mathsf{S\backslash NP/NP} : \lambda x\lambda y.loves(y,x) \quad \mathsf{NP} : john \quad \Rightarrow \quad \mathsf{S\backslash NP} : \lambda y.loves(y,john)$$

This combinator is licensed by the type equivalence of the range of the syntactic functor (NP) with the type of the argument. Once the combinator is licensed at the level of the syntax, there is no need for further type checking at the level of the semantics since we know, through the principle of combinatory type transparency, that the semantic argument's type must match that of the functor's range. The syntactic licensing of semantic composition requires each CCG category to be associated with a single semantic type.

As well as being required during parsing, the link between semantic type and syntactic category will also be used later in Chapter 3 to generate the syntactic categories of constituents for which the semantics is known. This reverses the mapping used above and now, for any given semantics, the principle of categorial type transparency is used to propose a consistent set of CCG syntactic categories. There is currently no good one-to-one mapping between semantic type system and basic syntactic category and in Montague's grammar (with basic semantic types e and t) there is no distinction made between the semantic types of nouns and intransitive verbs (both with type $\langle e, t\rangle$). When the mapping from semantic type to syntactic category is used to generate the syntactic contents of a grammar, it is desirable to keep this mapping as close as possible to one-to-one. I will address this by choosing the basic syntactic categories to match the semantic typeset in Chapters 4 and 5 and by adding sub-categorisation features to certain semantic types in Chapter 6.

2.2.4 Parsing with CCG

The Cocke-Younger-Kasami (CYK) algorithm (Algorithm 1) is a commonly used chart parsing algorithm that uses a grammar G to parse a sentence of length n in $O(n^3|G|)$ time. In principle, the number of categories, and therefore rules in G may be unbounded for a CCG grammar. However, in practice, the grammar induction algorithms that I present in this thesis only ever generate a finite set of CCG categories. Furthermore, the parsers used in Chapters 4, 5 and 6 prune the contents of each cell in the CYK chart using a beam of size b. With this pruning, the asymptotic complexity of the CYK algorithm is $O(n^3b^2)$. The CYK parsing algorithm is used extensively in the systems presented in this thesis and shall also later be inverted for *parse proposal* when learning grammars from a corpus of (sentence, logical-form) pairs.

```
Input  : Sentence $[w_{0:1}, \ldots, w_{n-1:n}]$, Lexicon $\Lambda$, CCG combinators $Comb$
Output: Packed chart Chart
Chart $= [\ [\{\}_1, \ldots, \{\}_n]_1,\ \ldots\ , [\{\}_1, \ldots, \{\}_n]_n\ ]$;
for $i = 1, \rightarrow, n$ do
    for $j = 1, \rightarrow, n$ do
        Chart$[i][j] = \{\mathsf{A} : a \mid \langle \mathrm{w_{i:i+j}} \vdash \mathsf{A} : a \rangle \in \Lambda\}$;
for $i = 2 \rightarrow n$ do
    for $j = 1 \rightarrow (n - i) + 1$ do
        for $k = 1 \rightarrow i - 1$ do
            for $\mathsf{B} : b \in$ Chart$[j][k]$ do
                for $\mathsf{C} : c \in$ Chart$[j + k][i - k]$ do
                    Chart$[j][i]$ = Chart$[j][i] \cup \{\mathsf{A} : a \mid \mathsf{B} : b\ \ \mathsf{C} : c\ \rightarrow \mathsf{A} : a \in$
                    $Comb\}$ ;
```

Algorithm 1: The Cocke-Younger-Kasami parsing algorithm

Chapter 3

Generating Derivations from Sentence and Logical-Form

The previous chapter introduced a logical representation of natural language semantics and the CCG grammar that can be used to build compositional logical-forms from the surface form of the sentence via a *derivation*. This thesis is concerned with learning probabilistic CCG grammars that can parse sentences onto logical-forms. These probabilistic grammars consist of: a CCG lexicon Λ; the CCG combinatory schemata introduced in Section 2.2; and a probabilistic model that chooses between competing parses of a sentence. They will be learnt from a set of N (sentence, logical-form) pairs $\{(s_i, m_i)|i = 1, \ldots, N\}$. When inducing grammars for semantic parsing, this training set represents the input and ideal output of the parser being learnt. When modelling language acquisition, the training data represents the linguistic and non-linguistic input available to the child.

The combinators of CCG are fixed. The lexicon and probabilistic model are initially unknown and must be learnt from the training data. In the systems presented here, both the lexicon and probabilistic parsing model are learnt from *allowed CCG derivations* of the (sentence, logical-form) pairs in the training data. Lexical items are read directly off the derivations. The probabilistic model is learnt by accumulating statistics over set of all allowed derivations of all training pairs.

The allowed derivations $\{t\}$ of the pair (s, m) are returned by the function $\mathcal{T}$ from Equation 1.2:

$$\{t\} = \mathcal{T}(s, m)$$

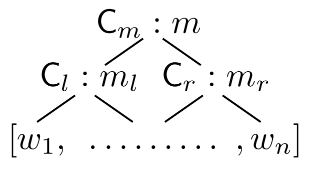
$\mathcal{T}$ proposes all combinatorially possible parses of the sentence s that return the logical-form m at their root. This chapter is dedicated to a general definition of the function

$\mathcal{T}$. In later chapters, $\mathcal{T}$ is adapted to the task of incremental lexical expansion for inducing a semantic parser (Chapter 4) and generation of all possible derivations when modelling language acquisition (Chapter 6). In both of these implementations, the core functionality of $\mathcal{T}$ is the same as the general case presented here.

3.1 Introduction

$\mathcal{T}$ generates all parses $\{t\}$ of the pair (s, m) by first proposing a parse tree root node that spans all s and then recursively splitting this root node to build a binary CCG parse tree. The root node proposed by $\mathcal{T}$ contains the logical-form m and a CCG syntactic category $\mathsf{C_m}$ that is generated via CCG's mapping from semantic type to syntactic category as described in Section 2.2.3. $\mathcal{T}$ then proposes all binary CCG parse trees that span the sentence s by recursively *splitting* the root node $\mathsf{C_m} : m$.

A single split of the CCG category $\mathsf{C_m} : m$ is a linearly ordered pair of CCG categories $(\mathsf{C_l} : m_l, \mathsf{C_r} : m_r)$ that can be re-combined via the combinatory rules of CCG to give the original category $\mathsf{C_m} : m$. Each half of a category split is used to label the top node of a sub-tree in a parse t. The contents of the subtree rooted at $\mathsf{C_l} : m_l$ is generated by either mapping the root category directly onto the word-span that it covers or by recursively splitting the both the category and word-span. The same is done for the subtree rooted at $\mathsf{C_r} : m_r$. A schema for parse trees is given below.



When proposing parses, $\mathcal{T}$ does not just consider a single split of any parse node. Rather it proposes all possible splits of that parse node. The set of all possible splits of a parse node with CCG category $\mathsf{C_m} : m$ spanning words $[w_1, \ldots, w_n]$ is the cross product of all category splits $(\mathsf{C_l} : m_l, \mathsf{C_r} : m_r)$ with all splits of the word-span. Key to $\mathcal{T}$ is the function $\mathcal{S}$ that generates all splits of a CCG category:

$$\{(\mathsf{C_l} : m_l, \mathsf{C_r} : m_r)\} = \mathcal{S}(\mathsf{C_m} : m). \tag{3.1}$$

Which itself makes use of the function $\mathcal{S}^m$ that generates all splits (f, g) of a logical expression h. This function performs the *Semantic Decomposition* of h.

$$\{(f, g)\} = \mathcal{S}^m(h). \tag{3.2}$$

This chapter describes first how the semantic decomposition function $\mathcal{S}^m$ is defined. Then I show how this function may be combined with the *inverted CCG combinators* to propose all splits of a CCG category into a linearly ordered pair (the function $\mathcal{S}$). Finally, in Section 3.4 I describe how $\mathcal{S}$ is used recursively in $\mathcal{T}$.

3.2 Semantic Decomposition

This section describes the function $\mathcal{S}^m$ that is used to generate all splits (f, g) of a logical-form h via *semantic decomposition* using a *higher-order unification* like operation. The semantic decomposition of h into a single pair (f, g) shall be referred to as a *split* of h.

All of the CCG combinators introduced in Section 2.2 use either the function application or function composition operations of the lambda-calculus to combine the semantics of their two input categories. When defining semantic decomposition for parse proposal we are only interested in splits that are consistent with the CCG combinators used to build parses. Subsequently, the full set of allowable splits (f, g) of the logical expression h can be defined as:

$$\{(f,g) \mid h = f(g)\} \ \cup \ \{(f,g) \mid \lambda x.h(x) = \lambda x.f(g(x))\} \tag{3.3}$$

where h can be recreated from f and g either via function application or function composition. As an example, the logical form $h = count(x, split(x) \wedge allowed(x))$, representing the meaning of the sentence 'How many splits are allowed?', could be split via a reversal of function application into the pair:

$$f = \lambda q.count(x, q(split(x), x)) \ , \qquad g = \lambda p \lambda z.p \wedge allowed(z) \tag{3.4}$$

And the primary functor from this pair could then itself be split via a reversal of function composition. If $h = \lambda q.count(x, q(split(x), x)$, then the following f and g are allowed:

$$f = \lambda m.count(x, m(x)) \ , \qquad g = \lambda q \lambda z.q(split(z), z) \tag{3.5}$$

Solving either of the conditionals in Equation 3.3 in order to find a single (functor, argument) pair (f, g) is a specific case of higher-order matching - a sub-case of higher-order unification - (Stirling, 2010) as it requires substitutions to be found for the potentially higher-order variables f and g. However, the function $\mathcal{T}$ requires that we solve these equations for all possible pairs (f, g). Without any constraints on the form of f and g Equation 3.3 describes an infinite set of logical expression pairs. Fortunately, there are a set of justifiable constraints that can be used to constrain the number of possible pairs to be considered.

3.2.1 Logical Constraints on Semantic Decomposition

Logical constraints on semantic decomposition are used to restrict the number of possible pairs (f, g) into which h can be split. Given the two constraints below there are a finite number solutions to Equation 3.3 for any logical expression h.

No Vacuous Variables are allowed in either f or g. Vacuous variables are variables that are bound by a lambda-term but which do not occur in the body of the expression. For example, the variable x is vacuous in $\lambda x.y$ iff the logical expression y does not contain the variable x. Vacuous variables have no reasonable meaning within our model of semantics (lambda-terms binding vacuous variables discard any argument that they apply to).

Each Pair is Unique under variable renaming and β reduction of the two separate components. Any two pairs of expressions that are equal under these operations are equivalent under any model. Once we have one, we do not need the other.

In addition to these general constraints, I later experiment with extra constraints on semantic decomposition. These are based upon analysis of a specific task (Section 4.4.3) and beliefs about linguistic universals (Section 6.4).

3.2.2 Semantic Decomposition as Sub-Tree Substitution

Compositional logical expressions may be represented as trees (Knight, 1989) in which the vertices represent logical constants, non-logical constants, variables or lambda terms. The directed edges represent parent-child relationships. As an example, the logical expression $\lambda x.f(x) \wedge g(x)$ is represented as a tree in Figure 3.1.
$\mathcal{S}^m$ represents h as a logical tree T_h and generates a set of logical tree pairs $\{(T_f,\ T_g)\}$ that represent the splits $\{(f, g)\}$ of h. Figure 3.2 illustrates the logical split from Equation 3.4. The original tree T_h is split into a functor tree T_f and an argument tree T_g. I will use this split as a running example. First I illustrate the process used to generate this single tree split, then I go on to describe how all tree splits are generated by $\mathcal{S}^m$.

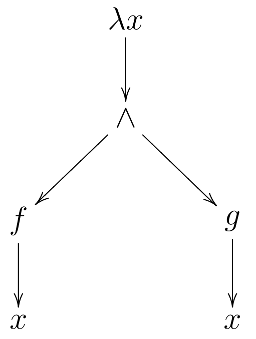
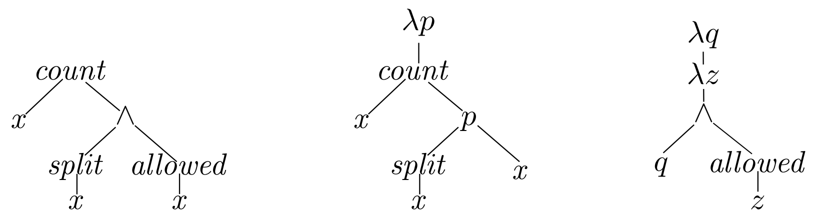


Figure 3.1: Tree Representation of $\lambda x.f(x) \wedge g(x)$.



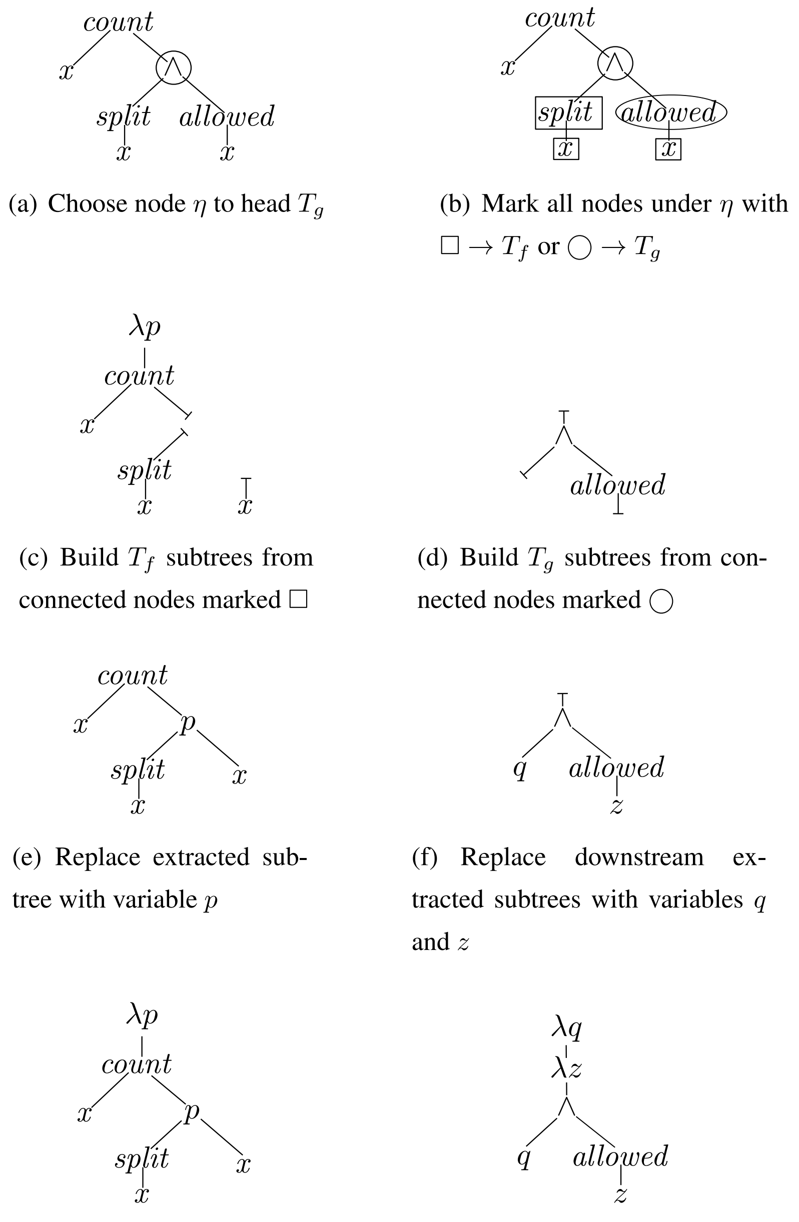
(a) T_h (b) T_f (c) T_g

Figure 3.2: Example tree split

Figure 3.3 illustrates the generation of the tree split above. First, in (a), a node η from T_h is chosen to be the root of the new argument tree T_g. Then, for each node below this in T_h a choice is made as to whether that node should go in T_f or T_g. Variables always go in the same tree as their binder. The marking of each node's destination is illustrated in Figure 3.3(b). Once all nodes in the sub-tree rooted at η are marked with a destination, the algorithm builds sub-trees out of similarly marked connected nodes. In our example, there are three sub-trees that are marked for T_f (Figure 3.3(c)) and one that is marked for T_g (Figure 3.3(d)).

The three sub-trees destined for T_f are joined by substituting a variable p for the single sub-tree in Figure 3.3(d). This is a *sub-tree substitution.* The resulting tree is illustrated in Figure 3.3(e). Two sub-tree substitutions are used to create Figure 3.3(f). One for each of the sub-trees in Figure 3.3(c) that occurred below η in the original logical tree. Figures 3.3(e) and 3.3(f) illustrate a single logical tree with free variables for both T_f and T_g. These free variables are bound with lambda-terms in the final logical trees, illustrated in Figures 3.3(g) and 3.3(h).

The key procedure in the single split routine illustrated in Figure 3.3 is the one used to build the connected sub-trees in Figure 3.3(c) and 3.3(d). This procedure walks up



(a) Choose node η to head T_g

(b) Mark all nodes under η with $\square \to T_f$ or $\bigcirc \to T_g$

(c) Build T_f subtrees from connected nodes marked $\square$

(d) Build T_g subtrees from connected nodes marked $\bigcirc$

(e) Replace extracted subtree with variable p

(f) Replace downstream extracted subtrees with variables q and z

(g) Add λ term for p

(h) Add λ terms for q and z

Figure 3.3: Illustration of logical tree splitting algorithm used to generate single split of logical tree T_h into functor tree T_f and argument tree T_g.

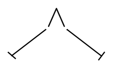
from the leaves of the original tree T_h and builds connected sub-trees one node at a time. The next section describes the operations used by this procedure at a node η in T_h when the connected sub-trees associated with each of η's children have already been ascertained and the intended destination (T_f or T_g) is known for all nodes.

3.2.3 `BuildSubTree`

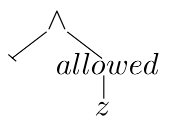
`BuildSubTree` (Algorithm 2) builds a new sub-tree η_{new} rooted at node η and returns, along with this sub-tree, a set of sub-trees $\eta_{\mathbf{out}}$ that have been substituted in the sub-tree rooted at η_{new}. Each node has the following attributes:

$\eta.children$ **:** Children of η in T_h
$\eta.target$ **:** f if η marked for T_f, g if η marked for T_g
$\eta.linkedVar$ **:** Variable with which η has been substituted (if any)

`BuildSubTree` takes as input a node η and for each of η's children a pair $(a^i, \mathbf{a_{out}}^i)$ where a^i is a sub-tree rooted at η's i'th argument and $\mathbf{a_{out}}^i$ is a list of all the sub-trees that have been substituted in a^i. For the node η, `BuildSubTree` creates a sub-tree η_{new} with an empty slot for each of η's children. E.g. continuing with the running example from Figure 3.3:

$\eta_{new} =$  $, \ (a^1, \mathbf{a_{out}}^1) = (split(x), [\,]) \ , \ (a^2, \mathbf{a_{out}}^2) = (allowed(y), [x])$

Then `BuildSubTree` cycles through η's children and checks if the associated a_i has the same target as η (Algorithm 2 line 6). If a_i has the same target as η then it is set as η_{new}'s i'th argument. This maintains the structure from T_h. All of the substituted sub-trees in $\mathbf{a_{out}}$ are added to $\eta_{\mathbf{out}}$. This is because they have been substituted in the tree rooted at η_{new} as well as the tree rooted at a_i. In our example, a^2 has the same target (g) as η. η_{new} and $\eta_{\mathbf{out}}$ are updated as follows:

$\eta_{new} =$  $, \quad \eta_{\mathbf{out}} = \{x\}$

If a^i does not have the same target as η then a new variable v is generated and set as η_{new}'s i'th argument as a *sub-tree substitution* for a^i. Each of the sub-trees in $\mathbf{a_{out}}^i$ is added to v as an argument (line 12) and a λ-term is created to bind the associated

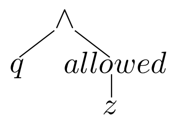
```
Algorithm: BuildSubTree
Input   : Node $\eta$ with $n$ children,
          List of pairs $P = [(a^1, \mathbf{a_{out}}^1), \ldots, (a^n, \mathbf{a_{out}}^n)]$
Output: New subtree $\eta_{new}$,
          Set of substituted subtrees $\eta_{\mathbf{out}}$
begin
    $\eta_{new} = \eta$;
    $\eta_{\mathbf{out}} = \{\ \}$;
    for $i = 0 \to n - 1$ do
        $(a^i,\ \mathbf{a_{out}}^i) = P[i]$ ;
        if $a.target = \eta.target$ then
            $\eta_{new}.\mathtt{setArg}(i, a^i)$;
            $\eta_{\mathbf{out}} = \eta_{\mathbf{out}} \cup \{\mathbf{a_{out}}^i\}$ ;
        else
            $v =$ New variable ;
            for $k = 0 \to \mathtt{count}(\mathbf{a_{out}}^i) - 1$ do
                $v.\mathtt{setArg}(\mathtt{count}(v.children), \mathbf{a_{out}}^i[k])$;
                $u = \mathbf{a_{out}}^i[\mathtt{count}(\mathbf{a_{out}}^i) - k - 1].linkedVar$;
                $a^i = \lambda u.a^i$;
            end
            $\eta_{\mathbf{out}} = \eta_{\mathbf{out}} \cup \{a^i\}$;
            $v.\mathtt{setType}(\mathtt{type}(a^i))$;
            $a^i.linkedVar = v$;
            $\eta_{new}.setArg(i, v)$;
        end
    end
    return $(\eta_{new},\ \eta_{\mathbf{out}})$;
end
```

Algorithm 2: Build tree η_{new} with variable substitutions for sub-trees in $\eta_{\mathbf{out}}$

free variable in a^i (line 14). The order in which these λ terms are added to a^i correlates with the order in which the related arguments are added to v. The type of v is the same as the type of a^i once the λ-terms have been added. The new variable v has been substituted for a^i in the tree rooted at η_{new}. Consequently, a^i is added to the set of substituted sub-trees $\eta_{\textbf{out}}$.

In our running example, a^1 does not have the same target as η. The tree rooted at a^1 is therefore substituted with the variable p. There are no previously substituted trees in $\mathbf{a_{out}}^i$. η_{new} and $\eta_{\textbf{out}}$ are updated as follows:

$$\eta_{new} =$$



$$, \qquad \eta_{\textbf{out}} = \{x, split(x)\}$$

And this is returned since all argument slots have been filled. η_{new} is the tree rooted at η with sub-tree substitutions. $\eta_{\textbf{out}}$ contains the sub-trees that were substituted.

```
Algorithm: BuildAllSubTrees
Input  : Node $\eta$ with $n$ children
Output: Set of pairs $Q = \{(\eta_{new}{}^{i}, \eta_{\textsf{out}}{}^{i}) \mid i = 0, \ldots, n-1\}$
begin
    $Q = \{\ \}$;
    if $n = 0$ then
        $\eta_{new} = \eta, \ \eta_{new}.target = f$;
        $Q = Q \cup \{\eta_{new}\}$;
        $\eta_{new} = \eta, \ \eta_{new}.target = g$;
        $Q = Q \cup \{\eta_{new}\}$;
    else
        $A = [[\ ]_0, \ldots, [\ ]_{n-1}]$;
        for $i = 0 \to n-1$ do
            $\eta.children[i].target = null$;
            $A[i] =$ BuildAllSubTrees$(\eta.children[i])$ ;
        end
        for $B \in A[0] \times \cdots \times A[n-1]$ do
            if $\eta.target = null$ then
                $\eta.target = f$;
                $(\eta_{new}, \eta_{\textsf{out}}) =$ BuildSubTree $(\eta, B)$;
                for $\eta_{\textsf{out}}{}' \in$ Permutations$(\eta_{\textsf{out}})$ do
                    $Q = Q \cup \{(\eta_{new}, \eta_{\textsf{out}}{}')\}$;
                end
            end
            $\eta.target = g$;
            $(\eta_{new}, \eta_{\textsf{out}}) =$ BuildSubTree $(\eta, B)$ ;
            for $\eta_{\textsf{out}}{}' \in$ Permutations$(\eta_{\textsf{out}})$ do
                $Q = Q \cup \{(\eta_{new}, \eta_{\textsf{out}}{}')\}$;
            end
        end
    end
    return $Q$;
end
```

Algorithm 3: Build all trees rooted at η with variable substitutions for subtrees.

3.2.4 `BuildAllSubTrees`

`BuildAllSubTrees` (Algorithm 3) is a recursive function algorithm that returns, for an input node η, a set of pairs $\{(\eta_{new},\ \eta_{\mathbf{out}})\}$. Each of these pairs contains a sub-tree η_{new} and an ordered list of the substitutions made in this sub-tree $\eta_{\mathbf{out}}$. Unless the target of η has been determined by a higher process, the set contains all sub-trees rooted at η marked for both T_f and T_g. Otherwise it contains all sub-trees rooted at η marked for just T_g - an option that is only used when η is the top node of an argument sub-tree.

If the input node η has no children, then `BuildAllSubTrees` returns two sub-trees. One is a copy of η that has been marked for T_f. The other is a copy of η that has been marked for T_g.[1]

If η has one or more children, `BuildAllSubTrees` calls itself on each of η's n children to get all sub-trees rooted below η. $A[i]$ stores the result of `BuildAllSubTrees` called on η's i'th child. In our running example, when $\eta = \wedge$, A is:

$$A[1] = [(split^f(x)\ ,\ [\,]), (split^g(y)\ ,\ [x])] \tag{3.6}$$

$$A[2] = [(allowed^f(x)\ ,\ [\,]), (allowed^g(z)\ ,\ [x])]. \tag{3.7}$$

The cartesian product of the sets contained in A is used to generate a set of tuples $\{B\} = \{\langle(a^0, \mathbf{a_{out}}^0), \ldots, (a^{n-1}, \mathbf{a_{out}}^{n-1}\rangle\}$ representing all possible argument assignments (line 13). Each tuple B contains a single $(a^i, \mathbf{a_{out}}^i)$ for each of η's children. In our example, $\{B\}$ is:

$$B = \{\langle(split^f(x), [\,])\ ,\ (allowed^f(x)\ ,\ [\,])\rangle,\ \langle(split^f(x), [\,])\ ,\ (allowed^g(z)\ ,\ [x])\rangle,$$
$$\langle(split^g(y), [x])\ ,\ (allowed^f(x)\ ,\ [\,])\rangle,\ \langle(split^g(y), [x])\ ,\ (allowed^f(z)\ ,\ [x])\rangle\}$$

And for each of these tuples representing a full set of argument assignments for η `BuildAllSubTrees` generates two sub-trees rooted at η. One with η_{new} marked for T_f (line 16) and one with η_{new} marked for T_g (line 22). For each of these two sub-trees all permutations are found of the substituted sub-tree set $\eta_{\mathbf{out}}$ (lines 17 and 23). The set of logical sub-trees in $\eta_{\mathbf{out}}$ will be added as arguments to a newly created variable higher in the original tree. Generating all permutations of $\eta_{\mathbf{out}}$ ensures that `BuildAllSubTrees` returns all allowed argument orders — and by extension

[1]For reasons of clarity the pseudo-code ignores the fact that variables may only be marked with one destination.

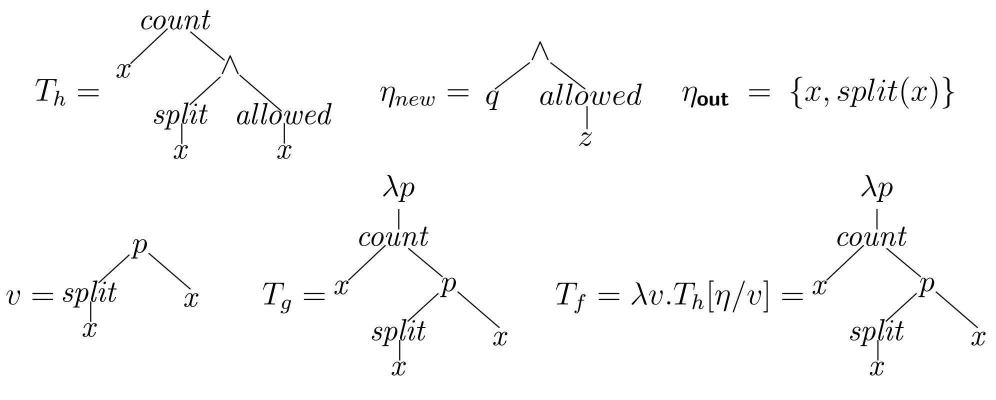
all orders of the λ-terms added in Algorithm 2. In our running example, the node $\eta = \wedge$ has been marked for T_g only. The full set of $(\eta_{new}, \eta_{\textsf{out}})$ pairs returned by `BuildAllSubTrees` is:

$$\begin{aligned} Q = \{&(p \wedge q, [split(x), allowed(x)]),\ (p \wedge q, [allowed(x), split(x)]),\\ &(p \wedge allowed(z), [split(x), x]),\ (p \wedge allowed(z), [x, split(x)]),\\ &(split(y) \wedge q, [allowed(x), x]),\ (split(y) \wedge q, [x, allowed(x)]),\\ &(split(y) \wedge allowed(y), [x])\} \end{aligned}$$

3.2.5 Generating All Splits

Finally, having defined `BuildSubTree` and `BuildAllSubTrees` it is possible to define the function $\mathcal{S}^m$ that generates all splits (T_f, T_g) of T_h. This is illustrated in Algorithm 4.

$\mathcal{S}^m$ generates splits of T_h by cycling over all nodes η in T_h and generating all sub-trees and sub-tree substitution sets $(\eta_{new}, \eta_{\textsf{out}})$ with `BuildAllSubTrees`. T_f is then created by substituting a new variable v for η in T_h, adding each of the logical sub-trees in $\eta_{\textsf{out}}$ to this variable as arguments and adding a lambda-term binding v at the root of the new logical tree. T_g is generated from η_{new} and $\eta_{\textsf{out}}$ by adding to η_{new} a single lambda term binding each of the variables that was used to replace one of the sub-trees in $\eta_{\textsf{out}}$. The generation of the (T_f, T_f) from our running example is illustrated below.



Lines 16 to 21 are used to generate logical tree pairs that represent splits of T_h that have been performed by the reversal of *function composition*. In this case, T_g remains the same as the one allowed by a reversal of *function application*. Line 17 checks that the highest scoped λ-term in T'_h is one that could be returned by function composition of a functor with T_g. T_f is generated by removing λ terms from the root of T'_h and their bound variables from the tree under v before substituting v for η.

```
Algorithm: $\mathcal{S}^m$
Input  : Tree to split $T_h$
Output: Set of pairs $S = \{(T_f, T_g)\}$
begin
    $S = \{\}$;
    for $\eta \in T_h$ do
        $\eta.target = g$;
        for $(\eta_{new}, \eta_{\mathsf{out}}) \in \texttt{BuildAllSubTrees}(\eta)$ do
            $v =$ New variable ;
            for $k = 0 \to \texttt{count}(\eta_{\mathsf{out}}) - 1$ do
                $v.\texttt{setArg}(\texttt{count}(v.children), \eta_{\mathsf{out}}[k])$;
                $u = \eta_{\mathsf{out}}[\texttt{count}(\eta_{\mathsf{out}}) - k - 1].linkedVar$;
                $\eta_{new} = \lambda u.\eta_{new}$;
            end
            $T_g = \eta_{new}$;
            $v.\texttt{setType}(T_g)$;
            $T_f = \lambda v.T_h[\eta/v]$ ;
            $S = S \cup \{(T_f, T_g)\}$ ;
            $T'_h = T_h,\ T'_g = T_g$;
            while $T'_h = \lambda z.T''_h \wedge z \notin T''_h[\eta/\epsilon] \wedge T'_g = \lambda z'.T''_g \wedge z \equiv z'$ do
                $v.\texttt{setType}(T''_g)$;
                $T_f = \lambda v.T''_h[\eta/v][z/\epsilon]$ ;
                $S = S \cup \{(T_f, T_g)\}$ ;
            end
        end
    end
    return $S$;
end
```

Algorithm 4: Generate all splits (T_f, T_g) of T_h.

3.2.5.1 Conjunctions and Disjunctions

Algorithm 4 will not generate all splits of a conjunction or disjunction with more than two conjuncts or disjuncts. For example, given a single tree representation of the logical expression $a \wedge b \wedge c$, it is not possible to perform both of the following splits.

$$a \wedge b \wedge c \;\Rightarrow\; \lambda x.a \wedge x \qquad b \wedge c$$

$$a \wedge b \wedge c \;\Rightarrow\; \lambda x.b \wedge x \qquad a \wedge c$$

For each conjunction and disjunction of arity > 2 in T_h a new tree $T_h^{\star}$ is created for each subset of more than one conjuncts (or disjuncts). This tree introduces a new conjunction (or disjunction) that has this subset as arguments and which is added to the original conjunction (or disjunction) in the place of the conjunct subset. This new node is chosen as η and the splitting algorithm proceeds as described.

3.2.6 Complexity

The complexity of Algorithm 4 is exponential in the number of nodes in T_h. The greatest contributor to the complexity in the vast majority of cases is the Cartesian product on line 13 of Algorithm 3. The complexity of this Cartesian product can be calculated using the recursive function below:

$$|\texttt{BuildAllSubTrees}(\eta)| = \begin{cases} 2 & n = 0 \\ \prod_{i=0}^{n-1} |\texttt{BuildAllSubTrees}(a_i)| & \text{otherwise} \end{cases} \tag{3.8}$$

giving a complexity of 2^N for a node η which heads a tree containing N nodes.

The number of permutations of the substituted sub-trees, calculated in lines 17 and 23 of Algorithm 3 is factorial in the size of $\eta_{\textsf{out}}$. Also the number of tree manipulations required to model all possible splits of a conjunction or disjunction is exponential in the number of conjuncts or disjuncts.

The computational cost of $\mathcal{T}$ is sufficiently large to preclude its use with complex logical-forms. However, in practice, extra constraints to the splitting procedure can be introduced to restrict the complexity of each of the three key contributors listed above. We shall see examples of these in Chapters 4 and 6.

3.3 Syntactic Splitting

The section above describes an algorithm for splitting an original logical expression h into a pair of logical expressions (f, g). This section describes how the combinators of CCG can be used along with a mapping from semantic type to syntactic category to assign syntactic categories and linear order to the two halves of the logical expression split.

Given a CCG category $\mathsf{X} : h$ and a split of the logical expression h into a single pair (f, g) there will be two ways of splitting the syntactic category X into a pair of linearly ordered syntactic categories. All category splits are licensed by an *Inverted CCG Combinator*.

Inverted CCG Combinators :

1. Inverted Forward Application
   $\mathsf{X} : f(g) \quad \Rightarrow \quad \mathsf{X/Y} : f \quad \mathsf{Y} : g$

2. Inverted Backward Application
   $\mathsf{X} : f(g) \quad \Rightarrow \quad \mathsf{Y} : g \quad \mathsf{X\backslash Y} : f$

3. Inverted Forward Composition
   $\mathsf{X/Z} : \lambda x.f(g(x)) \quad \Rightarrow \quad \mathsf{X/Y} : f \quad \mathsf{Y/Z} : g$

4. Inverted Backward Composition
   $\mathsf{X\backslash Z} : \lambda x.f(g(x)) \quad \Rightarrow \quad \mathsf{Y\backslash Z} : g \quad \mathsf{X\backslash Y} : f$

5. Inverted Generalized Forward Composition
   $(\mathsf{X/Z})/\$_1 : \ldots\lambda z.f(g(z, \ldots)) \quad \Rightarrow \quad \mathsf{X/Y} : f \quad (\mathsf{Y/Z})/\$_1 : \ldots\lambda z.g(z, \ldots)$

6. Inverted Generalized Backward Composition
   $(\mathsf{X\backslash Z})\backslash\$_1 : \ldots\lambda z.f(g(z, \ldots)) \quad \Rightarrow \quad (\mathsf{Y\backslash Z})\backslash\$_1 : \ldots\lambda z.g(z, \ldots) \quad \mathsf{X\backslash Y} : f$

Figure 3.4: Inverted CCG Combinators

Each of these inverted CCG combinators corresponds to one of the CCG combinators introduced in Section 2.2.2. The inverted combinators consistent with a single split (f, g) of h are licensed by a match of the combinatory operation that is required to recombine both the syntactic category and logical expression. Allowed operations are *function application* (Rules 1 and 2), *function composition onto a function of valence* 1 (Rules 3 and 4), and *function composition onto a function of valence* > 1 (Rules 5 and 6).

Like the combinators of CCG presented in Section 2.2, these inverted combinators are described by use of general schemata that uses variables to fill in for syntactic categories. Unlike the combinators of CCG, the results of the productions above contain syntactic categories that do not occur in the input (the variable Y in all examples). These syntactic categories need to be named. This is done via the mapping from semantic type to syntactic category described in Section 2.2.3.

For example, in Rules 1 and 2 given above, the allowable settings of syntactic category Y are defined by the type of the logical expression a by use of a function cat that describes the mapping from semantic type to syntactic category.

$$\{\mathsf{Y}\} = \texttt{cat}(\texttt{type}(a)) \tag{3.9}$$

In Rules 2 and 3, the allowable settings of Y are defined by the type of $g(x)$ where x is the variable from the input, the type of which corresponds to the syntactic category Z.

The mapping from semantic type to syntactic category may return more than one syntactic category. This could be for one of two reasons. Firstly, the syntactic category may be complex in which case there is nothing in the semantic type that can be used to inform the directionality of the slash operators used. Secondly, there is currently no theory of semantic type that supports a one-to-one mapping between semantic type and the commonly used atomic CCG categories. For example, in Montague's grammar which uses the two base types e (entity) and t (truth value), the noun category has the same type $< e, t >$ as an intransitive verb.

While the mapping from semantic type to syntactic category does not inform the directionality of the slash operators in complex categories, the allowable settings of the slashes are constrained by the *Principle of Inheritance* (Steedman, 2000).

> **Principle of Inheritance :** If the category that results from the application of a combinatory rule is a function category, then the slash defining directionality for a given argument in that category will be the same as the one(s) defining directionality for the corresponding argument(s) in the input func-

tion(s).

This principle is generally used by CCG to disallow compositional productions:

$$\mathsf{X/Z} \not\Rightarrow \mathsf{Y\backslash Z} \quad \mathsf{X\backslash Y}$$

but it can also be used to dissallow the applicative production:

$$\mathsf{X/Y/Z} \not\Rightarrow \mathsf{X/Y/Z^{\star}/(Y\backslash Z^{\star})} \quad \mathsf{Y\backslash Z}$$

when the $\mathsf{Z}^{\star}$ in the domain of the first functor is known to refer to the same semantic argument as the $\mathsf{Z}^{\star}$ in the range of this functor.

The function $\mathcal{S}$ takes as input a CCG category $\mathsf{X} : h$ and produces all splits $\{(\mathsf{C_l} : m_l,\ \mathsf{C_r} : m_r)\}$ of this CCG category by first splitting the logical expression h with $\mathcal{S}^m$ and then generating two category splits for each of the logical splits using the category splitting operations listed above.

$$\{(\mathsf{C_l} : m_l,\ \mathsf{C_r} : m_r)\} \ = \ \mathcal{S}(\mathsf{X} : h)$$

3.4 Lexicon Expansion via Parse Proposal

The function $\mathcal{S}$ can be recursively to define all syntactic parses that are consistent with a (sentence, meaning) pair.

CCG lexical entries $\mathrm{w} \ \vdash \mathsf{X} : h$ pair the word w with the CCG category $\mathsf{X} : h$. Lexical entries are generated by hypothesising a mapping between each of the CCG categories proposed by $\mathcal{S}$ and all of the word-spans onto which that category could be projected. The 'word' associated with a lexical entry may be a multiword element (MWE) representing a span of multiple words in s.

All parses and lexical entries consistent with a training pair (s, m) can be proposed by running the inverted version of the CYK algorithm given in Algorithm 5 to pack a parse chart from the top down. The CCG syntactic category C_m consistent with the type of m is chosen to represent the root of all syntactic parse trees. The parse chart is then populated by recursively splitting CCG categories in all parse chart cells that span more than a single word in s using the $\mathcal{S}$. A single lexical item $[\mathrm{w_i}, \ldots, \mathrm{w_j}] \ \vdash \mathsf{X} : x$ is added to the lexicon Λ for each CCG category $\mathsf{X} : x$ that maps onto word-span $[\mathrm{w_i}, \ldots, \mathrm{w_j}]$ in the fully packed chart.

```
Input  : Sentence $s = [w_1, \ldots, w_n]$,  CCG Category $\mathsf{C_m} : m$
Output: Packed chart Chart,  Lexicon $\Lambda$
Chart $= [\,[\{\}_1, \ldots, \{\}_n]_1,\ \ldots\ , [\{\}_1, \ldots, \{\}_n]_n\,]$;
Chart$[1][n-1] = \mathsf{C_m} : m$;
$\Lambda\ =\ \Lambda\ \cup \{[\mathrm{w_1}, \ldots, \mathrm{w_n}] \vdash \mathsf{C_m} : m\}$ ;
for $i = n \to 2$ do
    for $j = 1 \to (n-i)+1$ do
        for $\mathsf{X} : x\ \in$ Chart$[j][i]$ do
            for $(\mathsf{Y} : y,\ \mathsf{Z} : z)\ \in\ \mathcal{S}(\mathsf{X} : x)$ do
                for $k = 1 \to i-1$ do
                    Chart$[j][k]$ = Chart$[j][k]\ \cup\ \{\mathsf{Y} : y\}$ ;
                    Chart$[j+k][i-k]$ = Chart$[j+k][i-k]\ \cup\ \{\mathsf{Z} : z\}$ ;
                    $\Lambda\ =\ \Lambda\ \cup\ \{[\mathrm{w_j}, \ldots, \mathrm{w_{j+k-1}}] \vdash \mathsf{Y} : y\}$ ;
                    $\Lambda\ =\ \Lambda\ \cup\ \{[\mathrm{w_{j+k}}, \ldots, \mathrm{w_{j+i}}] \vdash \mathsf{Z} : z\}$ ;
```

Algorithm 5: Inverted CYK for generating CCG derivations

Chapter 4

Semantic Parser Induction

This chapter presents work that was published in Kwiatkowski et al. (2010).

4.1 Introduction

One of the aims of this thesis is to learn a parser that can map natural language sentences onto formal representations of their meaning. When retrieved, these formal representations could then be used by any downstream task that needs to do logical inference with the meaning of a sentence. In order to be used for logical inference, the formal meaning representation must be grounded in the world within which inference is going to take place. A typical 'world' within which we want to do inference is a database of entities, attributes and relations. Recent work has addressed the problem of learning the mapping from sentences onto database queries by training semantic parsers on corpora containing (sentence, logical-form) pairs. The following training pair is typical. It pairs an English sentence with a meaning representation expressed in first order predicate logic with the lambda-calculus (from now on, a lambda-expression):

Sentence: which states border texas
Meaning: $\lambda x.state(x) \wedge next\_to(x, tex)$

Although, this is not the only type of data that we may want to model. Other tasks may use different formal or natural languages. In the following example, drawn from a different corpus, the sentence is in Turkish and the meaning representation is a functional query language (FunQL) in which all of the terms are functions or entities. There are no logical formulae (c.f. Section 2.1.1).

Sentence: hangi eyaletin texas ye siniri vardir
Meaning: $answer(state(borders(tex)))$

Given (sentence, logical-form) pairs like the ones illustrated above, the goal is to learn a parser that can map new, unseen, sentences to their corresponding meaning. Previous approaches to this problem have been tailored to specific natural languages, specific meaning representations, or both. This chapter presents an approach based upon the general procedure of parse proposal presented in Chapter 3 that is agnostic both as to the natural language and as to the meaning representation used. The approach has only two requirements. First, the meaning representation must be decomposable to the extent that the words or word-spans to be lexicalised can be mapped onto separable components of meaning. Second, it must be possible to build the meaning of a sentence from these components of meaning using combinatory operations such as *function application* and *function composition.* As our approach uses the higher-order unification type operations from Section 3.2 we call it the Unification Based Learner (UBL).

The work in this chapter is motivated by the need for a black box method of inducing a semantic parser that could be adapted for any new domain and language by simply training it on a suitable dataset. The reason for generalising to multiple languages is obvious. The need to support multiple meaning representations arises from the fact that there is no standard representation of natural language meaning. Instead, existing representations are ad hoc, tailored to the application of interest. For example, the FunQL representation introduced above is well suited to a database querying task in which all of the answers are sets of entities. The current variation in meaning representations is not likely to be resolved any time soon as there is an ever present conflict between the competing demands of the expressivity of a logical representation and the ease of doing inference with that logical representation. This conflict invariably leads system designers to choose a logical representation that is *just expressive enough* - a condition that is defined by the task at hand.

The need to cover multiple languages and meaning representations justifies the parse proposal procedure $\mathcal{T}$ introduced in Chapter 3 which makes no assumptions about word order or the size of the wordspans and logical elements to be lexicalised. However, this procedure massively overgenerates parses for any one language. In order to be viable as a general purpose semantic parser UBL must be applicable to large datasets and, for this reason, UBL cannot store or explore the full space of parses proposed by $\mathcal{T}$. Instead, UBL uses the operations of semantic decomposition and syntactic production defined in Chapter 3 to incrementally expand the lexicon Λ in a way that yields high probability parses of (sentence, logical-form) training pairs. UBL scores parses with a log-linear model parameterised by the weight vector θ. For a given (sentence, logical-form) pair (s,m) in the training corpus, UBL uses the function $\mathcal{T}'$ to generate a single parse t':

$$t' = \mathcal{T}'(s,m,\Lambda,\theta) \tag{4.1}$$

containing at most two lexical items that are not already included in Λ. The parse t' is generated by performing local, efficient, manipulations to the current highest scoring parse of (s,m). The new lexical items used in t' are added to Λ. The process of lexical expansion is *incremental* and *guided by the parsing model*. The process of lexical expansion is interleaved with the process of parameter estimation for the probabilistic parsing model giving a greedy lexical search strategy that visits only a high probability portion of the lexical space supported by $\mathcal{T}$.

The rest of this chapter proceeds as follows. Sections 4.2 and 4.3 review log-linear parsing models and the stochastic gradient descent algorithm that UBL uses to score parses and learn θ respectively. Section 4.4 then describes the function $\mathcal{T}'$ that proposes a single new parse tree for each training instance and Section 4.5 shows how this function is used for lexical expansion interleaved with a parameter update step. In Sections 4.6 and 4.7, I evaluate UBL on the benchmark GeoQuery semantic parsing dataset with both the lambda-expressions and FunQL meaning representations illustrated above and with multiple natural languages. I compare the performance of UBL to previous methods (Kate and Mooney, 2006; Wong and Mooney, 2006, 2007; Zettlemoyer and Collins, 2005, 2007; Lu et al., 2008), which are designed with either language- or representation- specific constraints that limit generalisation, as discussed in more detail along with other related approaches in Section 4.9.

Despite being the only approach that is general enough to run on all of the data sets, UBL achieves similar performance to the others, even outperforming them in several cases.

4.2 Log Linear CCGs

Given a CCG lexicon Λ, there will, in general, be many possible parses for each sentence. UBL selects the most likely alternative using a log-linear model, which consists of a feature vector ϕ and a parameter vector θ. The joint probability of a logical form m constructed with a parse t, given a sentence s is defined as:

$$P(t,m|s;\theta,\Lambda) = \frac{e^{\theta\cdot\phi(t,m,s)}}{\sum_{(t',m')} e^{\theta\cdot\phi(t',m',s)}} \tag{4.2}$$

Log-linear models have been successfully used in wide coverage CCG parsing by Clark and Curran (2007) and subsequent work. Nothing in the model structure precludes complex model features defined over any discriminating attribute of parse, logical-form and sentence. This freedom to define arbitrarily complex features makes log-linear models very attractive for modelling structures with complex dependencies[1]. From the perspective of the approach presented here, the log linear model is also attractive as it does not require full initial enumeration of all possible features and feature weights. Instead, we only ever care about the features $\phi(s,m,t)$ and feature weights associated with the $\langle s,m,t\rangle$ triples supported by a given sentence s. Section 4.6 defines the features used by UBL in full. These include, for example, lexical features that indicate when specific lexical items in Λ are used in the parse t.

The parsing, or inference, problem to be solved by UBL at test time is to find the most likely logical-form m given a sentence s, assuming the parameters θ and lexicon Λ are known:

$$\mathcal{P}(s) = \arg\max_{m} p(m|s;\theta,\Lambda) \tag{4.3}$$

where the probability of the logical-form is found by summing over all parses that produce it:

$$p(m|s;\theta,\Lambda) = \sum_{t} p(t,m|s;\theta,\Lambda) \tag{4.4}$$

In practice there will often be multiple parses returning a single logical-form from a single sentence. This is partially due to the ambiguity in the lexicon resulting from the lack of syntactic supervision. It is also partially due to the *derivational ambiguity* that is implicit in CCG. (Steedman, 2000) In either case, for the task of semantic parsing,

[1]Although inference in models that assign features to non-local structures can be computationally intensive

we are only interested in the logical-form built by the parse, not the derivation itself. For this reason the distribution over derivations t is modelled as a hidden variable.

The sum over parses in Eq. 4.4 does not require explicit enumeration of each distinct parse t. It can be calculated with the inside-outside algorithm on a packed parse chart that has been built with a CYK parsing algorithm. Even with the packed chart representation, when Λ is big and ambiguous, there may be too many distinct chart entries to enumerate. To maintain the speed of parsing at a reasonable level, UBL prunes the number of chart entries associated with each span according to their score under the log linear model.

4.3 Stochastic Gradient Descent

To estimate the parameters themselves, UBL uses stochastic gradient descent (Bottou, 1991). Stochastic gradient descent (SGD), also known as sequential gradient descent, is an online optimisation routine that can be used to find local minima of differentiable objective functions.

The objective function used by UBL is the negative conditional log likelihood. For the training set $\{(s_i, m_i) : i = 1, \ldots, N\}$ this is:

$$O = \sum_{i=1,\ldots,N} -\mathrm{log}P(m_i|s_i;\theta,\Lambda) \tag{4.5}$$

This objective function over the training data is parameterised by θ. Stochastic gradient descent minimises O by updating θ along the local gradient of O with respect to θ calculated on the basis of a single training example (s_i, m_i). The local gradient of O for this single training example is:

$$O_i = -\log P(m_i|s_i;\theta,\Lambda) \tag{4.6}$$

and the derivative of this with respect to a single model parameter θ_i is:

$$\begin{aligned}\frac{\partial O_i}{\partial \theta_j} &= E_{p(t|s_i,m_i;\theta,\Lambda)}[\phi_j(t,s_i,m_i)] \\ &\quad -E_{p(t,m|s_i;\theta,\Lambda)}[\phi_j(t,s_i,m_i)]\end{aligned} \tag{4.7}$$

This derivative defines the direction (positive or negative) of the update for parameter θ_j. It also partially defines the magnitude of this update. Parameter updates are also scaled according to a learning schedule. The learning schedule used by UBL is described in Section 4.6. It should be noted that the derivative in Equation 4.7 is the difference of two expectations, one conditioned on the current correct logical-form m_i and one that is not. This update will be zero when θ_j is not used in any parses of the sentence s_i that return an incorrect logical-form. As with Eq. 4.4, all of the expectations in Eq. 4.7 are calculated through the use of the inside-outside algorithm on a pruned parse chart. If parses are pruned then Equation 4.7 becomes an approximation to the gradient. However, as the pruning is performed on the basis of parse probability, the pruned chart will contain most of the true chart's probability mass making this approximation a reasonable one. The impact of this approximation will be assessed further in Section 5.7.

4.4 Guided Lexical Expansion

Chapter 3 presented a definition of the function $\mathcal{T}$ that proposes all parses t consistent with the training pair (s, m) and describes how this function can be used to populate a lexicon Λ from a corpus of (sentence, logical expression) pairs. The vast majority of lexical items proposed by $\mathcal{T}$ will be incorrect for any given target language. Furthermore, for any reasonable sized training set the number of lexical items proposed by $\mathcal{T}$ will be prohibitively large. For this reason, UBL uses a probabilistically guided lexical expansion step that adds at most two lexical items per training example. Instead of using $\mathcal{T}$ to propose all parses $\{t\}$ of the current training example (s_i, m_i), UBL uses the current lexicon and parsing model to guide *constrained parse proposal*, generating a single parse t' with the function $\mathcal{T}'$.

$$t' \; = \; \mathcal{T}'(s_i, m_i, \Lambda, \theta) \tag{4.8}$$

The rest of this section describes $\mathcal{T}'$ and how it is used to incrementally expand the lexicon on the basis of a single training pair (s_i, m_i), the current lexicon Λ and the current parameter set θ.

4.4.1 Initial Lexicon

Before training UBL the lexicon is initialised with a single lexical item per training pair consisting of the entire sentence s_i and its associated meaning representation m_i paired with syntactic category S.

$$\Lambda_0 \; = \; \{\mathrm{s_i} \vdash \mathsf{S} : m_i \; : \; i, \ldots, N\} \tag{4.9}$$

An example initial lexicon, for the two sentence dataset containing 'New York borders Vermont' and 'New York borders Massachussetts' is:

$$\begin{aligned} \text{New York borders Vermont} &\vdash \mathsf{S} : next\_to(ny, vt) \\ \text{New York borders Massachusetts} &\vdash \mathsf{S} : next\_to(ny, mass) \end{aligned}$$

Although these initial, sentential lexical items can parse the training sentences, they will not generalise well to unseen sentences. To learn effectively, we will need to recursively split overly specific entries of this type into pairs of new, smaller, entries that generalise better. E.g. the lexicon:

$$\begin{aligned} \text{New York} &\vdash \mathsf{NP} : ny \\ \text{Vermont} &\vdash \mathsf{NP} : vt \\ \text{Massachusetts} &\vdash \mathsf{NP} : mass \\ \text{borders} &\vdash \mathsf{S\backslash NP/NP} : \lambda x \lambda y.next\_to(y, x) \end{aligned}$$

which could parse an unseen example such as "Massachusetts borders Vermont".

4.4.2 Targeted Splitting

The recursive splitting of the original lexical items in Λ_0 is done through the processes of semantic decomposition and parse proposal introduced in Chapter 3. However, in order to restrain the number of parses proposed, $\mathcal{T}'$ only applies the splitting procedure to the parse nodes in the current highest scoring correct parse of the sentence. In this sense, correct parses of a training sentence s_i are those that return the correct logical expression m_i. The highest scoring correct parse t_i^* is then:

$$t_i^* = \arg\max_t p(t|s_i, m_i; \Lambda, \theta) \tag{4.10}$$

where each parse t must use lexical items drawn from the current lexicon Λ. This parse is computed using the CYK algorithm (Algorithm 1) to pack a parse chart, and the Viterbi algorithm to choose the highest scoring derivation. Each of the nodes in t_i^* describes a potential lexical item spanning a string of words and having a CCG syntactic category and logical expression associated with it. Λ is expanded by adding either the single lexical item created by lexicalising one of the parse nodes in t_i^*, or a pair of lexical items created via a single split of one of the parse nodes in t_i^*. Below, I describe how UBL simplifies the semantic decomposition and syntactic production operations used by $\mathcal{T}$ in order to limit the number of splits of a given parse node. Then

I describe how these simplified operations are integrated into the function $\mathcal{T}'$ before going on to introduce the full training algorithm.

4.4.3 Restricted Higher-Order Unification

The set of possible splits (f, g) for a logical expression h was defined in Chapter 3 as the solution to the pair of higher-order unification problems in Equation 3.3: either $f(g) = h$ or $\lambda x.f(g(x)) = h$. Section 3.2.1 lists a set of constraints that place a finite bound on the number of possible splits. However, the number of splits for h proposed by the function $\mathcal{S}$ illustrated in Algorithm 4 is still exponential in the complexity of h. This exponential complexity stems from two sources—the number of subsets of a coordination's arguments, and the number of combinations of disconnected logical substructures. These are described below along with extra constraints that UBL uses to define a new overcome them.

When h contains a conjunction or disjunction, such as $h = \lambda x.city(x) \wedge major(x) \wedge in(x, tex)$, $\mathcal{S}$ allows any subset of the expressions in the conjunction (or disjunction) to be assigned to f (or g). The number of subsets for a conjunction with n conjuncts is:

$$\sum_{k=1}^{n} {}_nC_k = 2^n - 1.$$

And in order to avoid listing an exponential number of coordination splits, UBL adds the following constraint to $\mathcal{S}$.

Limited Coordination Extraction: The expression g cannot contain more than N of the conjuncts that appear in any coordination in h. For example, with $N = 1$ the expression $g = \lambda x.city(x) \wedge major(x)$ could not be used in a solution to the splitting problem given the h above. UBL uses $N = 4$. While this constraint is not linguistically motivated, it is reasonable when parsing to human generated logical-forms. This is because it is unlikely that any system designer would create a logical language that requires more than 4 conjuncts to be model a single lexical item, regardless of whether or not the logical language was intended to model natural language.

$\mathcal{S}$ allows g to be built out of logical structures that are disconnected in h. In order to achieve this, Algorithm 3 allows any of the non-variable nodes in the logical tree headed by η to go in either g or h. If there are n non-variable nodes in this sub-tree, then there will be 2^n possible markings of these nodes. Furthermore, Algorithm 3 generates a split for all permutations of the λ-terms that are added at non-root positions in f and g. For k non-root λ-terms, this adds $k!$ splits. Both of these prohibitive additions to the complexity of $\mathcal{S}$ can be avoided through the following constraint.

Limited Substitution: The function f cannot contain new variables applied to any non-variable sub-expressions from h. For example, if $h = \lambda x.in(x, tex)$, the pair $f = \lambda q.q(tex)$ and $g = \lambda y \lambda x.in(x, y)$ is forbidden. This is equivalent to saying that all nodes under η in Algorithm 3 and Figure 3.3 must be marked to go in g avoiding the 2^n term above. It also prevents any non-root λ-terms from being added in f and g avoiding the $k!$ term above. Depending on the logical representation and view of the lexicon used, this may be over restrictive. For example, if the sentence 'I'll run' has its meaning modelled as $will(run(i))$ this restriction prevents the split that gives:

$$\text{I'll} \vdash \mathsf{S/(S\backslash NP)} : \lambda x.will(x(i))$$
$$\text{run} \vdash \mathsf{S\backslash NP} : \lambda y.run(y)$$

However, this type of split is not required in any of the domains on which UBL is run and, furthermore, the example above can be trivially (and more correctly) solved by splitting 'I'll' into two word units. A more serious restriction of limited application is its blocking of *type raising*. This is solved as a special case, for every (f, g) solution given by $\mathcal{S}$ with limited application, UBL adds a type raised alternative $(\lambda x.x(g), f)$. This amounts to the lexicalised type raising assumption of CCG.

Together, the constraints on semantic decomposition given in Section 3.2.1 and above guarantee that the number of splits is, in the worst case, an N-degree polynomial of the number of constants in h. The constraints used by UBL were designed to increase the efficiency of the splitting algorithm without impacting performance on development data drawn from the GeoQuery domain.

4.4.4 Assigning CCG Categories

Section 3.3 outlined the procedure of *syntactic category splitting* used in parse proposal. The category splitting procedure tells us that, given the original CCG category $\mathsf{X} : h = \mathsf{S\backslash NP} : \lambda x.in(x, tex)$, a mapping from logical type e to syntactic category NP, and the split of the logial expression h into $f = \lambda y \lambda x.in(x,y)$, and $g = tex$, we could produce the following two pairs of new categories:

$$(\mathsf{S\backslash NP/NP} : \lambda y \lambda x.in(x,y) \ , \ \mathsf{NP} : tex \ )$$
$$(\mathsf{NP} : tex \ , \ \mathsf{S\backslash NP\backslash NP} : \lambda y \lambda x.in(x,y) \ )$$

which were constructed by first choosing the syntactic category for g, in this case NP, and then enumerating the possible directions for the new slash in the category containing f.

The syntactic category for the CCG category containing g is proposed via a mapping `cat` from semantic type to allowable syntactic categories. For standard representations of CCG `cat` cannot describe a single syntactic category for every type. However, in order to reduce the search space of possible parses, UBL uses a definition of `cat` that proposes a single CCG syntactic category for each semantic type. This is done by the introduction of vertical slashes and the restriction of the atomic category set.

Vertical slashes are underspecified CCG slash operators in which the direction of application is not known. Vertical slashes may match either forward or backward slashes during function application:

$$\mathsf{S/(S/NP)} \quad \mathsf{S|NP} \Rightarrow \mathsf{S}$$
$$\mathsf{S/(S\backslash NP)} \quad \mathsf{S|NP} \Rightarrow \mathsf{S}$$

In order to avoid an explosion in the number of parses licensed by categories with vertical slashes, vertical slashes do not define syntactic functors of any directionality:

$$\mathsf{S|NP} \quad \mathsf{NP} \nRightarrow \mathsf{S}$$
$$\mathsf{NP} \quad \mathsf{S|NP} \nRightarrow \mathsf{S}$$

Vertical slashes are used exclusively by `cat`. Therefore, `cat` does not need to deal with ambiguity in slash direction.

The new syntactic category for g is determined based on its type T e.g. $\texttt{type}(tex) = e$ and $\texttt{type}(\lambda x.state(x))) = \langle e,t \rangle$. Then, the function $\texttt{cat}(T)$ takes an input type T and returns a single syntactic category as follows:

$$\texttt{cat}(T) = \begin{cases} \mathsf{NP} & \text{if } T = e \\ \mathsf{S} & \text{if } T = t \\ \texttt{cat}(T_2)|\texttt{cat}(T_1) & \text{when } T = \langle T_1, T_2 \rangle \end{cases} \tag{4.11}$$

The basic types e and t are assigned syntactic categories NP and S, and all functional types are assigned complex categories recursively. For example $\texttt{cat}(\langle e,t\rangle) = \mathsf{S|NP}$ and $\texttt{cat}(\langle e,\langle e,t\rangle\rangle) = \mathsf{S|NP|NP}$. This definition of CCG categories is unconventional in that it never assigns basic syntactic categories to functional types. There is no distinct syntactic category N for nouns (which have semantic type $\langle e,t\rangle$)[2]. Instead, the complex category S|NP is used. Along with the vertical slashes used for functional types, the constraint of basic syntactic categories to atomic semantic types allows cat to propose a single CCG category for every possible semantic type.

Once the syntactic category associated with g has been defined via Equation 4.11, two of the inverted CCG combinators in Figure 3.4 can be used to define the syntactic category for f (one forward and one backward). For each split (f, g) of the logical expression h in the CCG category $\mathsf{X} : h$ there are two CCG category splits $(\mathsf{A} : f\ ,\ \mathsf{B} : g)$ and $(\mathsf{B} : g\ ,\ \mathsf{C} : f)$.

4.4.5 Choosing Lexical Entries

Now, having described how UBL splits a single parse node I define the lexical generation function $\mathcal{T}'$ introduced in Equation 4.4.5 and repeated here:

$$t' \ = \ \mathcal{T}'(s, m, \Lambda, \theta)$$

And explain how this is used to expand the lexicon. Figure 4.5 illustrates the operation of $\mathcal{T}'$ on a single training instance (s_i, m_i). At this point $\mathcal{T}'$ has access to the previous PCCG with lexical items Λ_{i-1} and parameters θ_{i-1}. This is exemplified in Figure 4.3. In addition, $\mathcal{T}'$ has access to a function that assigns parameter weights to new lexical items—described later in Section 4.6.1.

$\mathcal{T}'$ first uses the current lexicon and parameter set to find the highest scoring parse t^* of (s_i, m_i). In the example this parse is illustrated in Figure 4.4. All new lexical

[2]If we had a more complex semantic typing system distinguishing e.g. predicates that describe events, then : would be able to distinguish between these categories as we will see in Section 6.4.

s_i : which states border texas

m_i : $\lambda x.state(x) \wedge next\_to(x, tex)$

Figure 4.2: Training pair

Λ_{i-1}	θ_{i-1}
which states border texas $\vdash$ S : $\lambda x.state(x) \wedge next\_to(x, tex)$	1.86
which states $\vdash$ S/(S|NP) : $\lambda f \lambda x.state(x) \wedge f(x)$	2.65
border texas $\vdash$ S|NP : $\lambda x.next\_to(x, tex)$	2.69
texas $\vdash$ NP : tex	10
$\vdots$	$\vdots$

Figure 4.3: Current PCCG

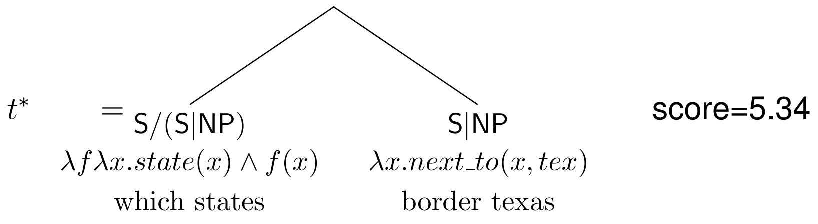

t^* =
S/(S|NP)
$\lambda f \lambda x.state(x) \wedge f(x)$
which states
S|NP
$\lambda x.next\_to(x, tex)$
border texas
score=5.34

Figure 4.4: Best parse t^*

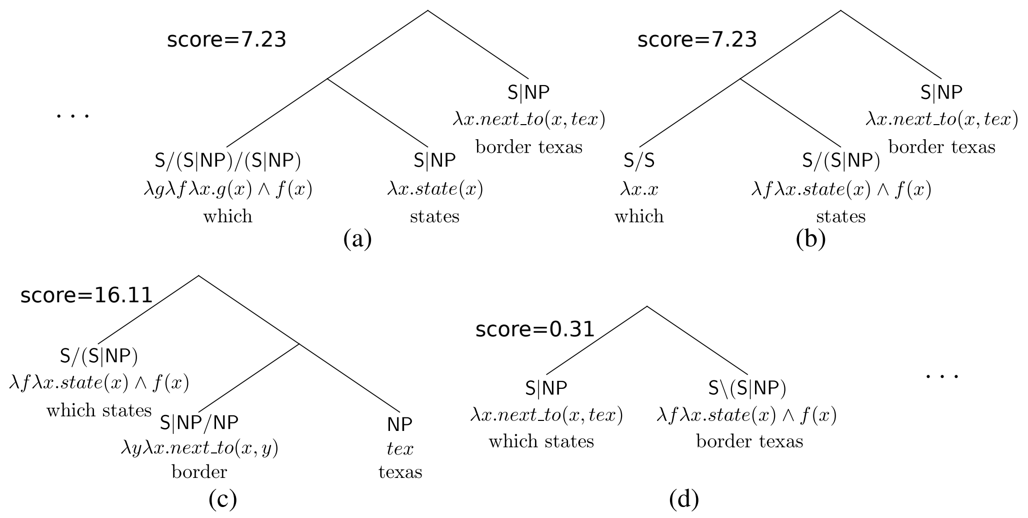

. . .
score=7.23
S/(S|NP)/(S|NP)
$\lambda g \lambda f \lambda x.g(x) \wedge f(x)$
which
S|NP
$\lambda x.state(x)$
states
S|NP
$\lambda x.next\_to(x, tex)$
border texas
(a)
score=7.23
S/S
$\lambda x.x$
which
S/(S|NP)
$\lambda f \lambda x.state(x) \wedge f(x)$
states
S|NP
$\lambda x.next\_to(x, tex)$
border texas
(b)
score=16.11
S/(S|NP)
$\lambda f \lambda x.state(x) \wedge f(x)$
which states
S|NP/NP
$\lambda y \lambda x.next\_to(x, y)$
border
NP
tex
texas
(c)
score=0.31
S|NP
$\lambda x.next\_to(x, tex)$
which states
S\(S|NP)
$\lambda f \lambda x.state(x) \wedge f(x)$
border texas
(d)
. . .

Figure 4.4: Parses Tried by $\mathcal{T}'$

Figure 4.5: $\mathcal{T}'$

items proposed by $\mathcal{T}'$ are proposed according to local manipulations to this parse tree. Nodes can be either *split* or *merged*. Merging nodes equates to replacing the sub-tree rooted at an internal parse node with a lexical entry containing the words covered by that sub-tree and the CCG category with which the internal parse node is labelled. The only internal parse node in the tree t^* in Figure 4.4 is the root node and the lexical item $\text{which states border texas} \vdash \mathsf{S} : \lambda x.state(x) \wedge next\_to(x, tex)$ created by merging the tree rooted at this node is already in the lexicon.

All nodes in t^* spanning more than one word may also be split. $\mathcal{T}'$ enumerates all pairs $(\mathsf{C_h} : m_h,\ w_{i:j})$, for $i + 1 < j$, where $\mathsf{C_h} : m_h$ is a category occurring at a node in y^* and $w_{i:j}$ are the (two or more) words it spans. For t^* in Figure 4.4 there are three such pairs: one for the root node $\mathsf{C_h} : m_h = \mathsf{S} : \lambda x.state(x) \wedge next\_to(x)$ and the phrase $w_{i:j} =$“what states border texas”, and one for each of the two leaf nodes. For each pair $(\mathsf{C_h} : m_h, w_{i:j})$, $\mathcal{T}'$ considers re-analysing this parse node by replacing it with a pair of new lexical items that are obtained by *splitting* $\mathsf{C_h} : m_h$ into $(\mathsf{C_l} : m_l, \mathsf{C_r} : m_r)$ as described above and splitting the wordspan in two. This replaces the sub-tree spanning $w_{i:j}$ rooted at with three node parse tree in which two lexical items $\mathrm{w_{i:k}} \vdash \mathsf{C_l} : m_l$ and $\mathrm{w_{k:j}} \vdash \mathsf{C_r} : m_r$ are combined.

All splits of each category $\mathsf{C_h} : m_h$ and word-span $w_{i:j}$ are considered. Figure 4.4 shows 4 new parses created by replacing the sub-tree rooted at a parse node in t' with a new split of that node. The parses in Figure 4.4(a)-(c) are created by splitting leaf nodes. The one in Figure 4.4(d) is created by re-splitting the root node. All of the new lexical items used in these parses are scored according to an initialisation function that collects corpus statistics and scores lexical items according to co-occurrence of words and logical constants. The parses considered by $\mathcal{T}'$ are scored according to their un-normalised log-likelihood. Only local features are introduced by a split or merge of a parse node. Therefore the score of each of the new trees can be calculated efficiently. Parses (a) and (b) have the same score due to the fact that they redistribute logical constants and words in the same way. All of the new parses in Figure 4.4 (a),(b) and (c) have a higher score than t^*. Parse (d) has a lower score. The parse in Figure 4.4(a) improves most upon t^* because it uses the high weighted lexical item $\text{texas} \vdash \mathsf{NP} : tex$. This parse is returned by $\mathcal{T}'$ as t' - the new best analysis of (s_i, m_i).

All of the lexical items in t' are read off the derivation with the function `lex`. For the illustrated t', $\texttt{lex}(t')$ returns the following lexical items:

$$\begin{aligned}\text{which states} &\vdash \mathsf{S/(S|NP)} : \lambda f \lambda x.state(x) \wedge f(x)\\ \text{border} &\vdash \mathsf{S|NP/NP} : \lambda y \lambda x.next\_to(x,y)\\ \text{texas} &\vdash \mathsf{NP} : tex\end{aligned}$$

and these are added to the Λ_{i-1} to give the new lexicon, Λ_i that has been updated on the basis of (s_i, m_i). There is only one new lexical item generated in the illustrated example: border $\vdash$ $\mathsf{S|NP/NP}$: $\lambda y \lambda x.next\_to(x,y)$. The parse generated by $\mathcal{T}'$ can contain at most two new lexical items: $|\Lambda_i \backslash \Lambda_{i-1}| \leq 2$. In many cases the parse returned by $\mathcal{T}'$ is the same as the maximum scoring parse under Λ_{i-1} ($t' = t^*$). In this case the contents of the lexicon are left unchanged.

4.5 The UBL Learning Algorithm

The UBL learning algorithm, illustrated in Algorithm 6, takes as input a dataset of (sentence, logical-expression) pairs $\{(s_i, m_i) : i = 1, \ldots, n\}$ and returns a PCCG containing a lexicon Λ and parameter vector θ of feature weights. UBL interleaves the lexical expansion step presented in Section 4.4 with the parameter update step presented in Section 4.3. Interleaving these two steps allows UBL to use the current state of the probabilistic parsing model to guide lexical expansion.

4.5.1 Two Step Learning Algorithm

Algorithm 6 presents the unification-based learning algorithm, UBL. This algorithm steps through the data incrementally and performs two steps for each training example. First, new lexical items are induced for the training instance by splitting and merging nodes in the best correct parse, given the current parameters. Next, the parameters of the PCCG are updated by making a stochastic gradient update on the marginal likelihood, given the updated lexicon.

Step 1: Updating the Lexicon Most of the work done by UBL is in incrementally updating the lexicon as described in Section 4.4. A single lexical update step is illustrated in lines 10-4.12. The function $\mathcal{T}'$ from Equation 4.8 generates a new highest scoring parse of (s_i, m_i) and any unseen lexical items in this parse are added to Λ. The parameter vector θ is extended to contain a parameter for each new lexical item.

```
Input  : Training set $\{(s_i, m_i) : i = 1, \ldots, n\}$ of sentences $s_i$ paired with
         logical expressions $m_i$. Set of NP lexical items $\Lambda_{\mathsf{NP}}$. Giza scores for
         all word-constant pairs. Function $\mathcal{T}'$. Function lex. Number of
         iterations $T$, learning rate parameter $\alpha_0$ and cooling rate parameter $c$.
Output: CCG lexicon $\Lambda$ and log-linear model parameter vector $\theta$.
begin
    Set $\Lambda = \{\mathrm{s_i} \vdash \mathsf{S} : m_i\}$ for all $i = 1, \ldots, n$;
    Set $\Lambda = \Lambda \cup \Lambda_{\mathsf{NP}}$;
    Initialise $\theta$ for contents of $\Lambda$ according to word-constant Giza scores;
    for $j = 0, \ldots, T-1$ do
        for $i = 1, \ldots, n$ do
            begin Expand Lexicon
                // Use $\mathcal{T}'$ to propose best parse
                $t' = \mathcal{T}'(s_i, m_i, \Lambda, \theta)$;
                // Add lexical items from $t'$ to $\Lambda$
                $\Lambda = \Lambda \cup \mathtt{lex}(t')$;
                Expand $\theta$ to contain entries for all $\Lambda$ using Eqn. 4.12;
            end
            begin Update Parameters
                // Calculate learning rate
                $k = 1 + j \times n$ ;
                $\eta_k = \frac{\alpha_0}{1 + c \times k}$;
                // Calculate gradient
                $\Delta = E_{p(t|s_i,m_i;\theta,\Lambda)}[\phi_j(t, s_i, m_i)] - E_{p(t,m|s_i;\theta,\Lambda)}[\phi_j(t, s_i, m_i)]$ ;
                // Update parameters
                $\theta = \theta + \eta_k \Delta$;
            end
        end
    end
end
```

Algorithm 6: UBL learning algorithm.

Step 2: Parameter Updates For each training example we update the parameters θ using the stochastic gradient updates given by Equation. 4.7 and the update schedule given by Equation 4.13.

Discussion The alternation between refining the lexicon and updating the parameters drives the learning process. The initial model assigns a conditional likelihood of one to each training example (there is a single lexical item for each sentence s_i, and it contains the labelled logical expression m_i). This initial model already achieves the highest possible negative log-likelihood (the global objective function given in Equation 4.5). Although the splitting step can only decrease or maintain the conditional likelihood of the data, the new entries it produces are less specific and should generalise better. Since UBL initially assigns positive weights to the parameters for new lexical items, the overall approach prefers splitting; trees with many lexical items will initially be much more likely. However, if the learned lexical items are used in too many incorrect parses, the stochastic gradient updates will down weight them to the point where the lexical induction step can merge or re-split nodes in the trees that contain them. This allows the approach to correct the lexicon and, hopefully, improve future performance.

The final solution learnt by UBL will, for any task with any ambiguity at the word level, achieve a lower conditional log-likelihood than the initial model. The online nature of the algorithm prefers lexical items that are commonly used in correct parses. Although the initial sentential lexical items can only ever be up-weighted (they are never used in a wrong parse), Zhang (2004) point out that stochastic gradient descent has "an implicit regularization scheme that is conveniently parameterized by the stopping point T". This implicit regularisation prevents UBL from settling upon the initial, overfit, solution. The stopping point T is chosen empirically on the basis of a development set. Observations of UBL running suggest that it is highly unlikely that it would ever revert back to the original solution in any achievable number of iterations.

4.6 Experimental Setup

4.6.1 Features and Initialisation

Each (syntactic derivation, logical-form, sentence) triple (t, m, s) triggers a set of features $\phi(t, m, s)$ that are used, along with the parameter vector θ to score this triple according to Equation 4.2. UBL uses two types of features: *lexical features* that score lexical items used in a parse and *semantic features* that score the logical-form m returned by a parse.

Each lexical item $l \in \Lambda$ has a single feature ϕ_l associated with it. This feature fires each time l is used in a parse. The weight $\theta_0[\phi_l]$ associated with this feature is initialised according to co-occurrence statistics between the words in l and the logical constants in l's logical-form, calculated prior to training. For example the lexical item

$$\text{highest} \vdash \mathsf{NP/(S|NP)} : \arg\max_x(\lambda y.f(y), \lambda z.height(z))$$

has the single word $\{\text{highest}\}$ and the two logical constants $\{\arg\max, height\}$. The initial parameter weight for this lexical item is calculated using the co-occurrence statistics estimated with the Giza++ (Och and Ney, 2003) implementation of IBM Model 1. For each word w, constant c pair the function $\texttt{Giza}(w, c)$ returns a score. These scores are used to calculate the initial parameter weight for a lexical item l as follows.

$$\theta_0[\phi_l] \;=\; 10 \times \frac{\sum_{w\in l}\sum_{c\in l} \; \texttt{Giza(w,c)}}{|\{w : w \in l\}| \times |\{c : c \in l\}|} \tag{4.12}$$

IBM model 1 returns a conditional probability $p(target|source)$. I choose the source to be the words and the target to be logical constants. This has the downside of returning zero scores for lexical items with no logical constants such as $\text{what} \vdash \mathsf{S/S} : \lambda x.x$. However, in our domain of interest there are more words with a single constant assignment than there are constant sets with a single word assignment (there are more synonyms than homonyms). Choosing the words as the source side therefore gives a peakier probability distribution. This helps UBL choose between lexical items when expanding the lexicon. UBL can run without this statistical initialisation of the lexical parameters. In Section 4.7 I present results with and without this initialisation.

Semantic features fire on the logical-form m returned by a parse. These features operate on predicate-argument relationships. Each time a predicate p in m takes an argument a with type $\texttt{type}(a)$ in argument slot i, a pair of features are fired: $\phi_{\langle p,a,i\rangle}$ for the predicate-argument relation and $\phi_{\langle p,\texttt{type}(a),i\rangle}$ for the predicate argument-type relation. For example, the logical-form

$$\lambda x.state(x) \wedge next\_to(x, tex)$$

fires the following 6 features. Two for each predicate-argument relation. Where v represents a variable and s represents the type of a state. The GeoQuery dataset has a type

Predicate-argument features	$: \{\phi_{\langle state,v,0\rangle},\ \phi_{\langle next\_to,v,0\rangle},\ \phi_{\langle next\_to,tex,1\rangle}\}$
Predicate argument-type features	$: \{\phi_{\langle state,s,0\rangle},\ \phi_{\langle next\_to,s,0\rangle},\ \phi_{\langle next\_to,s,1\rangle}\}$

hierarchy with 10 sub-categorisations of the entity type. Each predicate also has a set of allowed argument-type assignments. These are illustrated in Appendix A.2. The type of a variable is the most coarse grained type that is allowed in all of that variable's occurrences. In the logical-form above, the variable x is assigned type s - due to the constraints imposed by the predicate $state$. As well as the predicate-argument features above, for every variable v that occurs as an argument of a pair of predicates (p_1, p_2) in positions i_1 and i_2 respectively, UBL fires a feature $\phi_{\langle \mathtt{type}(v),p_1:i_1,p_2:i_2\rangle}$. For the example above UBL fires one of these: $\phi_{\langle s,state:0,next\_to:1\rangle}$. The parameter weights for all semantic features are initialised to zero.

The lexicon Λ is initialised by adding a single lexical item $s_i \vdash \mathsf{S} : m_i$ for each of the n training pairs in the dataset $\{(s_i, m_i) : i = 1, \ldots, n\}$. In addition, there is the option to add the entries of an NP list Λ_{NP}. This contains proper noun lexical items such as:

$$\text{texas} \vdash \mathsf{NP} : tex$$
$$\text{texas state} \vdash \mathsf{NP} : tex$$
$$\text{the state of texas} \vdash \mathsf{NP} : tex$$

which are automatically created from the set of entities in the logical language with a small set of hand generated templates. UBL does not require Λ_{NP} to operate but I include this initialisation to allow comparison to previous approaches. I evaluate UBL with and without this initialisation in Section 4.7. When Λ_{NP} is used, each $l_{\mathsf{NP}} \in \Lambda_{\mathsf{NP}}$ with word-string w_{NP} is assigned an initial parameter weight of $10 \times |w_{\mathsf{NP}}|$. This is equivalent to the highest weight that Equation 4.12 could return for any lexical entry with word-string w_{NP}.

The iterative learning algorithm illustrated in Algorithm 6 is run over the data T times. Parameter updates are scaled by the learning rate

$$\epsilon_k = \frac{\alpha_0}{1 + c \times k} \tag{4.13}$$

where k represents the number of parameter updates done so far. This learning rate satisfies the fundamental condition for convergence given by Kushner and Yin 1997, pp. 88.

$$\sum_{k=0}^{\infty} \epsilon_k = \infty, \;\; \epsilon_k \geq 0, \;\; \epsilon_k \to 0, \;\; \text{for } n \geq 0;$$
$$\epsilon_k = 0, \;\; \text{for } n < 0.$$

UBL used the learning rate $\alpha_0 = 1.0$ and cooling rate $c = 10^{-5}$ in all training scenarios, and ran the algorithm for $T = 20$ iterations. These values were selected with cross validation on the Geo880 development set.

4.6.2 Data and Evaluation

I evaluate UBL on the GeoQuery datasets, which contain natural-language queries of a geographical database paired with logical representations of each query's meaning. The full Geo880 dataset contains 880 (English-sentence, logical-form) pairs, which has been split into a development set of 600 pairs and a test set of 280 pairs, following Zettlemoyer and Collins (2005). The Geo250 dataset is a subset of Geo880 containing 250 sentences that have been translated into Turkish, Spanish and Japanese as well as the original English. Due to the small size of this dataset I use 10-fold cross validation for evaluation. This cross validation is performed using the same folds as Wong and Mooney (2006, 2007) and Lu et al. (2008), allowing a direct comparison.

The GeoQuery data is annotated with both lambda-expressions and FunQL meaning representations, which we have seen examples of throughout the paper. I report results for both representations, using the standard measures of *Recall* (percentage of test sentences assigned correct logical forms), *Precision* (percentage of logical forms returned that are correct) and *F1* (the harmonic mean of *Precision* and *Recall*). UBL has been optimised entirely for Recall. This is because recall represents, in this task, raw accuracy. There is no trivial solution that will achieve high Recall while Precision is, by definition, lower-bounded by Recall (UBL can predict at most one solution for each reference point in the test set).

4.6.3 Comparison Systems

I compare UBL to similar systems presented by Zettlemoyer and Collins (2005, 2007) - from now on ZC05 and ZC07 - that use CCG grammars but propose lexical items with language-specific templates rather than the language-independent function $\mathcal{T}$. I also compare UBL to the approaches of Kate and Mooney (2006) (KRISP) that uses a support vector machine to learn parses; Wong and Mooney (2006, 2007) (WASP and

λ-WASP) which use synchronous context free grammars; and Lu et al. (2008) that uses a generative model of both sentence and meaning. All of these approaches are tied to a single logical representation of meaning. All of them are also discussed in greater detail in Section 4.9.

4.6.4 Skipping Words

During testing it is not uncommon for the semantic parser to encounter unknown words or known words used in unknown contexts. In the case that the word is a function word that does not contribute semantic content it is sometimes possible to construct a parse of the sentence that returns the correct logical-form by *skipping* this word. In order to maximise the recall (raw accuracy) of UBL on the task of returning full sentential logical-forms I introduce a variant of UBL that is allowed to skip words with a fixed negative cost of -2.0. This is referred to as UBL-s.

4.7 Results and Discussion

Tables 4.1, 4.2, and 5.2 present the results for all of the experiments. In aggregate, they demonstrate that UBLlearns accurate models across languages and for both meaning representations. This is a new result; no previous system is as general.

We also see the expected tradeoff between precision and recall that comes from the inclusion of word skipping, which is labelled UBL-s. With the ability to skip words, UBL-s achieves the highest recall of all reported systems for all evaluation conditions. However, UBL achieves much higher precision and better overall F1 scores, which are generally comparable to the best performing systems.

The comparison to the CCG induction techniques of ZC05 and ZC07 (Table 5.2) is particularly striking. These approaches used language-specific templates to propose new lexical items and also required as input a set of hand-engineered lexical entries to model phenomena such as quantification and determiners. However, the use of higher-order unification allows UBL to achieve comparable performance while automatically inducing these types of entries.

For a more qualitative evaluation, Table 4.4 shows a selection of lexical items learned with high weights for the lambda-calculus meaning representations. Nouns such as “state” or “estado” are consistently learned across languages with the category $S|NP$, which stands in for the more conventional N. The algorithm also learns

System	English			Spanish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
WASP	70.0	**95.4**	80.8	72.4	91.2	81.0
Lu08	72.8	91.5	81.1	79.2	**95.2**	**86.5**
UBL	78.1	88.2	**82.7**	76.8	86.8	81.4
UBL-s	**80.4**	80.8	80.6	**79.7**	80.6	80.1
System	Japanese			Turkish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
WASP	74.4	**92.0**	**82.9**	62.4	**97.0**	75.9
Lu08	76.0	87.6	81.4	66.8	93.8	78.0
UBL	78.5	85.5	81.8	70.4	89.4	**78.6**
UBL-s	**80.5**	80.6	80.6	**74.2**	75.6	74.9

Table 4.1: Performance across languages on Geo250 with FunQL meaning representations.

System	English			Spanish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
λ-WASP	75.6	91.8	82.9	80.0	92.5	**85.8**
UBL	78.0	**93.2**	**84.7**	75.9	**93.4**	83.6
UBL-s	**81.8**	83.5	82.6	**81.4**	83.4	82.4
System	Japanese			Turkish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
λ-WASP	81.2	90.1	**85.8**	68.8	90.4	**78.1**
UBL	78.9	**90.9**	84.4	67.4	**93.4**	**78.1**
UBL-s	**83.0**	83.2	83.1	**71.8**	77.8	74.6

Table 4.2: Performance across languages on Geo250 with lambda-expression meaning representations.

language-specific constructions such as the Japanese case markers “no” and “wa”, which are treated as modifiers that do not add semantic content. Language-specific word order is also encoded, using the slash directions of the CCG categories. For example, “what” and “que” take their arguments to the right in the wh-initial English and Spanish. However, the Turkish wh-word “nelerdir” and the Japanese question marker “nan desu ka” are sentence final, and therefore take their arguments to the left.

There is less variation and complexity in the learned lexical items for the FunQL

System	FunQL			Lambda Expressions		
	Rec.	Pre.	F1	Rec.	Pre.	F1
Cross Validation Results						
KRISP	71.7	**93.3**	81.1	–	–	–
WASP	74.8	87.2	80.5	–	–	–
Lu08	81.5	89.3	**85.2**	–	–	–
λ-WASP	–	–	–	86.6	92.0	89.2
Independent Test Set						
ZC05	–	–	–	79.3	**96.3**	87.0
ZC07	–	–	–	86.1	91.6	88.8
UBL	81.4	89.4	**85.2**	85.0	94.1	**89.3**
UBL-s	**84.3**	85.2	84.7	**87.9**	88.5	88.2

Table 4.3: Performance on the Geo880 data set, with varied meaning representations.

representation. The fact that the meaning representation is deeply nested influences the form of the induced grammar. For example, recall that the sentence "what states border texas" would be paired with the meaning $answer(state(borders(tex)))$. For this representation, lexical items such as:

$$\text{what} \vdash \mathsf{S/NP} : \lambda x.answer(x)$$
$$\text{states} \vdash \mathsf{NP/NP} : \lambda x.state(x)$$
$$\text{border} \vdash \mathsf{NP/NP} : \lambda x.borders(x)$$
$$\text{texas} \vdash \mathsf{NP} : tex$$

can be used to construct the desired output. In practice, UBL often learns entries with only a single slash, like those above, varying only in the direction, as required for the language. Even the more complex items, such as those for quantifiers, are consistently simpler than those induced from the lambda-calculus meaning representations. For example, one of the most complex entries learned in the experiments for English is $\text{the smallest} \vdash \mathsf{NP\backslash NP/(NP|NP)} : \lambda f \lambda x.smallest\_one(f(x))$.

There are also differences in the aggregate statistics of the learned lexicons. For example, the average length of a learned lexical item for the (lambda-expression, FunQL') meaning representations is: (1.21,1.08) for Turkish, (1.34,1.19) for English, (1.43,1.25) for Spanish and (1.63,1.42) for Japanese. For both meaning representations

English
population of $\vdash \mathsf{NP/NP} : \lambda x.population(x)$
smallest $\vdash \mathsf{NP/(S\vert NP)} : \lambda f.arg\,min(y, f(y), size(y))$
what $\vdash \mathsf{S\vert NP/(S\vert NP)} : \lambda f \lambda x.f(x)$
border $\vdash \mathsf{S\vert NP/NP} : \lambda x \lambda y.next\_to(y, x)$
state $\vdash \mathsf{S\vert NP} : \lambda x.state(x)$
most $\vdash \mathsf{NP/(S\vert NP)\backslash(S\vert NP)\backslash(S\vert NP\vert NP)} : lambdaf \lambda g \lambda h \lambda x.argmax(y, g(y), count(z, f(z, y) \wedge h(z)))$

Japanese
no $\vdash \mathsf{NP\vert NP/(NP\vert NP)} : \lambda f \lambda x.f(x)$
shuu $\vdash \mathsf{S\vert NP} : \lambda x.state(x)$
nan desu ka $\vdash \mathsf{S\backslash NP\backslash(NP\vert NP)} : \lambda f \lambda x.f(x)$
wa $\vdash \mathsf{NP\vert NP\backslash(NP\vert NP)} : \lambda f \lambda x.f(x)$
ikutsu $\vdash \mathsf{NP\vert(S\vert NP)\backslash(S\vert NP\vert(S\vert NP))} : \lambda f \lambda g.count(x, f(g(x)))$
chiiki $\vdash \mathsf{NP\backslash NP} : \lambda x.area(x)$

Turkish
nedir $\vdash \mathsf{S\backslash NP\backslash(NP\vert NP)} : \lambda f \lambda x.f(x)$
sehir $\vdash \mathsf{S\vert NP} : \lambda x.city(x)$
nufus yogunlugu $\vdash \mathsf{NP\vert NP} : \lambda x.density(x)$
siniri$\vdash \mathsf{S\vert NP/NP} : \lambda x \lambda y.next_to(y, x)$
kac tane $\vdash \mathsf{S\backslash NP/(S\vert NP\vert NP)\backslash(S\vert NP)} : \lambda f \lambda g \lambda x.count(y, f(y) \wedge g(y, x))$
ya siniri $\vdash \mathsf{S\vert NP\backslash NP} : \lambda x \lambda y.next\_to(y, x)$

Spanish
en $\vdash \mathsf{S\vert NP/NP} : \lambda x \lambda y.loc(y, x)$
que es la $\vdash \mathsf{S/NP/(NP\vert NP)} : \lambda f \lambda x.f(x)$
pequena $\vdash \mathsf{NP\backslash(S\vert NP)\backslash(NP\vert NP)} :$ $\lambda g \lambda f.arg\,min(y, f(y), g(y))$
estado $\vdash \mathsf{S\vert NP} : \lambda x.state(x)$
mas $\vdash \mathsf{S\backslash(S\vert NP)/(S\vert NP)\backslash(NP\vert NP\vert(S\vert NP))} : \lambda f \lambda g \lambda h.argmax(x, h(x), f(g, x))$
mayores $\vdash \mathsf{S\vert NP\backslash(S\vert NP)} : \lambda f \lambda x.f(x) \wedge major(x)$

Table 4.4: Example learned lexical items for each language on the Geo250 lambda-expression data sets.

the model learns significantly more multiword lexical items for the somewhat analytic Japanese than the agglutinative Turkish. There are also variations in the average number of learned lexical items in the best parses during the final pass of training: 192 for Japanese, 206 for Spanish, 188 for English and 295 for Turkish. As compared to the other languages, the morphologically rich Turkish requires significantly more lexical variation to explain the data.

Finally, there are a number of cases where the UBL algorithm could be improved in future work. In cases where there are multiple allowable word orders, the UBL algorithm must learn individual entries for each possibility. For example, the following two categories are often learned with high weight for the Japanese word "chiisai":

$$\mathsf{NP}/(\mathsf{S}|\mathsf{NP})\backslash(\mathsf{NP}|\mathsf{NP}) : \lambda f \lambda g.argmin(x, g(x), f(x))$$
$$\mathsf{NP}|(\mathsf{S}|\mathsf{NP})/(\mathsf{NP}|\mathsf{NP}) : \lambda f \lambda g.argmin(x, g(x), f(x))$$

and are treated as distinct entries in the lexicon. Similarly, the approach presented here does not model morphology, and must repeatedly learn the correct categories for the Turkish words "nehri," "nehir," "nehirler," and "nehirlerin", all of which correspond to the logical form $\lambda x.river(x)$. Both of these limitations will be addressed further in Chapter 5.

4.8 Algorithm Analysis

The UBL learning algorithm performs a greedy search in the space of possible lexicons, guided by the current probabilistic parsing model. After 20 iterations over the single dataset the parameters of the parsing model have not converged but examination of the parser's performance on a development set shows that parse accuracy has. Choosing 20 as the number of training iterations is equivalent to choosing a level of regularisation according to Zhang (2004), who suggests that running SGD over the data multiple times can be regarded as a form of bias-variance trade-off that serves as an implicit method of regularisation that is parameterised by the number of iterations.

In order to make UBL's results comparable to those of the other systems I initialised the learning process with the NP list of entity names and a set of word-constant co-occurrence scores accumulated by running the Giza++ implementation of IBM model 1 over the training corpus. UBL can be trained without either of these and Figure 4.6 illustrates the performance of: UBL with both the NP list and the Giza initialisation; UBL-Giza with the NP list but no Giza initialisation; UBL-NP with the Giza initialisation but no NP list; and UBL-NP-Giza with neither the NP list nor the Giza initialisation. These results were collected through 10 fold cross validation on the 600 instance training set. Both the NP list and the Giza have a significant positive effect on accuracy of the model. Without either UBL does learn some of the easier constructions in the corpus but, since the lexical search is greedy, and since it is also initially driven by the Giza scores or high weighted NP lexical items, the unguided UBL-NP-Giza suffers far more from the huge size of the lexical search space.

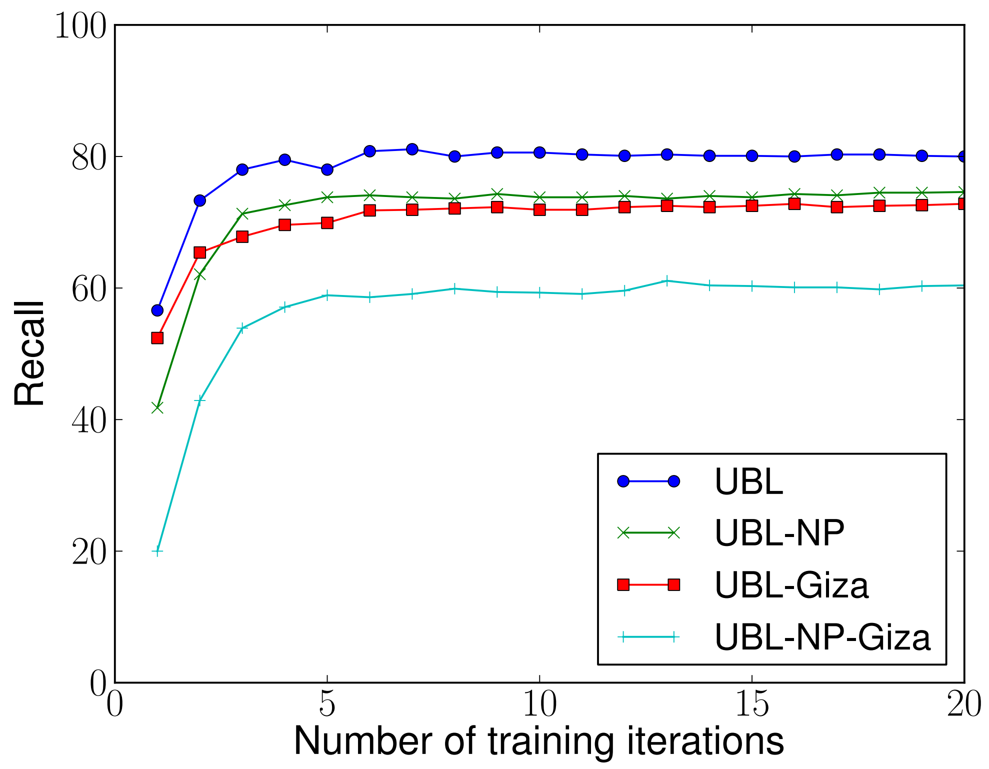


Figure 4.6: Ablation study of UBL initialisations

4.9 Related Work

Parsing sentences to compositional logical representations of their meaning was first attempted by Zelle and Mooney (1996) who presented a deterministic shift-reduce parser - CHILL. The rules of this parser are induced from a corpus of (sentence, logical-form) pairs with Inductive Logic Programming. The lexicon, however, is hand generated. Thompson and Mooney (1999) extended CHILL with a statistical lexical learner, removing the need for any human input over the annotation of sentences with logical-forms. However, in this system logical-forms are still composed by a deterministic parser from the set of lexical symbols.

Kate et al. (2005) present an approach to semantic parsing that leverages successful work in statistical syntactic parsing by mapping syntactic parse trees, returned by an existing syntactic parser, onto logical-forms. The rules used by this mapping are learnt from the rules allowed in the logical language from a corpus of (sentence, logical-form) pairs. Each of these rules is licensed by either a natural language phrase or a pattern in the syntactic parse tree. Once learnt, they are used deterministically. Ge and Mooney (2005) bring the benefits of statistical parsing to the task of parsing to semantics by training a statistical semantic parser (SCISSOR) on a corpus of semantic parse trees. In this corpus, not only the meaning of each sentence is given but also the way in which this meaning maps onto the sentence. Therefore, SCISSOR requires a significant amount of annotation over systems such as mine that learn just from (sentence, logical-form) pairs.

Zettlemoyer and Collins (2005) - from now on ZC05 - learn a statistical CCG parsing model from (sentence, logical-form) pairs. In this input, the correct parses are unknown and must be learnt from a set of allowed parses. Whereas the work presented in this thesis uses the CCG combinators in the function $\mathcal{T}$ to define a set of parses for any (sentence, logical-form) pair, ZC05 uses a small set of language specific templates. These templates match logical-form substructures. They encode information about semantic and syntactic lexical structure. Lexical items are generated for a (sentence, logical-form) pair by taking the cross product of all word strings with the set of all CCG categories generated by templates from the logical-form. As we have seen, this approach achieved good results with high precision in particular when tested on the GeoQuery domain. However, the inflexibility of the small set of lexical templates used means that the lexicons learnt by ZC05 could not achieve the same degree of parse coverage as UBL and for this reason UBL significantly outperforms ZC05 in exact match recall.

Kate and Mooney (2006) present KRISP - an alternative approach to learning a semantic parser from (sentence, logical-form) pairs. KRISP has access to the formal grammar that is used to build logical expressions in the domain's logical languages. Each production in this formal grammar is treated as a *semantic concept* and an SVM with a string similarity kernel is used to learn a correspondence between word-strings and semantic concepts. Wong and Mooney (2006) also learns a semantic parser from (sentence, meaning) pairs. This approach uses a synchronous context free grammar (SCFG) to simultaneously build a syntactic parse tree of the sentence and a FUNQL logical expression. This system is called WASP. The initial formulation of WASP can only support tree structured logical expressions. It cannot, therefore, model the equivalence of multiple instances of the same variable. Wong and Mooney (2007) add lambda-calculus variable binding operations to the rules of the SCFG to give λ-WASP which can learn Prolog logical expressions which are very similar in form to the lambda calculus representation used in this chapter. However, this approach only uses a very limited subset of the binding operations allowed by the lambda-calculus to deal with variable binding. It cannot therefore express the same range of languages as the grammar used by UBL which makes far more central use of the lambda-calculus ‘glue language’. Lu et al. (2008) define a tree structured generative model over sentences and FunQL meaning representations. This *hybrid tree* model is successful at parsing sentences to logical form however it is also very closely based upon the highly nested tree-structured FunQL meaning representation. The generative nature of the hybrid tree allow Lu and Ng (2011) to adapt it to the task of natural language generation from logical-forms.

As previously mentioned, the small set of lexical templates used by Zettlemoyer and Collins (2005) to generate lexical items could be overly restrictive in generating lexicons for semantic parsers, this became particularly obvious when parsing the harder Atis natural dialogue domain (of which we will see more in the next chapter). Zettlemoyer and Collins (2007) remedies this by relaxing the CCG combinatory rules to allow unconventional parses in which the syntactic context implicit in a lexical item’s slashes can be overridden. This work also introduces *semantic type shifting* operations that can morph lexical items according to a set of predefined transformations. These operations will be mentioned further in the next chapter where I present a system to induce abstractions over related lexical types.

All of the systems discussed so far assume the availability of a direct mapping from word-string to logical-form. However, in practice, conversations build meaning incrementally over multiple utterances. Later utterances build upon earlier ones and very often do not have a well formed independent meaning. Zettlemoyer and Collins (2009) solve this problem by explicitly modelling context sensitive references in utterance meanings and then resolving these references over adjacent utterances. This operation is not supported by any of the work that I present in this thesis. However, the operations of splitting logical-forms with higher-order unification could be applied across sentence boundaries to divide meanings built over multiple utterances as well as being used within a sentence to split word meanings. This has been suggested, but not implemented, by Dalrymple et al. (1991). Application of such an extension to UBL is left as future work.

Annotating sentences with meaning representations is time intensive. Various approaches to semantic parsing have been proposed that remove the need for this form of supervision. The approach presented by Clarke et al. (2010) use training corpora made up of (question, answer) pairs. Each logical-form returned by the parser in this approach can be run against a database to give an answer that is compared, during training, to a gold-standard annotation. This comparison gives a binary indication of success that is used for training a probabilistic semantic parser. As there are no logical annotations however, in order to ground this semantic parser in the database being used, Clarke et al. (2010) build an initial lexicon containing function words and then predict word-predicate mappings using a set of heuristics including word-string similarity and WordNet similarity metrics. The availability and utility of these with vary with the domain being modelled. They will not work across languages.

Liang et al. (2011) uses the same form of supervision during training but presents a new representation of meaning: *dependency-based compositional semantics* (DCS). DCS builds logical expressions as trees that parallel syntactic dependency trees. While this parallel eases learning of the DCS trees, logical components are often incorrectly scoped. The incorrect positioning of logical nodes in the DCS tree is resolved via a *mark-execute strategy* that is somewhat analogous to Cooper storage (Cooper, 1975). Liang et al. (2011) resolves the problem of grounding DCS trees in the target world through the use of *lexical triggers*. These return logical constants for input words. Triggers are hand generated for a small set of function words and all entity names. Other triggers are fired by a word's POS tag if that tag is in the set {JJ,NN,NNS}. Predicates for verbs and prepositions are introduced by *trace triggers* fired by all words.

Goldwasser et al. (2011) presents a system which further reduces the degree of semantic supervision required during training. This system trains a semantic parser with an objective function defined in terms of *confidence measures* designed to reward consistent looking logical expressions. The system has the same initial lexicon as Clarke et al. (2010) and, further, uses the output of a pre-existing syntactic dependency parser to define structural features for each sentence. Within the GeoQuery task with the FunQL meaning representations, this is enough to learn a semantic parser, albeit one that is significantly outperformed by UBL and the other systems reviewed here.

While none of these systems require the costly annotation of sentential logical-forms, they also do not learn the lexicon used for parsing. Instead they rely on either unreliable (and language dependent) heuristics or a predefined domain specific schema to map words onto logical structure. When choosing an approach for semantic parsing the trade off between lexicon definition and semantic annotation should therefore be taken into account. Approaches such as UBL that use semantic annotations but forgo lexicon definition may be particularly attractive in domains (such as dialogue systems) where one already has access to a significant amount of natural language paired with some interpretation of its meaning.

Recently, a number of approaches have started to address the problem of learning semantic parsers from naturally occurring (natural-language, world-state) pairings. Liang et al. (2009) learns the correspondence between an unsegmented text and a word state made up of records. There is no clear alignment between text and records other than the assumption that part of the text may describe a subset of the records. In this task it is necessary to learn the correct segmentation of the text into utterances, the choice of relevant records and alignment of the utterances to the relevant records. In related work, Chen and Mooney (2008); Chen et al. (2010) learn from human commentaries of a simulated football game paired with representations of of game state changes. The alignments between the commentaries and the state changes are unknown. These alignments are learnt and used for both semantic parsing and generation of synthetic commentaries from game logs. Artzi and Zettlemoyer (2011) learn a semantic parser from a dialogue system's conversational logs. Dialogue system statements are annotated with logical expressions representing knowledge already obtained by the system. The mapping between these logical expressions and the user's utterances is a weak one since it is not clear when or how the dialogue system obtains each piece of knowledge about the user's intent. Nonetheless, Artzi and Zettlemoyer (2011) show that by defining an utterance level loss function that takes the whole conversation as an input, they can learn an accurate semantic parser. Provided that the representations of world-state are compositional and typed, the approach presented in this chapter could theoretically be adapted to the task of learning a parser when the alignment between sentence and meaning are not clear. However, as the number of potential alignments increases, so does the number of possible derivations. In Chapter 6 we will see a training setting in which each sentence is paired with multiple potential interpretations.

Humans learn language in a world within which they can perform actions in response to some linguistic input and then recieve feedback in the form of an indication of success or failure, or a correction. Properly grounding the task of learning a semantic parser in a world within which actions can be made has been addressed by Branavan et al. (2010) and Chen and Mooney (2011a). Branavan et al. (2010) learn to map natural language sentences to sequences of executable actions in a computer help and a computer game domain. Both of these domains have some indication of an action's success and a reward function is defined in terms of this. This reward function is used by a reinforcement learning algorithm to train a semantic parser that maps sentences onto actions. Chen and Mooney (2011b) addresses the problem that concepts expressed in natural language do not always refer to observations of actions in the target world. Often the natural language sentence makes use of abstractions that do not exist in the world representation. Chen and Mooney (2011b) learn these abstractions by identifying logical substructures that reoccur with words in a training corpus. These commonly used substructures are used to refine the over-specified representations of all actions used to act in the world. to give a *refined plan* that describes the meaning of the sentence. Chen and Mooney (2011a) use this plan refinement in a system that learns a semantic parser by first parsing a sentence, executing the resultant plan and observing the subsequent success or failure.

Finally there has recently been some work in using the output of a syntactic parser to propose semantic structure. CCG syntactic analyses are highly semantically motivated. Bos et al. (2004) make use of the transparency between syntax and semantics in CCG to present a system for building ungrounded semantic representations from the output of the Clark and Curran (2004) CCG parser. These representations are expressed in first order predicate logic with a neo-Davidsonian view of events. Lexical items in this system are predefined for closed-class words. Lexical items are generated for open-class words by creating a predicate that matches the type of the CCG category with which the word is labelled and naming this predicate with the lemmatised word-form. As these predicates are merely named and not matched to any meaning-symbol in a world within which inference can be done, this system cannot be used to solve the same problems as the semantic parsers that UBL learns.

Poon and Domingos (2009) extract ungrounded predicate argument structures directly from text with dependency parses. They then cluster semantically interchangeable components that abstract away syntactic and lexical variations of the same meaning. Relationships between clusters represent semantic relationships expressed in the text. Poon and Domingos (2009) show that this ungrounded semantic structure can be used to solve a simple question answering task. They build an ontology that clusters semantically similar words and relationships from a corpus of biomedical abstracts and then query this ontology using queries generated from similar text. Poon and Domingos (2010) show that performance on this task can be improved by adding a hierarchical element to the clustering and Titov and Klementiev (2011) addresses the same task with a hierarchical non-parametric Bayesian model. Although this work starts to build an ontology of its own within which inference can be done, this ontology is still not grounded in any external representation of meaning and so these “unsupervised semantic parser induction” approaches do not and cannot address the same problems as the approach that I have presented in this chapter.

4.10 Conclusions

This chapter has presented a system of inducing probabilistic CCG grammars for semantic parsing from a set of (sentence, logical-form) pairs. I have introduced a constrained version of the higher-order unification type operations introduced in Chapter 3 and an underspecified syntactic representation that further limits the space of possible derivations. UBL incrementally expands a CCG lexicon while simultaneously training a log-linear probabilistic parsing model. This incremental approach allows UBL to search efficiently in the very large space of possible CCG lexicons.

As it makes very few assumptions about the natural and formal languages being modelled, the approach presented in this chapter provides a very general framework for semantic parser induction. Subsequently, it makes an important step towards black box systems for semantic parser induction that will be applicable across domains and languages. I have demonstrated UBL on a benchmark semantic parsing task and have shown that it can achieve state of the art results across two different compositional meaning representations and four different languages.

The UBL system presented in this chapter was demonstrated on the benchmark GeoQuery dataset. This is the most commonly used dataset in the semantic parsing literature and it provides a good representation of the well edited sentences that could be expected when querying a database. Despite the small size of the dataset (600 training sentences), due to the small size of the GeoQuery domain, these sentences still manage to cover the majority of the vocabulary and lexico-syntactic constructions that are used in querying the domain. In larger domains, however, this may not be the case. Also, when sentences are not typed in to a query window but spoken spontaneously, the variation in lexico-syntactic constructions is much larger. Traditional CCG lexicons such as the one used by UBL do not deal with the sparsity caused by this variation very well. The next chapter addresses this problem by introducing a new *factored* representation of the lexicon.

Chapter 5

Lexical Factorisation for Increased Coverage

This chapter presents work that was published in Kwiatkowski et al. (2011).

5.1 Introduction

The previous chapter described a language independent method for learning a CCG semantic parser. That approach used a traditional CCG lexicon in which each lexical item is a separately stored (word, syntactic-category, logical-expression) triple. An example subset of the CCG lexical items required to model the Atis flight-booking domain is:

(1) flight $\vdash$ $\mathsf{S|NP} : \lambda x.flight(x)$

(2) flight $\vdash$ $\mathsf{S|NP/(S|NP)} : \lambda f \lambda x.flight(x) \wedge f(x)$

(3) flight $\vdash$ $\mathsf{S|NP\backslash(S|NP)} : \lambda f \lambda x.flight(x) \wedge f(x)$

(4) ground transport $\vdash$ $\mathsf{S|NP} : \lambda x.trans(x)$

(5) ground transport $\vdash$ $\mathsf{(S|NP)/(S|NP)} : \lambda f \lambda x.trans(x) \wedge f(x)$

(6) ground transport $\vdash$ $\mathsf{S|NP\backslash(S|NP)} : \lambda f \lambda x.trans(x) \wedge f(x)$

(7) Boston $\vdash$ $\mathsf{NP} : bos$

(8) Boston $\vdash$ $\mathsf{S|NP/(S|NP)} : \lambda f \lambda x.from(x, bos) \wedge f(x)$

(9) New York $\vdash$ $\mathsf{NP} : nyc$

(10) New York $\vdash$ $\mathsf{S|NP/(S|NP)} : \lambda f \lambda x.from(x, nyc) \wedge f(x)$

In this list, the word "flight" is paired with the predicate *flight* in three separate lexical items which are required for different syntactic contexts. Item (1) has the standard

S|NP category for entries of this type. Recall that this category is used to represent nouns in the grammar introduced in Section 4.4. An example phrase that uses this lexical item is:

$$\begin{array}{cccc}
\text{flight} & \text{to} & \text{Boston} & \\
\mathsf{S|NP} & \mathsf{(S|NP)\backslash(S|NP)/NP} & \mathsf{NP} & \\
\lambda x.flight(x) & \lambda x\lambda f\lambda y.to(y,x)\wedge f(y) & \mathit{boston} & \\
\cline{2-3}
 & \multicolumn{2}{c}{\mathsf{S|NP\backslash(S|NP)}} & > \\
 & \multicolumn{2}{c}{\lambda f\lambda x.to(x,boston)\wedge f(x)} & \\
\cline{1-3}
\multicolumn{3}{c}{\mathsf{S|NP}} & > \\
\multicolumn{3}{c}{\lambda x.flight(x)\wedge to(x,boston)} &
\end{array}$$

Meanwhile, lexical item (2) is useful when we wish to chain nouns as in the phrase “flight tomorrow to Boston”, where the nouns “flight” and “tomorrow” are put together as follows:

$$\begin{array}{ccc}
\text{flight} & \text{tomorrow} & \\
\mathsf{S|NP/(S|NP)} & \mathsf{S|NP} & \\
\lambda f\lambda x.flight(x)\wedge f(x) & \lambda x tomorrow(x) & \\
\cline{1-2}
\multicolumn{2}{c}{\mathsf{S|NP}} & > \\
\multicolumn{2}{c}{\lambda x.flight(x)\wedge tomorrow(x)} &
\end{array}$$

And lexical item (3) is useful for phrases for unconventional word order such as “from Boston flight to New York”.

$$\begin{array}{cccc}
\text{from} & \text{Boston} & \text{flight} & \\
\mathsf{S|NP/NP} & \mathsf{NP} & \mathsf{S|NP\backslash(S|NP)} & \\
\cline{1-2}
\multicolumn{2}{c}{\mathsf{S|NP}} & > & \\
\multicolumn{2}{c}{\lambda x.from(x,bos)} & & \\
\cline{1-3}
\multicolumn{3}{c}{\mathsf{S|NP}} & < \\
\multicolumn{3}{c}{\lambda x.from(x,bos)\wedge flight(x)} &
\end{array}$$

Representing these three lexical items separately is inefficient, since each word of this class (such as “ground transport”) will require three similarly structured lexical entries differing only in predicate name (items (4) to (6)).

There may also be systematic semantic variation between entries for a certain class of words. For example, in lexical item (6) “Boston” is paired with the constant *bos* that represents its meaning. However, item (7) also adds the predicate *from* to the logical form. This might be used to analyse the somewhat elliptical sentence “Show me Boston flights to New York”. For this sentence (and the partial analysis below) the predicate *from* is required in the semantic analysis but has no explicit mention in the surface form of the sentence:

$$\begin{array}{cc}
\text{Boston} & \text{flights} \\
\mathsf{S|NP/(S|NP)} & \mathsf{S|NP} \\
\lambda f \lambda x.from(x, bos) \land f(x) & \lambda x.flight(x) \\
\hline
\multicolumn{2}{c}{\mathsf{S|NP}} \\
\multicolumn{2}{c}{\lambda x.from(x, bos) \land flight(x)}
\end{array} >$$

This type of construction is particularly prevalent in unedited utterances from natural language dialogue and has been previously identified as a particular challenge for semantic parsers by Zettlemoyer and Collins (2007).

The work presented in this chapter builds upon the insight that a large proportion of the variation between lexical items for a given class of words is systematic. Therefore it should be represented once and applied to a small set of basic lexical units. I develop a *factored lexicon* that captures this insight by distinguishing *lexemes*, which pair words with logical constants, from *lexical templates*, which map lexemes to full lexical items. As we will see, this can lead to a significantly more compact lexicon that can be learned from less data. For example, the first six lexical items from the list above can be represented with the two lexemes:

$$l_1 : \ (\text{flight}, \textit{flight})$$

$$l_2 : \ (\text{ground transport}, \textit{trans})$$

and three templates:

$$t_1 : \ \lambda(\omega, \vec{v}).[\, \omega \ \vdash \ \mathsf{S|NP} : \lambda x.v_1(x) \,]$$

$$t_2 : \ \lambda(\omega, \vec{v}).[\, \omega \ \vdash \ \mathsf{S|NP/(S|NP)} : \lambda f \lambda x.v_1(x) \land f(x) \,]$$

$$t_3 : \ \lambda(\omega, \vec{v}).[\, \omega \ \vdash \ \mathsf{S|NP\backslash(S|NP)} : \lambda f \lambda x.v_1(x) \land f(x) \,]$$

Each word or phrase is associated with one or a few lexemes which can be combined with templates drawn from a shared set to give lexical items. All contextual uses of new word of a known class can be learnt on the basis of a single observation of that word. The first time the learning algorithm sees the new phrase "round trip", provided that it can learn the lexeme (round trip, *round_trip*), it will be able to reconstruct the same range of lexical items as those for "flight" in the example lexicon.

This chapter presents an approach to learning and parsing with factored probabilistic CCG grammars for semantic parsing. The approach is closely based upon the UBL algorithm presented in Chapter 4. However, instead of constructing fully specified lexical items for the learned grammar, it automatically generates sets of lexemes and lexical templates from the training data. As the new approach is a direct extension of UBL to include a factored lexicon it is called the Factored Unification Based Learner (FUBL).

Section 5.2 describes the structure of the factored lexicon and how it is used to parse sentences to logical-forms. Section 5.3 and 5.4 then describes how a probabilistic factored CCG is induced from a training corpus of (sentence, logical-form) pairs using a training algorithm that is very similar in nature to the one presented in Chapter 4.

I evaluate the approach on the GeoQuery domain introduced in Chapter 4 as well as on the natural language dialogue domain Atis. In contrast to the edited queries in the GeoQuery dataset, the Atis data contains transcriptions of spontaneous utterances that occurred in natural language dialogues. These can be difficult to analyse with a rigid grammar representation due to their great syntactic variation and the fact that semantic content is often only implied.

The factored PCCG achieves at or near state-of-the-art recall across both domains and all languages. In particular, the new approach greatly improves upon UBL in the harder Atis domain despite the underlying similarity of the two algorithms. As well as helping deal with data sparsity in current semantic parsing tasks, by separating language-specific syntactic regularities from word-specific semantic information, the factored representation of the CCG lexicon has great potential for future extensions in domain adaptation and semi-supervised extension of semantic parsers. These potential extensions are discussed in Section 5.8.

5.2 Factored Lexicons

A factored lexicon includes a set L of lexemes and a set T of lexical templates. In this section, I formally define these sets, and describe how they are used to build CCG parses. I will use a set of lexical items from our running example to discuss the details of how the following lexical items:

(1) flight $\vdash$ $\mathsf{S|NP} : \lambda x.flight(x)$

(2) flight $\vdash$ $\mathsf{(S|NP)/(S|NP)} : \lambda f \lambda x.flight(x) \wedge f(x)$

$\ldots$

(7) Boston $\vdash$ $\mathsf{NP} : bos$

(8) Boston $\vdash$ $\mathsf{S|NP\backslash(S|NP)} : \lambda f \lambda x.from(x, bos) \wedge f(x)$

are constructed from specific lexemes and templates.

5.2.1 Lexemes

A lexeme $(\mathrm{w}, \vec{c})$ pairs a word sequence w with an ordered list of typed logical constants $\vec{c} = [c_1 \ldots c_m]$. For example, item (1) and (2) above would come from a single lexeme $(\text{flight}, [flight_{\langle e,t \rangle}])$. Similar lexemes would be represented for other predicates, e.g. $(\text{ground transport}, [trans_{\langle e,t \rangle}])$; for functions e.g. $(\text{fare}, [cost_{\langle e,i \rangle}])$; for quantifiers e.g. $(\text{all}, [\forall])$; and for entity identifiers e.g. $(\text{New York}, [nyc_e])$.

Lexemes can also contain multiple constants, for example:

$$\begin{aligned} &(\text{cheapest}, [argmin_{\langle\langle e,t,\rangle,\langle e,i\rangle,e\rangle}, cost_{\langle e,i\rangle}]) \\ \text{and} \quad &(\text{shortest}, [argmin_{\langle\langle e,t,\rangle,\langle e,i\rangle,e\rangle}, time_{\langle e,i\rangle}]) \end{aligned}$$

which we will see more examples of later. While all constants used in lexemes are typed, in the descriptions used from hereon the type signatures may be omitted in the interests of a compact notation.

5.2.2 Lexical Templates

A lexical template takes a lexeme and produces a lexical item. Templates have the general form:

$$\lambda(\omega, \vec{v}).[\omega \vdash \mathsf{X} : h_{\vec{v}}]$$

where $h_{\vec{v}}$ is a logical expression that contains variables from the list $\vec{v}$. Applying this template to the input lexeme $(\mathrm{w}, \vec{c})$ gives the full lexical item $\mathrm{w} \vdash \mathsf{X} : h$ where the variable ω has been replaced with the wordspan w and the logical form h has been created by replacing each of the variables in $\vec{v}$ with the counterpart constant from

$\vec{c}$. For example, the lexical item (7) above would be constructed from the lexeme $(\text{Boston}, [bos])$ using the template $\lambda(\omega,\vec{v}).[\omega \vdash \mathsf{NP} : v_1]$:

$$\lambda(\omega,\vec{v}).[\omega \vdash \mathsf{NP} : v_1] \quad (\text{Boston}, [bos_e]) \quad \Rightarrow \quad \text{Boston} \vdash \mathsf{NP} : bos_e$$

And items (1) and (2) would both be constructed from the single lexeme $(\text{flight}, [flight])$ with the two different templates: $\lambda(\omega,\vec{v}).[\omega \vdash \mathsf{S|NP} : \lambda x.v_1(x)]$ and $\lambda(\omega,\vec{v}).[\omega \vdash \mathsf{S|NP/(S|NP)} : \lambda f \lambda x.v_1(x) \wedge f(x)]$:

$$\lambda(\omega,\vec{v}).[\omega \vdash \mathsf{S|NP} : \lambda x.v_1(x)] \;\; (\text{flight}, [flight]) \;\Rightarrow\; \text{flight} \vdash \mathsf{S|NP} : \lambda x.flight(x)$$
$$\lambda(\omega,\vec{v}).[\omega \vdash \mathsf{S|NP/(S|NP)} : \lambda f \lambda x.v_1(x) \wedge f(x)] \;\; (\text{flight}, [flight])$$
$$\Rightarrow \; \text{flight} \vdash \mathsf{S|NP/(S|NP)} : \lambda f \lambda x.flight(x) \wedge f(x)$$

5.2.3 Parsing with a Factored Lexicon

In general, there can by many different (lexeme, template) pairs that produce the same lexical item. For example, lexical item (7) in our running example above can be constructed from the lexemes $(\text{Boston}, [bos])$ and $(\text{Boston}, [from, bos])$, given appropriate templates.

$$\lambda(\omega,\vec{v}).[\omega \vdash \mathsf{S|NP\backslash(S|NP)} : \lambda f \lambda x.from(x, v_1) \wedge f(x)] \;\; (\text{Boston}, [bos]) \;\Rightarrow$$
$$\text{Boston} \vdash \mathsf{S|NP\backslash(S|NP)} : \lambda x.from(x, bos) \wedge f(x)$$
$$\lambda(\omega,\vec{v}).[\omega \vdash \mathsf{S|NP/(S|NP)} : \lambda f \lambda x \wedge v_1(x, v_2) \wedge f(x)] \;\; (\text{Boston}, [from, bos]) \Rightarrow$$
$$\text{Boston} \vdash \mathsf{S|NP/(S|NP)} : \lambda f \lambda x.from(x, bos) \wedge f(x)$$

To model this ambiguity, the selection of a (lexeme, template) pair is included as a decision to be made while constructing a CCG parse tree. Given the lexical item produced by the chosen lexeme and template, parsing continues as normal with the CCG combinators. This direct integration allows for features that signal which lexemes and templates have been used while also allowing for well defined marginal probabilities, by summing over all ways of deriving a specific lexical item.

5.3 Learning Factored Lexicons

Lexemes and templates are added to the factored lexicon by one of two procedures. One that factors lexical items into a single (lexeme, template) pair and one that generates lexical templates from internal nodes of high scoring parse trees. Both of these procedures are presented below. Section 5.4 will then describe how the factoring operations are integrated into the complete learning algorithm.

5.3.1 Maximal Factorings

Given a lexical item l of the form $\mathrm{w} \ \vdash X : h$ with words w, a syntactic category X, and a logical form h, I defines the *maximal factoring* to be the unique (lexeme, template) pair that can be used to reconstruct l and which also places all of the constants of h in the lexeme (listed in a fixed order based on an ordered tree traversal of h).

For example, the maximal factoring for the lexical item $\mathrm{Boston} \ \vdash \ \mathsf{NP} : bos$ is the pair we saw before: $(\mathrm{Boston}, [bos])$ and $\lambda(\omega, \vec{v}).[\omega \vdash \mathsf{NP} : v_1]$. Similarly, the lexical item $\mathrm{Boston} \ \vdash \ \mathsf{S|NP\backslash(S|NP)} : \lambda f.\lambda x.f(x) \wedge from(x, bos)$ would be factored to produce $(\mathrm{Boston}, [from, bos])$ and $\lambda(\omega, \vec{v}).[\omega \vdash \mathsf{S|NP\backslash(S|NP)} : \lambda f.\lambda x.f(x) \wedge v_1(x, v_2)]$.

As we will see in Section 5.4, this notion of factoring can be directly incorporated into the existing UBL algorithm. When the UBL would have added a lexical entry l to its CCG lexicon, FUBL can instead compute the factoring of l and add the corresponding lexeme and template to the factored lexicon.

5.3.2 Introducing Templates with Content

Maximal factorings, as just described, provide for significant lexical generalisation but do not handle all of the cases needed to learn effectively. For instance, the maximal split for the item $\mathrm{Boston} \ \vdash \ \mathsf{S|NP\backslash(S|NP)} : \lambda f.\lambda x.f(x) \wedge from(x, bos)$ would introduce the lexeme $(\mathrm{Boston}, [from, bos])$, which is suboptimal since each possible city would need a lexeme of this type, with the additional *from* constant included. Instead, we would ideally like to learn the lexeme $(\mathrm{Boston}, [bos])$ and have a template that introduces the *from* constant. This would model the desired generalisation with a single lexeme per city.

In order to permit the introduction of extra constants into lexical items, templates that contain logical constants are created through *partial factorings*. For instance, the template below can introduce the predicate *from*:

$$\lambda(\omega, \vec{v}).[\omega \vdash \mathsf{S|NP\backslash(S|NP)} : \lambda f.\lambda x.f(x) \wedge from(x, v_1)]$$

The use of templates to introduce extra semantic constants into a lexical item is similar to, but more general than, the English-specific *type-shifting* rules used in Zettlemoyer and Collins (2007), which were introduced to model spontaneous, unedited text. They are useful, as we will see, in learning to recover semantic content that is implied, but not explicitly stated, as in the original motivating phrase “flights Boston to New York.”

Lexical templates that introduce semantic content are generated from internal nodes of parse trees. This procedure builds on the intuition that they will be used to recover semantic content that is missing because of missing word, as in the example above. In this scenario, there should also be other sentences that actually include the word, in our example this would be something like “flights from Boston.” Provided FUBL has learnt the correct lexemes and templates, it should be able to produce the following correct parse:

$$\begin{array}{ccc}
\text{flights} & \text{from} & \text{Boston} \\
\mathsf{S|NP} & \mathsf{S|NP\backslash(S|NP)/NP} & \mathsf{NP} \\
\lambda x.flight(x) & \lambda y\lambda f\lambda x.f(x)\wedge from(x,y) & bos \\
\cline{2-3}
 & \multicolumn{2}{c}{\mathsf{S|NP\backslash(S|NP)} \quad >} \\
 & \multicolumn{2}{c}{\lambda f\lambda x.f(x)\wedge from(x,bos)} \\
\hline
\multicolumn{3}{c}{\mathsf{S|NP} \quad <} \\
\multicolumn{3}{c}{\lambda x.flight(x)\wedge from(x,bos)}
\end{array}$$

Given analyses of this form, FUBL introduces new templates that allows it to recover from missing words, for example if “from” was dropped from the sentence. FUBL identifies nodes that have two lexicalised children in the best parse trees found during training and introduce templates that can produce the non-terminal, even if one of the words is missing. In the example above, there is a single parse node with two lexicalised children: the non-terminal spanning “from Boston,”. Here, a partial factoring would introduce the desired template $\lambda(\omega,\vec{v}).[\omega \vdash \mathsf{S|NP\backslash(S|NP)} : \lambda f.\lambda x.f(x) \wedge from(x,v_1)]$ for mapping the lexeme $(\text{Boston},[bos])$ directly to the intermediate structure. A different partial factoring would also introduce the template $\lambda(\omega,\vec{v}).[\omega \vdash \mathsf{S|NP\backslash(S|NP)} : \lambda f.\lambda x.f(x) \wedge v_1(x,bos)]$ that maps the lexeme $(\text{flight},[flight])$ onto the chosen internal node.

Not all templates introduced this way will model valid generalisations. They are therefore incorporated into FUBL with an indicator feature with an associated parameter that controls their use.

5.3.3 Word-Stem Lexemes

When dealing with small amounts of training data in morphologically rich languages, parsers often suffer from problems of data sparsity. The Turkish Geo250 dataset contains four words “nehri,” “nehir,” “nehirler,” and “nehirlerin” corresponding to the predicate $\lambda x.river(x)$. In order to avoid having to store separate lexemes for all of these words we allow the introduction of lexemes that contain, instead of a word sequence, a word stem. At parse time, words are matched to these stem lexemes through the removal of a preffix and/or suffix. For example three of the four Turkish words above share the stem “nehir”. At parse time, the lexemes containing the word-string “nehir” will be returned for any of “nehir”, “nehirler” or ‘nehirlerin”. These lexemes are used with lexical templates in a parse as before. In order to constrain the overgeneration of parses in this way I restrict the set of stems and suffixes allowed to the set proposed by the unsupervised morphological analyser Morfessor (Creutz and Lagus, 2007) when run over the sentences in the training corpus. I also assign a feature to each prefix, suffix, (prefix,stem) pair and (suffix,stem) pair in order to control the use of these morphological mappings. The mapping from word to allowed stems is done with the function `stem`.

5.4 Learning Factored PCCGs

The Factored Unification Based Learning (FUBL) method extends the UBL algorithm to induce factored lexicons, while also simultaneously estimating the parameters of a log-linear CCG parsing model.

Algorithm 7 shows the FUBL learning algorithm. Like UBL it is trained on training data $\{(s_i, m_i) : i = 1 \ldots N\}$ where each example is a sentence s_i paired with a logical form m_i. The algorithm induces a factored PCCG, including the lexemes L, templates T, and parameters θ.

The algorithm is online, repeatedly performing both lexical expansion and a parameter update for each training example. The first step uses the function $\mathcal{T}'$ defined in Chapter 4 and is similar to the lexical expansion step in Algorithm 6 but includes extensions for updating the factored lexicon, as discussed in Section 5.3.

Lexical Initialisation The model is initialised with a factored lexicon as follows. MAX-FAC is a function that takes a lexical item l and returns the maximal factoring

```
Input  : Training set $\{(s_i, m_i) : i = 1, \ldots, n\}$ of sentences $s_i$ paired with logical
         expressions $m_i$. Set of entity lexemes $L_e$. Giza scores for all word-constant
         pairs. Function $\mathcal{T}'$. Function lex. Function MAX-FAC. Function
         PART-FAC. Function stem. Number of iterations $T$, learning rate parameter
         $\alpha_0$ and cooling rate parameter $c$.
Output: Lexemes $L$, Lexical Templates $T$ and log-linear model parameter vector $\theta$.
1  begin
2      for $i = 0, \ldots, n$ do
3          $(\phi, \pi) = \texttt{MAX-FAC}(\mathrm{s_i} \vdash \mathsf{S} : m_i)$;
4          $L = L \cup \{\phi\},\ T = T \cup \{\pi\}$ ;
5      end
6      $L = L \cup L_e$;
7      Initialise $\theta$ for contents of $L$ according to word-constant Giza scores;
8      for $j = 0, \ldots, T-1$ do
9          for $i = 1, \ldots, n$ do
10             begin Expand Lexicon
                   // Use $\mathcal{T}'$ to propose best parse
11                 $t' = \mathcal{T}'(s_i, m_i, L, T, \theta)$;
                   // Add maximal factorings from $t'$
12                 for $l \in \texttt{lex}(t')$ do
13                     $(\phi, \pi) = \texttt{MAX-FAC}(l)$;
14                     $L = L \cup \{\phi\},\ T = T \cup \{\pi\}$ ;
15                     $L = L \cup \{\texttt{stem}(\phi)\}$ ;
16                 end
17                 Expand $\theta$ to contain entries for all $L$ using Eqn. 4.12;
                   // Add factorings with constants from $t'$
18                 $T = T \cup \texttt{PART-FAC}(t')$;
19             end
20             begin Update Parameters
                   // Exactly the same as in Algorithm 6
21             end
22         end
23     end
24 end
```

Algorithm 7: FUBL learning algorithm

of it, that is the unique, maximal (lexeme, template) pair that can be combined to construct l, as described in Section 5.3.1. FUBL applies MAX-FAC to each of the training examples (s_i, m_i), creating a single way of producing the desired meaning m_i from a lexeme containing all of the words in s_i. The lexemes and templates created in this way provide the initial factored lexicon. The lexicon is also initialised with a single lexeme for each entity name such as $(\text{New York}, ny)$.

Factored Lexical Expansion The lexical expansion step of the learning algorithm in Figure 7 adds lexemes and templates to the factored lexicon by performing manipulations on the highest scoring correct parse of the current training example (s_i, m_i). First the $\mathcal{T}'$ procedure is run as in Algorithm 6 to generate a new best parse of the training instance. Then FUBL uses the function MAX-FAC to create the maximal factorings of each of the lexical items used in this parse as described in Section 5.3 and these are added to the factored representation of the lexicon.

New templates can also be introduced through partial factorings of internal parse nodes as described in Section 5.3.2. These templates are generated by using the function PART-FAC to abstract over the wordspan and a subset of the constants contained in the internal parse nodes of t'. This step allows for templates that introduce new semantic content to model elliptical language, as described in Section 5.3.2.

Parameter Updates FUBL learns the parameters of the log-linear parsing model in the same way as UBL - with the stochastic gradient descent update discussed in Section 4.3.

5.5 Experimental setup

Data Sets I evaluate FUBL on the GeoQuery dataset introduced in Section 4.6 as well as on the logically annotated Atis dataset created by Zettlemoyer and Collins (2007) based on the Atis data used by He and Young (2006). This Atis data contains 5410 (sentence, logical-form) pairs where the sentences are natural language queries to a flight booking system. These are split into a 4480 pair training set, a 480 pair development set and a 450 pair test set.

Evaluation Metrics I report exact match *Recall* (percentage of sentences for which the correct logical-form was returned), *Precision* (percentage of returned logical-forms that are correct) and *F1* (harmonic mean of Precision and Recall). For Atis I also report partial match statistics calculated according to the logical literals contained within the logical form following Zettlemoyer and Collins (2007). Each literal describes some property of the logical-form. For example the following prediction does not match the

$$\begin{aligned}\text{Reference}:&\ \lambda x.flight(x) \wedge time(x, 819) \wedge from(x, dv) \wedge to(x, sf)\\ \text{Prediction}:&\ \lambda x.flight(x) \wedge flight\_no(x, 819) \wedge from(x, dv)\end{aligned}$$

reference. However, there is significant overlap between the two logical forms which can be computed by comparing literals:

Reference Literals	Prediction Literals	
$flight(x)$	$flight(x)$	✓
$flight\_no(x, 819)$	...	✗
...	$time(x, 819)$	✗
$from(x, dv)$	$from(x, dv)$	✓
$to(x, sf)$	...	✗
2/4	2/3	

I report partial match Recall (percentage of correct literals returned), Precision (percentage of returned literals that are correct) and F1 score. For the example illustrated the Recall is $2/4 = 50\%$, the Precision is $2/3 = 67\%$ and F1 is $\frac{2\times2/4\times2/3}{2/4+2/3} = 57\%$.

Features Like UBL, FUBL has *lexical features* and *logical-form features*. Lexical features fire on the lexemes and templates used to build the lexical items used in a parse. For each (lexeme,template) pair used to create a lexical item we have indicator features ϕ_l for the lexeme used, ϕ_t for the template used, and $\phi_{(l,t)}$ for the pair that was used. I assign the features on lexical templates a weight of 0.1 to prevent them from swamping the far less frequent but equally informative lexeme features. When morphological transformations are used, each non-empty suffix or prefix fires a feature, and the combination of suffix, prefix and stem fires a feature.

Parameter Initialisation The weights for lexeme features are initialized according to coocurrance statistics between words and logical constants caluclated using IBM Model 1 as described in Equation 4.12. All initial entity name lexemes are given the same weight 10 as the initial NP list in UBL. The initial weights for templates are set by adding -0.1 for each slash in the syntactic category and -2 if the template contains logical constants. Features on lexeme-template pairs and all parse features are initialized to zero. All weights for morphological features are initialized to -1.

Comparison Systems I compare the performance of FUBL to all recently published, directly-comparable results. For GeoQuery, this includes the ZC05, ZC07 (Zettlemoyer and Collins, 2005, 2007), λ-WASP (Wong and Mooney, 2007), UBL systems and DCS (Liang et al., 2011). For Atis, I report results from ZC07, and UBL as well as from HY06 (He and Young, 2006) which does not return compositional logical-forms but does return nested concept labels that are closely related to the logical literals used in the partial match metrics introduced above.

Partial Factorings and Morphology Both the lexical templates introduced from partial factorings and the morphological transformations on lexemes add significant generative power to the lexical representation. In sparse training scenarios this will be beneficial but, since both of these mechanisms significantly increase the search space of possible parses, they may also have a detrimental effect. Subsequently, I treat these procedures as options in FUBL and report results with and without both of them. I also introduce a new option FUBL+pf* which introduces lexical templates from partial factorings in a post-processing step. After training the model, FUBL+pf* runs over the training corpus once and performs partial factorings on the best correct parses of all training pairs. Templates generated in this way are added to the set of lexical templates with weight -2.

5.6 Results

Tables 5.1 to 5.4 present the performance of FUBL in all testing scenarios. In Table 5.1 results are presented for FUBL run on Geo250 with morphological transformations on lexemes (FUBL+m) and without. These results should be interpreted with care since a single percentage point corresponds to 2-3 sentences. However, FUBL does achieve higher recall than any previous systems across the board. Adding morphological transformations increases recall for Turkish and Spanish but decreases recall for English and Japanese with the greatest effect on the morphologically rich Turkish. There seems to be a trade off here between the benefits of a more generative lexicon and the drawback of a larger search space. This is addressed in greater detail later in Section 5.7.

System	English			Spanish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
λ-WASP	75.6	**91.8**	82.9	80.0	**92.5**	**85.8**
UBL	81.8	83.5	82.6	81.4	83.4	82.4
FUBL	**83.7**	83.7	**83.7**	85.6	85.8	85.7
FUBL+m	82.9	82.9	82.9	**86.3**	86.6	86.4
System	**Japanese**			**Turkish**		
	Rec.	Pre.	F1	Rec.	Pre.	F1
λ-WASP	81.2	**90.1**	**85.8**	68.8	**90.4**	**78.1**
UBL	83.0	83.2	83.1	71.8	77.8	74.6
FUBL	**83.2**	83.8	83.5	72.5	73.7	73.1
FUBL+m	82.8	83.2	83.0	**74.0**	74.9	74.4

Table 5.1: Exact-match accuracy on the Geo250 data set.

Table 5.2 shows the results for the various systems run on the Geo880 dataset. In this scenario, FUBL does not use morphological transformations on lexemes. The only higher recall on Geo880 is achieved by DCS with prototypes - which uses significant English-specific resources, including manually specified lexical content, but does not require training sentences annotated with logical-forms. Again, the difference in recall between systems is marginal, with each percentage point relating to 2.8 sentences. Therefore, FUBL achieves state of the art recall and, crucially, the additional generative power of the factored lexicon has not hurt performance when compared to the

System	Rec.	Pre.	F1
Labelled Logical Forms			
ZC05	79.3	**96.3**	87.0
ZC07	86.1	91.6	**88.8**
UBL	87.9	88.5	88.2
FUBL	**88.6**	88.6	88.6
Labelled Question Answers			
DCS	**91.1**	-	-

Table 5.2: Exact match accuracy on the Geo880 test set.

closely related UBL system.

The move from the traditional CCG lexicon of UBL to the factored representation used by FUBL has not hurt performance on the benchmark GeoQuery datasets. The change has not significantly increased performance either. However, the factored lexicon was not designed to deal with the well edited sentences in the GeoQuery domain, it was designed to deal with the greater sparsity resulting from the great variation in syntactic and semantic constructions seen in spontaneous utterances of the sort contained in the Atis dataset.

System	Exact Match		
	Rec.	Pre.	F1
ZC07	74.4	**87.3**	80.4
UBL	65.6	67.1	66.3
FUBL	77.1	77.2	77.1
FUBL-pf	80.2	80.2	80.2
FUBL+pf*	**81.9**	82.1	**82.0**

Table 5.3: Performance on the Atis development set.

System	Exact Match			Partial Match		
	Rec.	Pre.	F1.	Rec.	Pre.	F1
ZC07	**84.6**	**85.8**	**85.2**	**96.7**	**95.1**	**95.9**
HY06	-	-	-	-	-	90.3
UBL	71.4	72.1	71.7	78.2	98.2	87.1
FUBL+pf*	82.8	82.8	82.8	95.2	93.6	94.6

Table 5.4: Performance on the Atis test set.

On the Atis development set, FUBL+pf* outperforms ZC07 by 7.5% of recall, FUBL-pf outperforms ZC07 by 6.3% of recall, and FUBL outperforms ZC07 by 2.7% of recall. On the test set however FUBL+pf* lags ZC07 by 2%. The reasons for this discrepancy are not clear, however, it is possible that the syntactic constructions found in the Atis test set do not exhibit the same degree of variation as those seen in the development set. This would negate the need for the very general lexicon learnt by FUBL.

The most important difference in performance illustrated in Tables 5.3 and 5.4 are the differences between FUBL and UBL which is almost exactly the same but cannot generate new lexical items at parse time and also cannot model syntactic and semantic generalisations over multiple lexical items. On both the Atis development and the Atis test set, FUBL significantly outperforms UBL by 16% and 11% respectively. This demonstrates the value of lexical generalisation in hard dialogue domains such as Atis.

Interestingly, the addition of partial factorings during training hurts the performance of FUBL although they do help if added in a post-processing step. This is due to the conflict between the increased lexical power and the requirements of learning in a larger lexical search space generated by partial factorings. This is analysed in greater depth later in Section 5.7.

Across evaluations, despite achieving high recall, FUBL achieves significantly lower precision than ZC07 and λ-WASP. This illustrates the trade-off from having a very general model of proposing lexical structure. With the ability to skip unseen words, FUBL returns a parse for all of the Atis test sentences, since the factored lexicons we are learning can produce a very large number of lexical items. These parses are, however, not always correct.

5.7 Analysis

The Atis results in Tables 5.3 and 5.4 highlight the advantages of factored lexicons. FUBL outperforms the UBL baseline by 16 and 11 points respectively in exact-match recall. Without making any modification to the CCG grammars or parsing combinators, FUBL is able to induce a lexicon that is general enough model the natural occurring variations in the data, for example due to sloppy, unedited sentences. Figure 5.1 shows a parse returned by FUBL for a sentence on which UBL failed. While the word "cheapest" is seen 208 times in the training data, in only a handful of these instances is it seen in the middle of an utterance. For this reason, UBL never proposes the lexical item, $\text{cheapest} \vdash \mathsf{NP}\backslash(\mathsf{S}|\mathsf{NP})/(\mathsf{S}|\mathsf{NP}) : \lambda f \lambda g.argmin(\lambda x.f(x) \wedge g(x), \lambda y.cost(y))$, which is used to parse the sentence in Figure 5.1. In contrast, FUBL uses a lexeme learnt from the same word in different contexts, along with a template learnt from similar words in a similar context, to learn to perform the desired analysis.

As well as providing a new way to search the lexicon during training, the factored lexicon provides a way of proposing new, unseen, lexical items at test time. New, non-NP, lexical items are used in 6% of the development set parses. Table 5.5 shows a selection of lexemes and templates learned for Atis. Examples 2 and 3 show that morphological variants of the same word must still be stored in separate lexemes since

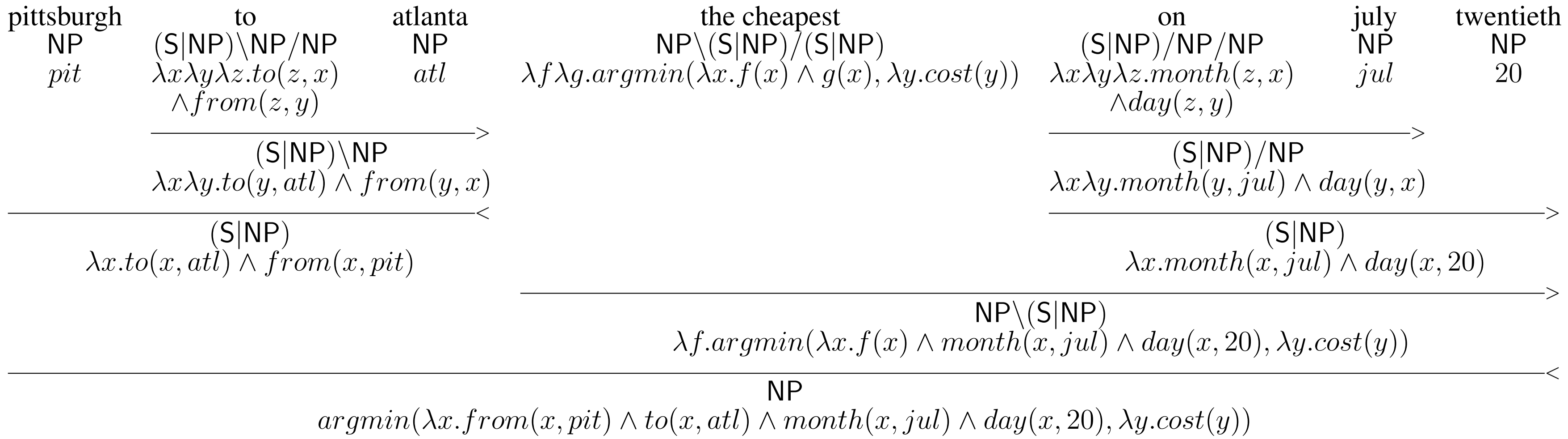

pittsburgh
NP
pit
to
(S|NP)\NP/NP
$\lambda x \lambda y \lambda z.to(z,x) \wedge from(z,y)$
atlanta
NP
atl
the cheapest
NP\(S|NP)/(S|NP)
$\lambda f \lambda g.argmin(\lambda x.f(x) \wedge g(x), \lambda y.cost(y))$
on
(S|NP)/NP/NP
$\lambda x \lambda y \lambda z.month(z,x) \wedge day(z,y)$
july
NP
jul
twentieth
NP
20
(S|NP)\NP
$\lambda x \lambda y.to(y,atl) \wedge from(y,x)$
(S|NP)/NP
$\lambda x \lambda y.month(y,jul) \wedge day(y,x)$
(S|NP)
$\lambda x.to(x,atl) \wedge from(x,pit)$
(S|NP)
$\lambda x.month(x,jul) \wedge day(x,20)$
NP\(S|NP)
$\lambda f.argmin(\lambda x.f(x) \wedge month(x,jul) \wedge day(x,20), \lambda y.cost(y))$
NP
$argmin(\lambda x.from(x,pit) \wedge to(x,atl) \wedge month(x,jul) \wedge day(x,20), \lambda y.cost(y))$

Figure 5.1: An example learned parse. FUBL can learn this type of analysis with novel combinations of lexemes and templates at test time, even if the individual words, like "cheapest," were never seen in similar syntactic constructions during training, as described in Section 5.7.

Most common lexemes by type of constants in $\vec{c}$.			
1	e	(Boston, $[bos]$)	(Denver, $[den]$)
2	$\langle e,t\rangle$	(flight, $[flight]$)	(flights, $[flight]$)
3	$\langle e,i\rangle$	(fare, $[cost]$)	(fares, $[cost]$)
4	$\langle e,\langle e,t\rangle\rangle$	(from, $[from]$)	(to, $[to]$)
5	$\langle\langle e,i\rangle,\langle e,t\rangle\rangle$	(cheapest, $[argmin, cost]$)	(earliest, $[argmin, dep\_time]$)
6	$\langle\langle i,\langle i,t\rangle\rangle,\langle e,i\rangle\rangle$	(after, $[>, dep\_time]$)	(before, $[<, dep\_time]$)
Most common templates matching lexemes above.			
1	$\lambda(\omega,\vec{v}).\omega\vdash \mathsf{NP} : v_1$		
2	$\lambda(\omega,\vec{v}).\omega\vdash \mathsf{S}\mid\mathsf{NP} : \lambda x.v_1(x)$		
3	$\lambda(\omega,\vec{v}).\omega\vdash \mathsf{NP}\mid\mathsf{NP} : \lambda x.v_1(x)$		
4	$\lambda(\omega,\vec{v}).\omega\vdash \mathsf{S}\mid\mathsf{NP}/\mathsf{NP}\backslash(\mathsf{S}\mid\mathsf{NP}) : \lambda x\lambda y.v_1(x,y)$		
5	$\lambda(\omega,\vec{v}).\omega\vdash \mathsf{NP}/(\mathsf{S}\mid\mathsf{NP}) : \lambda f.v_1(\lambda x.f(x),\lambda y,v_2(y))$		
6	$\lambda(\omega,\vec{v}).\omega\vdash \mathsf{S}\mid\mathsf{NP}\backslash(\mathsf{S}\mid\mathsf{NP})/\mathsf{NP} : \lambda x\lambda y\lambda z.v_1(v_2(z),x)\wedge y(x)$		

Table 5.5: Example lexemes and templates learned from the Atis development set.

the morphological transformations on lexemes are not used in this setting. However, as these lexemes now share templates, the total number of lexical variants that must be learned is reduced.

As we saw in the previous section, the addition of lexical templates through partial factorings (as described in Section 5.3.2) and morphological transformations on lexemes can, in some circumstances, hurt the performance of FUBL even though the probabilistic model has the ability to discriminate against incorrect generalisations. Every time a new lexical template or morphological transformation is created, the number of parses licensed by each sentence and (sentence, logical-form) pair increases. If either of these sets of parses is too big for FUBL to represent in a packed parse chart then the calculation of the gradient from Equation 4.7 used in the stochastic gradient descent step is approximate. The more parses that are licensed for each training instance, the more approximate this calculation is. Therefore, increasing the number of allowed parses for any training sentence has a detrimental effect on the accuracy of the parameter estimation procedure. Figure 5.2 illustrates the performance of FUBL+m (with morphological transformations) and FUBL (without) on the Geo880 development data (performed using 10 fold cross validation). The addition of morphology slows down

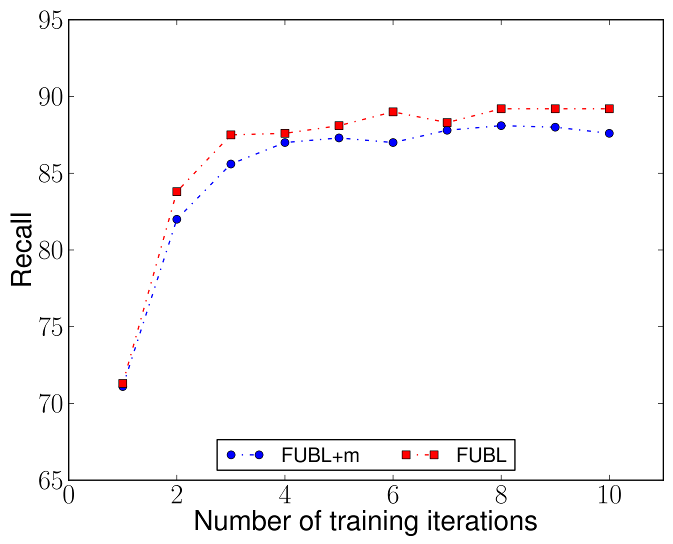


Figure 5.2: FUBL Performance on Geo880 With and Without Morphological Transformations

learning and is not recovered from by the end of training even though the model has the ability to discriminate against all of the (finite) set of morphological transformations allowed. More significant is the reduction in performance that occurs for the Atis data when partial factorings are allowed. This is surprising as elliptical constructions are found in a large proportion of the Atis utterances. What happens is that, in practice, FUBL learns to model many elliptical constructions with lexemes and templates introduced through maximal factorings without the need for partial factorings. For example, the lexeme $(\text{to}, [from, to])$ can be used with the correct lexical template to deal with our motivating example "flights Boston to New York". Templates that introduce content are therefore only used in truly novel elliptical constructions for which an alternative analysis could not be learnt. These are more common when FUBL is only trained on a subset of Atis, as illustrated in Figure 5.3. When trained on the first 100, 500 and 1000 training instances from the Atis data, the maximum recall of FUBL (with partial factorings) is greater than that of FUBL-pf (without). When the systems are trained on the first 2000 training instances, the difference in performance has all but disappeared and, when trained on the full training set, FUBL-pf significantly outperforms FUBL as it has to contend with a far smaller search space. This is illustrated in Figure 5.4. The contention that the reduction in performance resulting from the addition of partially factored templates is due to a problem with the parameter estimation procedure is supported by the increase in recall that occurs when templates from par-

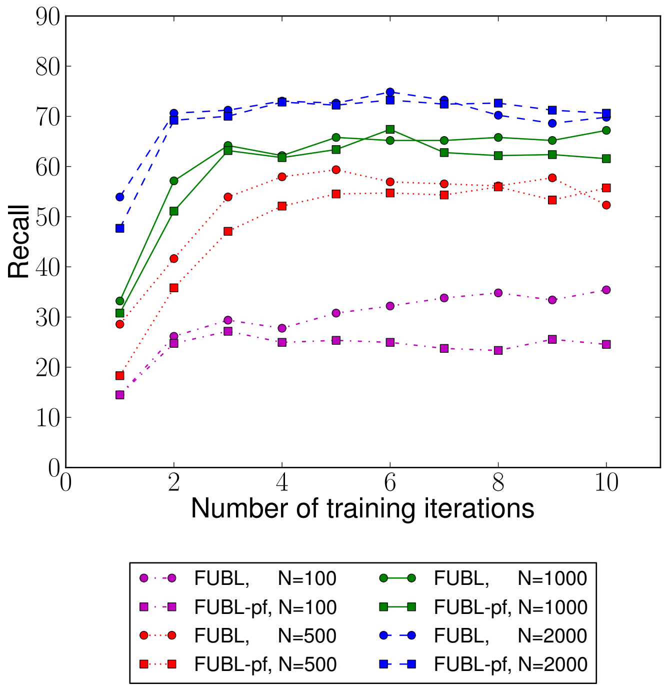


Figure 5.3: Performance on Subset of Atis With and Without Partial Factorings

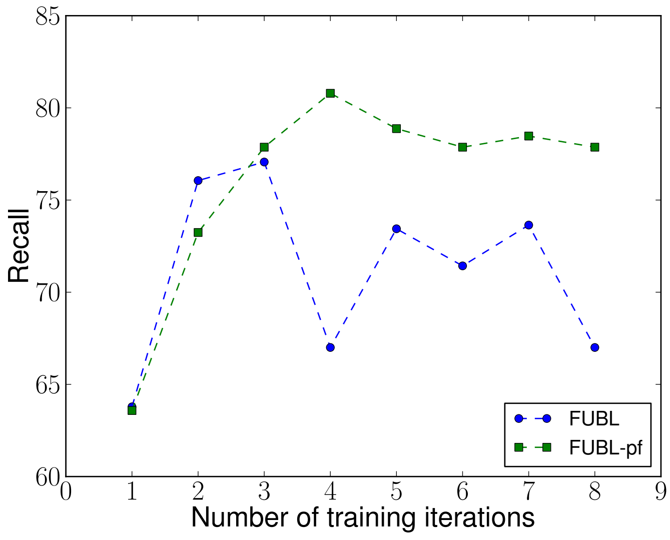


Figure 5.4: Performance on Atis With and Without Partial Factorings

tial factorings are introduced to FUBL+pf in a post-processing step. However, even in this case, the increase in performance of 1.2% is modest as the standard factorised lexicon is learning to handle many elliptical constructions without the need for lexical templates that introduce new logical constants.

5.8 Discussion and Future Work

I have showed that, in situations that suffer from lexical sparsity, factored CCG lexicons, which include both lexemes and lexical templates, provide a compact representation of lexical knowledge that can have advantages for learning. However, increasing the generative power of the factored lexicon (with extra lexical templates and morphological transformations on lexemes) can also have a detrimental effect on the parameter estimation step as the approximation to the local gradient of the objective function becomes less accurate. Future work could investigate the use of different parameter estimation alogrithms that do not require the calculation of all parse probabilities. Perceptron methods only require a single, most probable, parse to be found. This can be done without packing the entire chart of all possible parses via strategies such as $A^\star$ parsing (Klein and Manning, 2003; Pauls and Klein, 2010).

In addition to increasing the coverage of induced semantic parsers in domains for which logical annotations exist, the factored lexical representation has significant potential for lexical expansion or transfer learning between domains without the need for logical-form annotations. Lexical templates are language specific but domain independent. The lexemes that match words to logical constants contain domain specific information. Having learnt a set of lexical templates from annotated data, a parser could be adapted for a new domain through the process of simply learning a new set of lexemes. This could be done for domains in which a back-end database exists by learning the lexemes using supervision in the form of question-answer pairs as was done by Clarke et al. (2010); Liang et al. (2011). In domains where some lexemes are known, new lexemes could be learnt by adding and clustering predicate symbols with unsupervised methods of the type used by Poon and Domingos (2009, 2010); Titov and Klementiev (2011).

Chapter 6

Modelling Language Acquisition

The work presented in this chapter has been submitted for publication as Kwiatkowski et al. (2012).

6.1 Introduction

Children learn their first language by mapping the utterances that they hear onto what they believe those utterances mean. This *cognitive theory of language acquisition* (Pinker, 1979) assumes that the "child can hypothesise structured semantic representations corresponding to what parents are likely to be referring to and can refine such representations across multiple situations". The nature of the child's pre-linguistic representation of potential utterance meanings are unknown. Indeed, it has been argued by Tomasello (1999) that they are likely to be largely interpersonal and intensional in content. However, it is possible to represent at least a part of the meanings to which the child has access with compositional logical representations of the type introduced in Chapter 2 and illustrated here:

$$\begin{aligned} \mathsf{Utterance}: &\ \text{you have another cookie} \\ \mathsf{Meaning}: &\ have(you, another(x, cookie(x))) \end{aligned} \tag{6.1}$$

where the meaning is a logical expression that describes a relationship $have$ between the person you refers to and the object $another(x, cookie(x))$. Most situations in which the child hears an utterance will support a number of plausible utterance meanings, so the child will need to learn in the face of *propositional uncertainty*, which can be approximated with a set of contextually afforded meaning candidates, as here:

$$\begin{array}{rcl} \textsf{Utterance} & : & \text{you have another cookie} \\ \begin{array}{c}\textsf{Candidate} \\ \textsf{Meanings}\end{array} & : & \left\{ \begin{array}{c} have(you, another(x, cookie(x)) \\ eat(you, your(x, cake(x)) \\ want(i, another(x, cookie(x)) \end{array} \right\} \end{array}$$

The task facing the child is then to learn, from a sequence of such (utterance, meaning-candidates) pairs, the correct lexicon and parsing model for the target language. This task is is similar to the task of learning a semantic parser visited in Chapters 4 and 5.

This chapter presents a probabilistic account of the learning task described above. In order to be psycholinguistically plausible the learner must not require any language-specific information prior to learning and the learning algorithm must be strictly *online*: it sees each training instance sequentially and exactly once. It could be argued that, in order to achieve true psycholinguistic plausibility, the learning procedure should also fulfil other criteria such as limited memory and a restriction on available computational power. Later in the learning task, when the child is dealing with complex utterances, these considerations will certainly be important and they are addressed further in Section 6.9. However, even the criteria introduced above set the approach presented here apart from the work on inducing semantic parsers addressed in Chapters 4 and 5 and reviewed in Section 4.9. Although the task definition is the same, these systems are not designed to model language acquisition and therefore use batch training algorithms or multiple passes over the training data, and are often initialized with language-specific information (lists of noun phrases and additional corpus statistics).

While knowledge of specific languages cannot be included in a model of acquisition, linguists disagree about how much and what type of knowledge specific to the task of language acquisition should be included. Several previous models of the syntactic-semantic learning task (Siskind, 1992; Villavicencio, 2002; Buttery, 2006) are based on the Principles and Parameters (P&P) grammatical framework, which assumes detailed knowledge of linguistic alternations. In addition, these and certain (syntax-only) P & P learners are designed to learn a single, correct, deterministic grammar, whether they use deterministic or statistical learning algorithms (Gibson and Wexler, 1994; Sakas and Fodor, 2001; Yang, 2002). The present approach contrasts with all of these models by assuming a more general kind of domain-specific knowledge and a probabilistic grammar. The only notion of *universal grammar* to which the learner has access is the function $\mathcal{T}$ from Equation 1.2 - defined purely in terms of a small set of general combinatory schemata and a functional mapping from semantic type to syntactic category.

The learning problem facing the model of language acquisition is that of learning, from a corpus of (utterance, meaning-candidates) pairs $\{(s_i, \{m\}_i) : i = 1, \ldots, N\}$, the probabilistic parser $\mathcal{P}$ that predicts a most probable meaning for utterance s':

$$\mathcal{P}(s') = \arg\max_m \; p(s', m | \Lambda, \{n_{a \to b}\}) \tag{6.2}$$

where Λ is a CCG lexicon and $\{n_{a \to b}\}$ is a set of pseudo-counts that parameterise a probabilistic parsing model. The grammar is modelled probabilistically using an infinite Bayesian model with Dirichlet Process priors. For the experiments presented in this Chapter, the probabilistic parsing model is trained on a set of (utterance, meaning-candidates) pairs derived from the Eve corpus collected by Brown (1973) in a longitudinal study of a single child between the ages of 18 and 27 months. Tomasello (1995) has shown that by the age of 9-12 months children have already learnt to recognize others as intensional agents. It is not therefore unreasonable to assume that by the beginning of this corpus Eve has learnt to make inferences about intended meaning of the speaker, although the meaning representations used are greatly simplified in this respect.

Having learnt a proababilistic grammar, I evaluate it in three ways. First, I test the accuracy of the trained model in parsing unseen sentences onto gold standard annotations of their meaning and show that it outperforms the state-of-the-art semantic parser (UBL) from Chapter 4 when run with similar training conditions (i.e., neither system is given the corpus based initialization originally used by UBL). I then examine the learning curves of some individual words, showing that the model can learn word meanings for novel words bearing known syntactic categories on the basis of a single exposure, similar to the *fast mapping* phenomenon observed in children (Carey and Bartlett, 1978). Finally, I show that the learner captures the step-like learning curves for word order regularities that Thornton and Tesan (2007) claim children show, countering one of Thornton and Tesan's main criticisms of statistical grammar learners—that they tend to exhibit gradual learning curves rather than the abrupt changes in linguistic competence they observed in children.

This chapter proceeds as follows. Section 6.2 reviews some results from experimental studies of language acquisition that I wish to replicate with the computational approach presented here. These results will be referenced throughout this chapter, are used in a qualitative evaluation of the model in Section 6.8 and are discussed further in Section 6.9.

Section 6.3 introduces the logically annotated corpus of child-directed utterances from which a grammar must be learnt. Then section 6.4 defines the function $\mathcal{T}$ from Chapter 3 for this task and dataset, adding linguistically motivated constraints that allow $\mathcal{T}$ to be used tractably. Section 6.5 presents the generative probabilistic model with which I assign probabilities to all possible (utterance, logical-form, derivation) triples and Section 6.6 introduces the Online Variational Bayesian Expectation Maximization algorithm used to learn the correct parameterisation of this probabilistic model along with the full online training algorithm in which it is used. Section 6.8 presents some qualitative and quantitative evaluations of this learning approach and Section 6.9 discusses these with relation to the relevant psychological and computational literature.

Finally, Section 6.10 outlines previous work in modelling a child's acquisition of word-meanings and syntax and describes how the approach presented here differs from all of these.

6.2 Empirical Studies of Language Acquisition

A computational model of human language acquisition must not only learn the target language, it must also do so in a way that correlates with observations of language learning in children. Making this comparison is not easy since our models only approximate a small component of the child's perceptual and linguistic faculties—rendering impossible an exact replication of the experiments with which the child's linguistic faculty is tested. However, in this section I present a selection of observations of the development of linguistic competence in children, some of which have been performed in direct response to previous computational accounts of language acquisition. I describe how these observations may be related - albeit indirectly - to observations of the learning algorithm that I shall present in this chapter. I also relate these observations to the performance of previous computational models of language acquisition, all of which are reviewed in greater detail in Section 6.10.

Crain and Thornton (1998) investigated the manner in which children learn syntax through the categorisation of utterances elicited from children according to linguistic parameters defined in the principles and parameters (P&P) framework (Chomsky

(1981), Hyams (1986)). These experiments showed, among other things, that children do produce constructions for which they have seen no evidence before, such as medial-wh words in English. This initial variation in production provides evidence against accounts of learning that proceed purely through *input matching* and suggest that any model of language acquisition should always have the ability to support constructions allowed by universal grammar but which have never been observed.

Thornton and Tesan (2007) accumulate further experimental observations of children's linguistic performance in order to determine whether the triggering approach proposed by Baker (2005) and Wexler (1998) and implemented computationally by Gibson and Wexler (1994) and Sakas and Fodor (2001) should be preferred to the statistical parameter learner presented by Yang (2002). In order to discriminate between these approaches, Thornton and Tesan investigate the manner in which linguistic regularities corresponding to parameters in a P&P view of grammar are learnt. The study of the rates at which a small set of parameters were learnt led Thornton and Tesan to state that:

> "The empirical findings from our longitudinal study of four children's development of inflection and negation do not support the proposal that statistical learning is driving children's parameter-setting. Our empirical findings show, instead, that children initiate grammatical change at some point in time, and when change is initiated, it takes hold quickly"— Thornton and Tesan 2007, pp.93

While the parameters that Thornton and Tesan (2007) investigated represent inflectional and grammatical information that is not represented explicitly or at all in my model, the arguments against statistical learning on the basis of gradual learning rates is one that can be countered through observations of the learning rates of other linguistic phenomona. In order to do this I shall investigate the probabilities of competing word-orders over time.

The experimental work mentioned above examined specific morpho-syntactic parameters in the P&P framework. Separate work has also investigated the ability of children to learn the meanings of new words. One phenomonon with ample evidence behind it is *fast mapping* (Carey and Bartlett, 1978) in which children can learn the meaning of a new word on the basis of a single exposure if it is heard in a familiar context. Alishahi et al. (2008) show that, using the model and approach of Fazly et al. (2008) "the probabilistic bootstrapping approach to word learning naturally leads to the onset of fast mapping in the course of lexical development, without hard-coding any specialized learning mechanism into the model to account for this phenomenon". As the model that I present here also enforces a probabilistic bootstrapping effect when a large proportion of the context is known, it should be able to learn words in a way reminiscent of the fast mapping phenomonon.

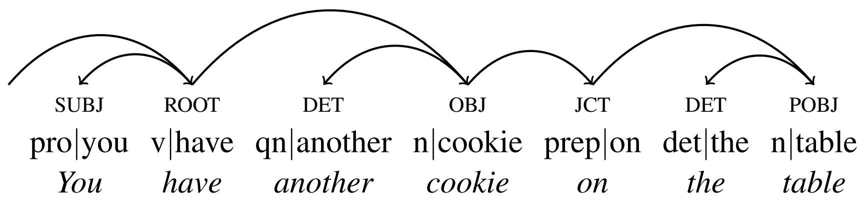
6.3 The Eve Corpus with Logical-Forms

The theory of language acquisition set forward by this chapter assumes that, in addition to utterances from the target language, the child has access to structured representations of those utterances' meaning. In order to test the model it is therefore neccessary to generate a corpus that pairs child directed utterances with a proxy for the structured representations available to the child. We currently have very little idea of how to write down a psychologically correct representation of the semantics available to the child. However, theories of such a semantics fall outside of the scope of this thesis. The only requirements of the semantics used as an input to the grammar learner is that it should be compositional in nature, it should be made up of typed components, and it should not explicitly or implicitly encode language specific information.

The logical expressions reviewed in Section 2.1 are compositional and have typed components. The argument in favour of their language independence is somewhat weaker. This is because we do not have access to a single logical language that underpins the semantics of all natural languages. There is therefore no way to know that the semantics of all languages will factor into the set of atomic constants, functions and predicates in the representation that we choose. I therefore settle on a slightly less strict notion of language independence. The logical expressions should not encode any *word order* information or any information about the *correct segmentation of the semantics* above the level of atomic semantic constituents.

Appendix A.1 to the present dissertation describes a deterministic mapping that generates a logical-forms from Sagae et al. (2004)'s syntactic dependency tree annotations of the Eve corpus (Brown, 1973). This mapping takes, as input, dependency trees such as:
And generates, by matching sub-parts of the tree to hand generated tree templates that correspond to well understood logical structure, logical-forms such as:

$$\lambda ev.have(you,\ another(y,\ cookie(y)),\ ev) \wedge on(the(z,\ table(z))\ ,ev)$$



In which the Davidsonian event variable ev has been used to handle the attachment of a prepositional predicate. The logically annotated Eve corpus contains 5,832 child-directed utterances paired with well typed logical representations of their meanings. These are used in training and testing the system presented in this chapter, and the corpus will in due course be made publicly available.

6.4 The Grammar

The model of language acquisition presented in this chapter assumes that during learning the child is capable of hypothesising all parses consistent with a $(s, \{m\})$ pair. This set of parses can be proposed with the function $\mathcal{T}$ introduced in Chapter 3. This function has so far been defined in general terms. In this section I describe how it is adapted to the problem of proposing parses for the model of language acquisition.

6.4.1 Constraints on Semantic Decomposition

The function $\mathcal{S}^m$ in Section 3.2 is used by $\mathcal{T}$ to decompose an original logical-form h into pairs of logical-forms $\{(f, g)\}$. While this function returns a finite number of splits for any given h, there will often still be too many of these splits to enumerate. Section 4.4 introduces a set of constraints on $\mathcal{S}^m$ that are designed for the task of learning grammars that parse sentence onto human annotated logical forms—where assumptions can be made about the granularity of logical predicates and the logical-form of lexicalisable units. When proposing a model of language acquisition one cannot make such assumptions, but $\mathcal{S}^m$ is too expensive to be used unconstrained. For this reason I introduce four new constraints on $\mathcal{S}^m$, three of which are supported by theories of universal linguistic phenomena.

Principle of Inheritance

When splitting a logical-form, the argument may be a function with valence greater than 1 (more than 1 lambda-term at the root). In $\mathcal{S}^m$ as defined in Algorithms 2-4 all orders of these lambda-terms are allowed. When splitting a logical-form $\mathcal{S}^m$ returns $n!$ splits in which the arguments are equivalent in every way other than the ordering of the n lambda-terms at their roots. For example, the following two splits are allowed:

$$h = \lambda x \lambda y.m(n(x,y)) \Rightarrow f = \lambda p \lambda x \lambda y.m(p(x,y)) \qquad \lambda w \lambda v.n(w,v)$$
$$h = \lambda x \lambda y.m(n(x,y)) \Rightarrow f = \lambda p \lambda x \lambda y.m(p(y,x)) \qquad \lambda v \lambda w.n(w,v)$$

But the reversed order of λw and λv in the second of these violates the *principle of inheritance* required by the *syntactic principle of inheritance* introduced in Section 3.3. Since the semantic principle of inheritance is necessary for the conservation of argument order in the category split, it a neccessary precursor to the syntactic principle of inheritance.

When a logical expression is decomposed by reversing function composition the semantic principle of inheritance is implicitly enforced. This is not, however, the case when the split is performed by reversing function application. For this, an extra constraint is created. In the following split, the principle of inheritance is upheld as argument order is maintained through the semantic split (both the parent and the argument bind x before y). This split is allowed.

$$\lambda x \lambda y.g(f(x,y)) \;\Rightarrow\; \lambda z \lambda x \lambda y.g(z(x,y)) \qquad \lambda x \lambda y.f(x,y)$$

Conversely, in the following split the argument binds y before x, re-ordering the order set by the parent. This split is not allowed.

$$\lambda x \lambda y.g(f(x,y)) \not\Rightarrow \lambda z \lambda x \lambda y.g(z(y,x)) \qquad \lambda y \lambda x.f(x,y)$$

Maximum Valence of Semantics

There are strong arguments that there is never any need in language for a predicate with valence of greater than four (Hayes et al., 2001, pp. 226). While this argument is made from the perspective of lexicalizable units, we infer that the same constraints can be applied to all constituents in a parse. For this reason we do not allow any semantic splits that would result in a constituent with a valence of more than four.

The Across-the-Board Constraint

Ross (1967) discovered *islands* that block the extraction of constituents from constructions. One of these islands blocks extraction out of co-ordinations unless it is *across-the-board*. The *Across-the-Board Constraint* states that it only possible to extract from a coordination construction if the same constituent is extracted from all conjuncts simultaneously. The following split is disallowed as $john$ is an argument of only one half of the conjunct:

$$like(mary, john) \wedge like(jack, jill) \not\Rightarrow \lambda x.like(mary, x) \wedge like(jack, jill) \qquad john$$

Whereas $john$ can be extracted if it is an argument of all conjuncts:

$$like(mary, john) \wedge hate(jack, john) \Rightarrow \lambda x.like(mary, x) \wedge hate(jack, x) \qquad john$$

or alternatively, the split has to separate the conjunction at the highest possible level. This is always allowed:

$$like(mary, john) \wedge like(jack, jill) \Rightarrow \lambda x.like(mary, john) \wedge x \qquad like(jack, jill)$$
$$like(mary, john) \wedge like(jack, jill) \Rightarrow \lambda x.like(jack, jill) \wedge x \qquad like(mary, john)$$

Limiting Sub-Tree Substitution

The main contribution to the complexity of $\mathcal{S}^m$ is the ability of Algorithm 3 to return logical trees in which arbitrarily many logical sub-structures have been substituted with variables. The semantic decomposition used by UBL avoids this complexity by disallowing logical forms in which a variable is applied to a non-variable. In modelling language acquisition I relax the constraint imposed by UBL and allow a single new variable applied to a non-variable in both f and g. Therefore the logical-form $\lambda x.will(x(i))$ used for “I'll” is allowed but $\lambda x \lambda y.will(x(i) \wedge y(you))$ is not.

The limit upon sub tree substitutions and principle of inheritance between them reduce the $O(2^n)$ complexity of Algorithm 3 to being polynomial in the number of logical nodes n. The across-the-board constraint does not affect the asymptotic complexity of the algorithm but I do find, empirically, that it significantly reduces the number of possible splits.

6.4.2 Semantic Type and Syntactic Category

The grammar used to model the Eve corpus has seven basic categories: $\mathsf{S_{dcl}}$ for declarative sentences about events; $\mathsf{S_t}$ for declarative stative sentences that make true-false statements about properties; $\mathsf{S_{wh}}$ and $\mathsf{S_{wht}}$ for wh-questions; $\mathsf{S_q}$ for yes-no questions; N for nouns; NP for noun phrases and PP for prepositional phrases. In order for there to be no type conflict when using the CCG combinators to parse, there must be a single semantic type assigned to each syntactic category. For this reason it is necessary to introduce the $\mathsf{S_t}$ and $\mathsf{S_{wht}}$ categories to deal with sentences like "it's brown" and "what's brown" that do not have events associated with them (introducing an event would involve creating a new predicate for "brown"). This is an imperfect solution but is sufficient for the purposes of the experiments presented here.

Syntactic Category	Semantic Type	Example Phrase
$\mathsf{S_{dcl}}$	$\langle ev, t\rangle$	I took it $\vdash \mathsf{S_{dcl}} : \lambda e.took(i, it, e)$
$\mathsf{S_t}$	t	I'm angry $\vdash \mathsf{S_t} : angry(i)$
$\mathsf{S_{wh}}$	$\langle e, \langle ev, t\rangle\rangle$	Who took it? $\vdash \mathsf{S_{wh}} : \lambda x \lambda e.took(x, it, e)$
$\mathsf{S_{wht}}$	$\langle e, t\rangle$	Who's angry? $\vdash \mathsf{S_{wht}} : \lambda x.angry(x)$
$\mathsf{S_q}$	$\langle ev, t\rangle$	Did you take it? $\vdash \mathsf{S_q} : \lambda e.Q(take(you, it, e))$
N	$\langle e, t\rangle$	cookie $\vdash \mathsf{N} : \lambda x.cookie(x)$
NP	e	John $\vdash \mathsf{NP} : john$
PP	$\langle ev, t\rangle$	on John $\vdash \mathsf{PP} : \lambda e.on(john, e)$

Table 6.1: Basic Syntactic Categories.

The mapping from basic syntactic category to semantic type is given in Table 6.1. Since there are multiple basic syntactic categories with the same semantic type and since I believe that this reflects a deficiency in the semantic typing system, I restrict the set of allowed basic syntactic categories with a small set of heuristics that subcategorise logical-forms according to their primary constants. These heuristics distinguish, for example, between the syntactic categories $\mathsf{S_{wht}}$ and N assigning the former to wh-sentences only and the latter to noun-predicates or noun-predicates with modifiers. With the addition of these heuristics, each logical form may be assigned multiple CCG syntactic categories but the only way in which they differ will be in the direction of their slashes.

6.5 A Generative Model of Parse Structure

The objective of the learning algorithm is to learn the correct parameterisation of a probabilistic model $P(s,m,t)$ over (utterance, meaning, derivation) triples. This *generative* model assigns a probability to each of the *grammar productions* $a \rightarrow b$ used to build the derivation tree t. Generative models over phrase-structure trees (Collins, 1997) and CCG derivation trees (Hockenmaier and Steedman, 2002) achieve high performance in the task of syntactic parsing. They are also desirable from the point of view of psycholinguistic plausibility as they model the joint distribution over sentence and meaning and could therefore be used to model both language production and comprehension. Furthermore, generative models have well defined marginal distributions which will prove to be useful when isolating specific linguistic phenomona for investigation as we will see in Section 6.8.

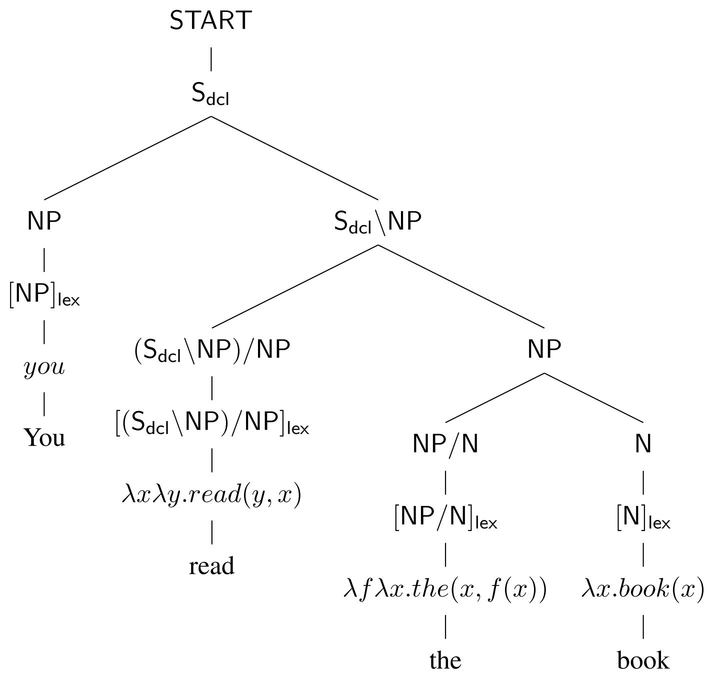
A CCG derivation tree is built using productions $a \rightarrow b$ that expand a head a into a result b. The probability of any given CCG derivation t with sentence s and semantics m is then calculated as the product of the conditional probabilities of all of its productions.

$$P(s,m,t) = \prod_{a \rightarrow b \in t} P(b|a) \tag{6.3}$$

For example, the derivation in Figure 6.1 contains 13 productions, and the probability of this derivation is the product of the 13 production probabilities.
Grammar productions are be either *syntactic*—used to build a syntactic derivation tree, or *lexical*—used to generate logical expressions and words at the leaves of this tree.

A syntactic production $\mathsf{C_h} \rightarrow \mathcal{R}$ expands a head node $\mathsf{C_h}$ into a result $\mathcal{R}$ that is either an ordered pair of syntactic parse nodes $\langle \mathsf{C_l}, \mathsf{C_r} \rangle$ (for a binary production) or a single parse node (for a unary production). Only two unary syntactic productions are allowed in the grammar: $\mathsf{START} \rightarrow \mathsf{A}$ to generate A as the top syntactic node of a parse tree and $\mathsf{A} \rightarrow [\mathsf{A}]_{\mathsf{lex}}$ to indicate that A is a leaf node in the syntactic derivation and should be used to generate a logical expression and word. All binary productions are licensed by one of the inverted CCG combinators introduced in Section 3.3. Syntactic derivations are built by recursively applying syntactic productions to non-leaf nodes in the derivation tree. Each syntactic production $\mathsf{C_h} \rightarrow \mathcal{R}$ has conditional probability $P(\mathcal{R}|\mathsf{C_h})$. There are 3 binary and 5 unary syntactic productions in Figure 6.1.

Lexical productions have two forms. *Logical expressions* are produced from leaf nodes


START
$\mathsf{S_{dcl}}$
NP
$\mathsf{S_{dcl}\backslash NP}$
$[\mathsf{NP}]_{\mathsf{lex}}$
you
You
$(\mathsf{S_{dcl}\backslash NP})/\mathsf{NP}$
$[(\mathsf{S_{dcl}\backslash NP})/\mathsf{NP}]_{\mathsf{lex}}$
$\lambda x \lambda y.read(y,x)$
read
NP
NP/N
$[\mathsf{NP/N}]_{\mathsf{lex}}$
$\lambda f \lambda x.the(x,f(x))$
the
N
$[\mathsf{N}]_{\mathsf{lex}}$
$\lambda x.book(x)$
book

$$\begin{aligned}P(t,s,m) =& P(\mathsf{S_{dcl}}|\mathsf{START}) \times P(\langle \mathsf{NP}, \mathsf{S_{dcl}\backslash NP}\rangle|\mathsf{S_{dcl}}) \times P(\langle (\mathsf{S_{dcl}\backslash NP})/\mathsf{NP}, \mathsf{NP}\rangle|\mathsf{S_{dcl}\backslash NP}) \times P(\langle \mathsf{NP/N}, \mathsf{N}\rangle|\mathsf{NP})\\ &\times P([\mathsf{NP}]_{\mathsf{lex}}|\mathsf{NP}) \times P([(\mathsf{S_{dcl}\backslash NP})/\mathsf{NP}]_{\mathsf{lex}}|(\mathsf{S_{dcl}\backslash NP})/\mathsf{NP}) \times P([\mathsf{NP/N}]_{\mathsf{lex}}|\mathsf{NP/N}) \times P([\mathsf{NP}]_{\mathsf{lex}}|\mathsf{NP}) \times P(you|[\mathsf{NP}]_{\mathsf{lex}})\\ &\times P(\lambda x \lambda y.read(y,x)|[(\mathsf{S_{dcl}\backslash NP})/\mathsf{NP}]_{\mathsf{lex}}) \times P(\lambda f \lambda x.the(x,f(x))|[\mathsf{NP/N}]_{\mathsf{lex}}) \times P(\lambda x.book(x)|[\mathsf{NP}]_{\mathsf{lex}})\\ &\times P(\mathrm{You}|you) \times P(\mathrm{read}|\lambda x \lambda y.read(y,x)) \times P(\mathrm{the}|\lambda f \lambda x.the(x,f(x))) \times P(\mathrm{book}|\lambda x.book(x))\end{aligned}$$

Figure 6.1: Derivation of sentence `You read the book` with meaning $read(you, the(x, book(x)))$.

in the syntactic derivation tree with productions of the form $\mathsf{A_{lex}} \rightarrow m_{leaf}$ that have conditional probability $P(m_{leaf}|\mathsf{A_{lex}})$. *Words* are then produced from these logical expressions with productions $m_{leaf} \rightarrow w$ and conditional probability $P(w|m_{leaf})$. An example logical production from Figure 6.1 is $[\mathsf{NP}]_{\mathsf{lex}} \rightarrow you$. An example word production is $you \rightarrow$ You.

The set of logical productions available to the grammar is constrained by the fact that the type of the logical expression produced must match the type of the syntactic category from which it is produced. Word productions are allowed to generate *multi-word elements*.

The generative model over (sentence, logical-form, derivation) triples is tight — if all natural language strings are in the language L, all logical-forms are in the logical language M and all derivations are in the universal syntactic grammar G, the following sum holds:

$$\sum_{s \in L, m \in M, t \in G} P(s, m, t) \ = \ 1$$

In the model presented here, the language L is the set of strings that can be created with the Latin alphabet; the language M consists of all logical forms that can be created with a pre-defined set of logical constants; and the universal syntactic grammar G is defined in terms of the combinators of CCG and a mapping from semantic type to syntactic category.

6.5.1 Distributions over Productions

Every production $a \rightarrow b$ used in a parse tree t is chosen from the set of productions that could be used to expand a head node a. If there are a finite K productions that could expand a then a K-dimensional *Multinomial distribution* parameterised by θ_a can be used to model the categorical choice of production.

$$b \sim \text{Multinomial}(\theta_a) \tag{6.4}$$

However, before training a model of language acquisition the set of productions needed to model the target language are unknown. In order to maintain a probability model with cover over the countably infinite number of possible productions,[1] I define a Dirichlet Process (DP) prior (Ferguson, 1973) for each possible production head a. For the production head a, $DP(\alpha_a, H_a)$ can be used to produce a distribution G_a over all possible production targets $\{b\}$ covered by the base distribution H_a.

$$G_a \sim \text{DP}(\alpha_a, H_a) \tag{6.5}$$

The form of the base distributions for syntactic, logical and word productions will be given later, along with the concentration parameters α_a, in Section 6.7.

It is possible to use the DP as an infinite prior from which the parameter set of a finite dimensional Multinomial may be drawn provided that we can choose a suitable finite partition of $\{b\}$. When calculating the probability of an (s, m, t) triple, the choice of this partition is easy. For any given production head a there is a finite set of usable

[1]There is an implicit limit set on the set of possible CCG productions and categories as a result of the maximum valence of the semantic decomposition step. However the set of possible CCG productions allowed by our *universal grammar* is still too large to enumerate. There are an unbounded number of possible lexical items (there is no limit on either the length of word-strings or the complexity of logical forms).

production targets $\{b_1, \ldots, b_{k-1}\}$ in t. I create a partition that includes one entry for each of these along with a final entry $\{b_k, \ldots\}$ that includes all other ways in which a could be expanded in different contexts. Then, applying the distribution G_a drawn from the DP to this partition gives a k-dimensional vector that is equivalent to a draw from a k dimensional Dirichlet distribution.

$$\begin{aligned}&(G_a(b_1), \ldots, G_a(b_{k-1}), G_a(\{b_k, \ldots\})) \\ &\qquad \sim \mathrm{Dir}(\alpha_a H_a(b_1), \ldots, \alpha_a H_a(b_{k-1}), \alpha_a H_a(\{b_k, \ldots\}))\end{aligned} \tag{6.6}$$

The Dirichlet distribution is the conjugate prior of the Multinomial distributions that we are interested in. Given a context of finite scope, supporting $k-1$ productions emanating from a, the k-dimensional Multinomial describing the distribution over these productions and the set of all others is parameterised by the vector θ_a^k, where:

$$\theta_a^k = (G_a(b_1), \ldots, G_a(b_{k-1}), G_a(\{b_k, \ldots\})) \tag{6.7}$$

Together, Equations 6.3-6.7 can be used to describe the joint distribution $P(\boldsymbol{D}, \boldsymbol{S}, \boldsymbol{\theta})$ over the observed training data $\boldsymbol{D} = \{(s_i, \{m\}_i) : i = 1, \ldots, N\}$, the latent variables $\boldsymbol{S}$ (signalling the productions used in each derivation t) and the parsing parameters $\boldsymbol{\theta}$. More specifically, when the latent variables signal one derivation per training instance $\boldsymbol{S} = \{t_i | i = 1, \ldots, N\}$ the joint distribution is:

$$P(\boldsymbol{D}, \boldsymbol{S}, \boldsymbol{\theta}) = \left(\prod_{i=1}^{N} \prod_{a \to b \in t_i} \theta_{a \to b} \right) \times \prod_{\theta_{a \to b} \in \boldsymbol{\theta}} \mathrm{Dir}(\theta_{a \to b} | \alpha_a H_a) \tag{6.8}$$

This joint distribution is described in terms of the base distributions H_a and concentration parameters a. There is a separate base distribution for each production head a and all of these are summarised below. Then, later in this section the relationship between the prior distribution over parameters given in Equations 6.7 and 6.6 and the posterior distribution that we wish to estimate is described before, in the following section, the training algorithm used to estimate these posteriors is defined.

6.5.2 Model Base Distributions

Each Dirichlet process used to describe the distribution over expansions of a given production head a requires a base distribution H_a that assigns a probability to every allowable expansion of a. There are three types of base distribution used in the model presented here. The syntactic rule base distribution describes all possible expansions of an internal node in a syntactic derivation tree. The logical-expression base distribution describes a distribution over all logical expressions of a given type. The word base distribution is over all possible words.

Syntactic Rules Base Distribution

Syntactic rules are used to expand a syntactic head a into a result $\mathcal{R}$. This result may be a stop node a_{lex} signalling that a is a leaf in the syntactic parse tree, an ordered pair $\langle \mathsf{C_l}, \mathsf{C_r} \rangle$ of CCG syntactic categories, or a single CCG syntactic category in the case that a is the START node. Productions from the start node have a base distribution over CCG categories $P(\mathsf{X})$. The category $\mathsf{Y/Z}$ has $\mathtt{range}(\mathsf{Y/Z}) = \mathsf{Y}$ and $\mathtt{domain}(\mathsf{Y/Z}) = \mathsf{Z}$. $\mathbf{C_{at}}$ is the set of basic CCG syntactic categories.

$$P(\mathsf{X}) = \begin{cases} 1/|\mathbf{C_{at}}| & \text{if } \mathsf{X} \text{ is atomic} \\ P(\mathtt{range}(\mathsf{X})) \times P(\mathtt{domain}(\mathsf{X})) & \text{otherwise} \end{cases} \tag{6.9}$$

This distribution over basic categories is also used by the base distribution for non START nodes. The distribution over these productions assigns probability 0.5 to all lex productions and, for all other productions, uses Equation 6.9 to assign a probability to the new argument category, returned from $\langle \mathsf{C_l}, \mathsf{C_r} \rangle$ by the arg function. The use of inverted function composition is scored according to the order of composition allowed (the number of outermost arguments in $\mathsf{C_h}$ with aligned slashes)—calculated with the `forcomp` function.

$$P(\mathcal{R} \mid a) = \begin{cases} 0.5 & \text{if } \mathcal{R} = \mathsf{lex} \\ 0.25 \times P(\mathtt{arg}(\mathsf{C_l}, \mathsf{C_r})) \times \frac{1}{1+\mathtt{forcomp}(\mathsf{C_h})} & \text{otherwise} \end{cases} \tag{6.10}$$

Logical Expression Base Distribution

Logical expressions are generated conditioned on CCG syntactic categories. Since each CCG syntactic category has a single semantic type associated with it, the logical expression base distribution needs to describe the probability of generating any allowable logical expression for a given type. The distribution used assumes that the model (and by extension the child) has access to all logical constants, predicates, functions, connectives and quantifiers in a set *consts*.

The type given by the syntactic category defines the number and type of lambda-bound variables at the root of the logical expression. These variables are created and added to a variable set *vars*. Then, the logical expression is generated by building a logical tree by recursively choosing to fill an empty argument slot with a node from either the set *vars* or the set *consts*. All nodes added to the tree must have the return type that matches the one required in that position by the node's parent. Each node added to the tree is added with a probability that reflects a draw from the uniform distribution over potential candidates. Conjunctions and disjunctions may take $2, \ldots, \infty$ arguments so the number of arguments n_{arg} is chosen with probability $1/n_{arg}$. Quantifiers and lambda-terms occuring within the tree may introduce new variables to the set *vars*. Due to the nature of the logical-forms in the training data and the semantic decomposition operations used, the number, order and type of lambda-terms occuring at non root positions in the logical tree is fully defined by the types of variables occurring higher in the tree. Therefore their introduction does not need to be modelled in the logical-expression base distribution.

Word Base Distribution

Words may contain characters from the Latin alphabet or spaces. A word containing c characters and s spaces has probability $\left(\frac{26}{26\times 500}\right)^c \times \left(\frac{1}{500}\right)^s$ under the base distribution over words. This reflects the belief that the child should prefer to not lexicalise multiword elements that cross many potential word boundaries.

6.5.3 The Dirichlet Process Posterior

The training algorithm presented in the next section will estimate, for each production head a, the posterior distribution over multinomial parameters θ_a^k that define the probabilities of each of the productions that can be used to expand a. These posteriors over multinomial parameters are themselves defined in terms of *pseudo-counts*. There is one pseudo-count $n_{a\rightarrow b}$ for each of the productions in the grammar. This pseudo-count describes the expectation of having seen $a \rightarrow b$ in the derivations used to parse observed training pairs $\{(s_i, \{m\}_i)\}$. Given the set of pseudo-counts $\{n^j_{a\rightarrow b}\}$ accumulated after the first j training instances, the parameter vector θ_a^k can be drawn from the *Dirichlet Process Posterior* as follows:

$$\theta_a^k \mid \{\ (s_i, \{m\}_i) \mid i = 1, \ldots, j\ \} \sim \tag{6.11}$$
$$\mathrm{Dir}(\alpha_a H(b_1) + n^j_{a\rightarrow b_1}, \ldots, \alpha_a H_a(b_{k-1}) + n^j_{a\rightarrow b_{k-1}}, \alpha_a H_a(\{b_k, \ldots\}) + \sum_{b'=b_k,\ldots} n^j_{a\rightarrow b'})$$

The posterior distribution over a finite subset of the parameters is, like the prior, Dirichlet distributed. This is because of the conjugacy between the Dirichlet prior and the Multinomial likelihood over possible expansions of the production head a.

6.6 Training the Model

The Bayesian model of the grammar presented in Section 6.5 can be used to define a joint distribution over the training data $\boldsymbol{D}$, the latent variables $\boldsymbol{S}$ that describe which productions were used in the derivations of each of the training pairs and the model parameters $\boldsymbol{\theta}$. At learning time we wish to estimate the posterior distribution over latent variables (parses) and model parameters. This posterior is, by Bayes' rule:

$$P(\boldsymbol{S}, \boldsymbol{\theta}|\boldsymbol{D}) = \frac{P(\boldsymbol{S}, \boldsymbol{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\boldsymbol{D})} \tag{6.12}$$

For models as complex as the one presented in Section 6.5 the posterior cannot be calculated analytically because the integral over latent variables and parameters required to calculate the data likelihood $P(\boldsymbol{D})$ is intractable.

A solution to this problem is to avoid the intractable integral over model parameters required in Equation 6.12 by approximating the posterior using the *mean field approximation* $Q(\boldsymbol{S}, \boldsymbol{\theta})$ that factorises over $\boldsymbol{S}$ and $\boldsymbol{\theta}$, allowing each to be treated separately:

$$P(\boldsymbol{S}, \boldsymbol{\theta}|\boldsymbol{D}) \approx Q(\boldsymbol{S}, \boldsymbol{\theta}) = Q_{\boldsymbol{S}}(\boldsymbol{S})Q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \tag{6.13}$$

We can then minimise the difference between $Q(\boldsymbol{S}, \boldsymbol{\theta})$ and the true posterior using the Variational Bayesian Expectation Maximisation (VBEM) algorithm (Beal, 2003).

Recent work by Kurihara and Sato (2006); Liang et al. (2007) has trained probabilistic grammars using variants of the VBEM algorithm. In the case of Liang et al. (2007) the probabilistic model was defined with the non-parametric DP prior. Other recent work has developed an online variant of the VBEM algorithm (Ghahramani and Attias, 2000; Sato, 2001; Beal, 2003; Honkela and Valpola, 2003; Hoffman et al., 2010; Wang et al., 2011) in which parameter updates are performed with respect to a subset of the training data.

In this section I describe the online VBEM algorithm and how it is applied to the model presentend in Section 6.5. Each of the parameter updates performed by the training algorithm is performed with respect to a single training instance $(s_i, \{m\}_i)$. The set of potential latent variable assignments for this training instance is contained in the set of parses $\{t\}_i$ proposed by $\mathcal{T}(s_i, \{m\}_i)$. At the end of this section I describe how the online VBEM parameter estimation procedure is integrated with the function $\mathcal{T}$ in an online training algorithm that learns the contents of a CCG lexicon Λ and the correct parameterisation of a CCG parsing model.

6.6.1 Online Variational Bayesian EM

Given the mean field approximation in Equation 6.13, our learning task is the one of minimising the difference between $Q(\boldsymbol{S}, \boldsymbol{\theta})$ and the true posterior $P(\boldsymbol{S}, \boldsymbol{\theta}|\boldsymbol{D})$. This difference can be measured using the Kullback Liebler (KL) divergence between the two distributions:

$$KL(Q(\boldsymbol{S}, \boldsymbol{\theta})||P(\boldsymbol{S}, \boldsymbol{\theta}|\boldsymbol{D})) = \int \sum_{\boldsymbol{S}} Q(\boldsymbol{S}, \boldsymbol{\theta}) \times \log\left(\frac{Q(\boldsymbol{\theta}, \boldsymbol{S})}{P(\boldsymbol{S}, \boldsymbol{\theta}|\boldsymbol{D})}\right) d\boldsymbol{\theta}$$

Which can be restated as the difference between the logarithm of the data likelihood and the *variational free energy* of $Q(\boldsymbol{S}, \boldsymbol{\theta})$.

$$\begin{aligned} KL(Q(\boldsymbol{S}, \boldsymbol{\theta})||P(\boldsymbol{S}, \boldsymbol{\theta}|\boldsymbol{D})) &= \log(P(\boldsymbol{D})) - \int \sum_{\boldsymbol{S}} Q(\boldsymbol{S}, \boldsymbol{\theta}) \times \log\left(\frac{P(\boldsymbol{S}, \boldsymbol{\theta}, \boldsymbol{D})}{Q(\boldsymbol{S}, \boldsymbol{\theta}))}\right) d\boldsymbol{\theta} \\ &= \log(P(\boldsymbol{D})) - F(\boldsymbol{D}, Q(\boldsymbol{S}, \boldsymbol{\theta})) \end{aligned} \tag{6.14}$$

Since $P(\boldsymbol{D})$ is fixed, reducing the KL divergence equates to maximising the variational free energy $F(\boldsymbol{D}, Q(\boldsymbol{S}, \boldsymbol{\theta}))$. Standard VBEM performs this maximisation in an iterative two step procedure that takes advantage of the factorised form of $Q(\boldsymbol{S}, \boldsymbol{\theta})$ and is very closely related to the Expectation Maximization algorithm (Neal and Hinton, 1998). In the VBE-step the distribution over parameters $Q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ is fixed and the free energy is maximised with respect to the distribution over latent variables $Q_{\boldsymbol{S}}(\boldsymbol{S})$. In the VBM-step $Q_{\boldsymbol{S}}(\boldsymbol{S})$ is fixed and the free energy is maximised with respect to $Q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$. Normal VBEM is a batch method. Each two-step update sees all of the training data at once. Online-VBEM breaks this dependence on total data observability by maximising an approximation to the free energy at each training instance on the basis of that training instance alone.

The total data free energy for the model presented in this chapter is calculated as a sum over all N training pairs $\boldsymbol{D} = \{(s_i, \{m\}_i)|i = 1, \ldots, N\}$ and all of the derivations

t in the parse forest $\{t\}$ proposed by $\mathcal{T}$ for each training pair.

$$F(\boldsymbol{D}, Q(\boldsymbol{S}, \boldsymbol{\theta})) = \sum_{i=1}^{N} \sum_{t \in \{t\}_i} \int Q(t)Q(\boldsymbol{\theta}) \times \log\left(\frac{P(t, \boldsymbol{\theta}, (s_i, \{m\}_i))}{Q(t)Q(\boldsymbol{\theta})}\right) d\boldsymbol{\theta} \quad (6.15)$$

However, online VBEM performs maximisation of the free energy with respect to a subset of the data. It does this by approximating the total data free energy by calculating the free energy with respect to the first j (observed) training pairs $\boldsymbol{D}_{1,\ldots,j} = \{(s_i, \{m\}_i) | i = 1, \ldots, j\}$ and rescaling it:

$$\begin{aligned} F(\boldsymbol{D}, Q(\boldsymbol{S}, \boldsymbol{\theta})) &\approx \left(\frac{N}{j}\right) F(\boldsymbol{D}_{1,\ldots,j}, Q(\boldsymbol{S}, \boldsymbol{\theta})) \quad (6.16) \\ &= \left(\frac{N}{j}\right) \sum_{i=1}^{j} \sum_{t \in \{t\}_i} \int Q_{\boldsymbol{S}}(t)Q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \times \log\left(\frac{P(t, \boldsymbol{\theta}, (s_i, \{m\}_i))}{Q_{\boldsymbol{S}}(t)Q_{\boldsymbol{\theta}}(\theta)}\right) d\boldsymbol{\theta} \end{aligned}$$

Online VBEM assumes that when the j'th training instance is seen, the distributions $Q_{\boldsymbol{S}}\{\boldsymbol{S}_{1,\ldots,j-1}\} = \{Q_{\boldsymbol{S}}(t) \mid t \in \{t\}_1 \cup \cdots \cup \{t\}_{j-1}\}$ over all latent variables used in the previous $j-1$ training instances are known; and that the distribution $Q_{\boldsymbol{\theta}}^{j-1}(\boldsymbol{\theta})$ over parameters has been determined for the previous $j-1$ training instances. Therefore, when the j'th training instance is seen, the sum in Equation 6.16 can only be maximised with respect to the latent variables $\{t\}_j$ used to model that instance, and the parameters $\boldsymbol{\theta}$.

Online-VBE step

In the online-VBE step, at training instance j, the parameter distribution $Q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ is fixed and the free energy is maximised with respect to $Q_{\boldsymbol{S}}(\{t\}_j)$. Fixing the distribution $Q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$ is equivalent to fixing the hyper-parameters $\boldsymbol{\alpha}_{j-1}$ of the Dirichlet priors over $\boldsymbol{\theta}$. Here, I assume that these hyper-parameters are known, I shall then explain where they come from later when discussing the online-VBM step.

Once the distribution over $\boldsymbol{\theta}$ has been fixed, the free energy is maximised with respect to $Q_{\boldsymbol{S}}(\{t\}_j)$. This is done by finding the distribution over derivations $\{t\}_j$ when they are scored using production weights $\hat{\boldsymbol{\theta}}_{j-1}$ that represent the *expectations* of the production probabilities under Dirichlet distributions parameterised by $\boldsymbol{\alpha}_{j-1}$. The expected value $\hat{\theta}_{a\to b}^{j-1}$ of $\theta_{a\to b}$ under a Dirichlet distribution parameterised by $\boldsymbol{\alpha}_a^{j-1}$ is the exponent of a difference of digammas (Beal, 2003, pp. 216). For each production $a \to b$ used in the parse forest $\{t\}_j$ the expected value $\hat{\theta}_{a\to b}^{j-1}$ of the parameter with which that production is scored is calculated as:

$$\hat{\theta}_{a\to b}^{j-1} = \frac{e^{\Psi(\alpha_{a\to b})}}{e^{\Psi\left(\sum_{\{b'\}}^{K} \alpha_{a\to b'}\right)}} \quad (6.17)$$

These expected parameter probabilities are used as production weights in an inside-outside algorithm run on a packed chart to get a packed representation of the distribution $\hat{Q}_{\boldsymbol{S}}(\{t\}_j)$ that maximises the free energy with respect to the latent variables.

Online-VBM step

The distribution over latent variables (derivations) $\hat{Q}_{\boldsymbol{S}}(\{t\}_j)$ calculated in the online-VBE step can be used to calculate the expectation of each production $a \rightarrow b$ being used in the derivation of the current training instance $(s_j, \{m\}_j)$.

$$E_{\hat{Q}_{\boldsymbol{S}}(\{t\}_j)}[a \rightarrow b] = \sum_{t' \in \{t\}_j} P(t'|s_j, \{m\}_j, \hat{\boldsymbol{\theta}}_{j-1}) \times \texttt{count}(a \rightarrow b \in t') \tag{6.18}$$

These production expectations are used by the online-VBM step to maximises the free energy with respect to the distribution over parameters.

The posterior distribution over parameters given in Equation 6.11 is defined in terms of pseudo-counts $\{n_{a \rightarrow b}\}$. The online-VBM step estimates the values of $\{n_{a \rightarrow b}\}$ that minimise the free energy approximation in Equation 6.16 with respect to $Q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$. The production expectation $E_{\hat{Q}_{\boldsymbol{S}}(\{t\}_j)}[a \rightarrow b]$ multiplied by N is the pseudo-count for $a \rightarrow b$ that would minimise the total data free energy if the training corpus consisted of $(s_j, \{m\}_j)$ repeated N times. This single instance estimate of the pseudo-count for $a \rightarrow b$ is used to calculate the actual pseudo-count stored after training instance $(s_j, \{m\}_j)$ by calculating its weighted average with the previous pseudo-count setting $n^{j-1}_{a \rightarrow b}$.

$$\begin{aligned} n^j_{a \rightarrow b} &= \frac{N}{j} E_{\hat{Q}_{\boldsymbol{S}}(\{t\}_j)}[a \rightarrow b] + \frac{j-1}{j} n^{j-1}_{a \rightarrow b} \\ &= n^{j-1}_{a \rightarrow b} + \frac{1}{j}(N \times E_{\hat{Q}_{\boldsymbol{S}}(\{t\}_j)}[a \rightarrow b] - n^{j-1}_{a \rightarrow b}) \end{aligned} \tag{6.19}$$

The pseudo-count update given above treats each training instance with equal weight. However, Sato (2001) points out that the early pseudo-count estimates are likely to be far less accurate than later ones which have access to a model trained on many previous training instances. For this reason it is desirable to use an adaptive learning rate $\eta(j)$ that is a function of the number of training instances seen. This learning rate can be tuned to dampen the effect of early estimations and also to *forget* early estimates late in training. With learning rate $\eta(j)$ the pseudo-count update for training instance j is:

$$n^j_{a \rightarrow b} = n^{j-1}_{a \rightarrow b} + \eta(j)(N \times E_{\{t\}}[a \rightarrow b] - n^{j-1}_{a \rightarrow b}) \tag{6.20}$$

And, following Hoffman et al. (2010), I use the learning rate:

$$\eta(j) \;=\; (\tau_0 + j)^{-\kappa} \tag{6.21}$$

In which the constant $\tau_0 \geq 0$ is used to decrease the effect of early estimates and the constant $\kappa \in (0.5, 1]$ allows later estimates to forget earlier ones.

The online-VBE step used fixed estimates of the hyper-parameters $\boldsymbol{\alpha}^{j-1}$ to fix the distribution $Q_{\boldsymbol{\theta}}^{j-1}(\boldsymbol{\theta})$ over production parameters. The individual hyper-parameter $\alpha^j_{a\rightarrow b}$ for the production $a \rightarrow b$ at time step j is calculated after the online-VBM step:

$$\alpha^j_{a\rightarrow b} \;=\; \alpha_a H_a(b) + n^j_{a\rightarrow b} \tag{6.22}$$

Where α_a and H_a are the concentration parameter and base distribution introduced in Section 6.5 and $n^j_{a\rightarrow b}$ is the pseudo-count accumulated for the production $a \rightarrow b$ over the course of the previous j observed training instances.

Sato (2001) points out that in online VBEM the approximated free energy is not guaranteed to increase with each update as a new contribution is added at each time instance. However, he also shows that the online VBEM can be viewed as a stochastic approximation for finding the maximum of the expected free energy. The online update schedule presented here knows, prior to training, the amount of training data that will be seen N. This is a small contravention of the conditions required for the algorithm to be truly online and Hoffman et al. (2010) points out that in the truly online case, as $N \rightarrow \infty$, this algorithm corresponds to an empirical Bayes estimator of the latent variables. In practice, the value of N is used by the algorithm to weight the effects of the prior against those of the observed data and need not exactly match the amount of data to be seen.

It should also be noted that, although an infinite Dirichlet process prior is used to model the distribution over parameters, only a finite subset of these parameters are ever used since each training instance supports only a finite number of parses. This contrasts with the infinite PCFG of Liang et al. (2007) which requires the infinite prior to be truncated during variational inference.

6.6.2 The Training Algorithm

Now the training algorithm used to learn the lexicon Λ and pseudocounts $\{n_{a\rightarrow b}\}$ can be defined. The algorithm, shown in Algorithm 8, passes over the training data only once and one training instance at a time. For each $(s_i, \{m\}_i)$ it uses the function $\mathcal{T}$ $|\{m\}_i|$ times (once for each logical-form in the context set) to generate a set of allowed parses $\{t\}'$.

```
Input : Corpus $D = \{(s_i, \{m\}_i) | i = 1, \ldots, N\}$, Function $\mathcal{T}$, Logical-form
        to syntactic category mapping cat, function lex to read lexical
        items from derivations.
Output: Lexicon $\Lambda$, Pseudocounts $\{n_{a \to b}\}$.
$\Lambda = \{\}, \ \{t\} = \{\}$
for $i = 1, \ldots, N$ do
    $\{t\}_i = \{\}$
    for $m' \in \{m\}_i$ do
        $\mathsf{C}_{\mathsf{m}'} = \mathtt{cat}(m')$
        $\{t\}' = \mathcal{T}(s_i, \mathsf{C}_{\mathsf{m}'} : m')$
        $\{t\}_i = \{t\}_i \cup \{t\}', \ \{t\} = \{t\} \cup \{t\}'$
        $\Lambda = \Lambda \cup \mathtt{lex}(\{t\}')$
    end
    for $a \to b \in \{t\}$ do
        $n^i_{a \to b} = n^{i-1}_{a \to b} + \eta(i)(N \times E_{\{t\}_i}[a \to b] - n^{i-1}_{a \to b})$
    end
end
```

Algorithm 8: Learning Λ and $\{n_{a \to b}\}$

The lexicon is populated by using the `lex` function to read all of the lexical items off from the derivations in each $\{t\}'$. In the parameter update step, the training algorithm updates the pseudocounts associated with each of the productions $a \to b$ that have ever been seen during training according to Equation (6.20).

The model only stores non-zero pseudocounts. The count vector is expanded with a new entry every time a new production is used. While the parameter update step cycles over all productions in $\{t\}$ it is not neccessary to store the set $\{t\}$ itself, just the set of productions that are ever used in $\{t\}$. In practice, the training algorithm also prunes productions from this set if their pseudo-counts are less than 10^{-3} of the total pseudo-count mass associated with the production head. Furthermore, most productions in $\{t\}$ will not be seen in any one $\{t\}_i$. The pseudo-count updates for those productions not in $\{t\}_i$ merely discount the parameters for those productions according to the predefined learning rate. In the interests of efficiency these discounting updates are postponed until the production is required in a parse.

6.7 Experimental Setup

Sentence Segmentation The majority of the previous computational work on modelling language acquisition (properly reviewed in Section 6.10) assume that the correct segmentation of the uttreances is known. Experiments by Saffran et al. (1996); Mattys et al. (1999); Mattys and Jusczyk (2001) support the claim that children have the ability to locate candidate word-boundaries by the time that they start learning syntax. However, it is less clear that these children are able to identify individual words as lexicalizable units. For this reason I allow the lexicalization of all subspans of the sentence. Although, as this significantly increases the search space and slows learning, I also present an evaluation of the model without multiword lexical items.

Maximum Utterance Length The learning procedure only operates on utterances of 10 or fewer words. This is done in order to speed learning and neglects very few potential training pairs, as discussed in Appendix A.1.2.

Propositional Uncertainty The learning task facing the child requires that the child should be able to learn syntax and semantics without knowing for certain what each utterance means when she hears it. I approximate the *propositional uncertainty* facing the child with a *context set* of candidate logical-forms $\{m\}$. This context set is created by pairing each utterance with the correct logical-form and the $|\{m\}|/2 - 1$ preceding and succeeding logical-forms. Selecting the $|\{m\}| - 1$ distracting logical-forms from the same time frame as the utterance is intended to guarantee that all logical candidates are drawn from a similar context.

Word Meaning Guessing The Eve data contains many words that occur only once. In order to alleviate the effect of unknown words when parsing a new sentence I allow the parser to guess the meanings and CCG syntactic categories of unknown words. This is done by first drawing a CCG syntactic category C from a a distribution over lexicalised categories, then drawing a *shell* logical-form with the correct type from the distribution $P(m^{sh}|\mathsf{C})$. Shell logical-forms are logical-forms in which the named logical constants are replaced with placeholders (e.g. $\lambda x.pred(x)$ is the shell logical-form of $\lambda x.dog(x)$). The unknown word dax could be assigned the semantics $\lambda x.pred(x)$ by first predicting that it is a noun and then predicting the logical expression $\lambda x.pred(x)$ conditioned on the category N.

$$\mathrm{dax} \ \rightarrow_{P(\mathsf{N})}\rightarrow \ \mathsf{N} \ \rightarrow_{P(\lambda x.pred(x)|\mathsf{N})}\rightarrow \ \lambda x.pred(x)$$

Each unknown word is paired with the 10 most likely lexical items predicted in this way and all of these new lexical items are scored according to the base distributions defined above. I report results with and without word-meaning guessing. When word-meanings are guessed, the logical-form returned by the parser is considered to be correct if it matches the gold-standard in all ways other than the naming of the shell logical constants.

The UBL Baseline I compare the model of language acquisition to the UBL system presented in Chapter 4. In order to present both systems with a comparable learning task, I do not initialise UBL with either the NP-list or word-logical constant co-occurrence scores used when learning a semantic parser. I also report results for UBL run (as the model of language acquisition) with a single pass over the data (UBL^1) and when trained for 10 iterations (UBL^{10}). It should be noted that UBL was designed to efficiently and greedily learn a suitable lexicon from potentially very complex (sentence, meaning) pairs whereas the model presented here examines the entire set of possible lexical items allowed by the training data. However, UBL suffices as a reasonable baseline against which to evaluate the new approaches success at learning a parsing model.

6.8 Evaluating the Learner

The system presented in this chapter is capable of learning a probabilistic grammar that will parse sentences onto logical forms. In this section I present a set of evaluations of the model trained on the logically annotated Eve data. First I show that the model of language acquisition (henceforth LanAcq) can learn a probabilistic parser that returns correct logical forms for new unseen sentences. I compare LanAcq to the UBL system from Chapter 4 and show that it outperforms UBL when neither are initialised with language specific information. Then, having evaluated the parser learnt by LanAcq on its ability to parse new sentences, I perform a more qualitative evaluation of some of the subtasks performed by the learner: word-meaning learning; and word-order learning. I examine the manner in which a number of word-meanings and syntactic regularities are learnt and show that they compare favourably to observations of language learning in children, avoiding criticisms levelled against previous statistical models of language acquisition.

6.8.1 Learning a Parsing Model

The Eve corpus is split into 20 temporally ordered files collected over the course of 9 months. While the contents of these files does not represent the entire linguistic input available to Eve, they do provide an approximation to the increasingly complex utterances to which she was exposed. I maintain the temporal order of the data and train the model in a single pass. This ordered training provides for a testing scenario that approximates the task facing Eve: the model is trained on an ordered subpart of the corpus (representing Eve's linguistic input up to a certain age) and then tested on the next portion of the corpus (representing the unseen language that Eve is learning to parse). I evaluate the parsing model by training it on the first n files and testing on file $n + 1$ (collected at a later age).

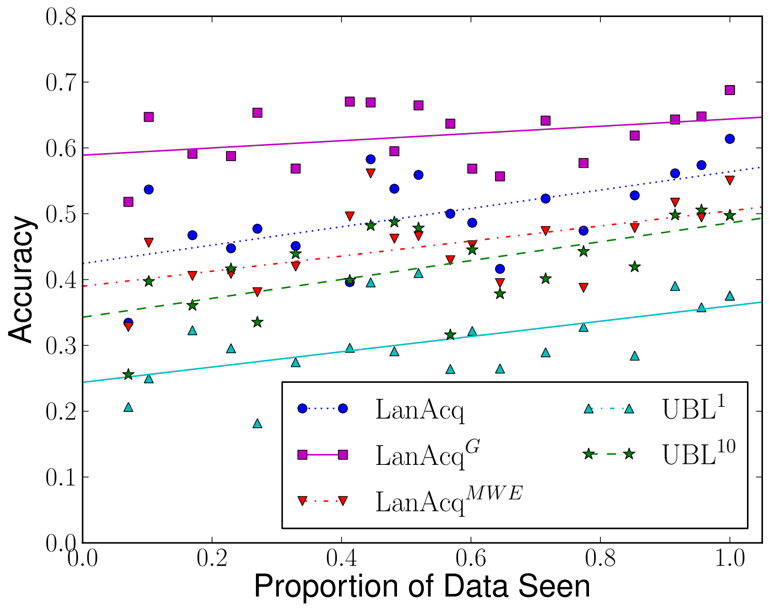


Figure 6.2: Parse Accuracy: Train on files $0, \ldots, n$ test on file $n + 1$

The performance of the model of language acquisition with known word boundaries (LanAcq), the model of language acquisition with known word-boundaries and word-guessing (LanAcq^G), the model of language acquisition with unknown word-boundaries (LanAcq^{MWE}), and UBL with both training scenarios (trained for 1 and 10 iterations) are illustrated in Figure 6.2. The accuracy illustrates the proportion of utterances in the current file $n + 1$ for which the parsing model trained on the previous n files returns the correct logical-form. All of these parsers have been learnt from data in which each utterance is paired with a single logical-form. LanAcq outperforms both UBL^1 and UBL^{10} with and without word guessing and known word-boundaries. While LanAcq only passes over the data once, it hypothesises all parses of all training instances visiting the entire lexical space consistent with the training corpus. Doing this allows LanAcq to learn a better, more consistent, lexicon than even UBL^{10} which performs a greedy search in this lexical space and may never hypothesise many correct lexical items. When trained with only one pass over the data, UBL cannot learn many lexical items at all due to its incremental splitting approach. On early test files in particular UBL^1 can only correctly parse utterances that it has seen before. Even a simple baseline that memoizes previously seen utterances achieves an accuracy of 29.7% across all 19 tests. This is due to the high degree of repetition in child directed speech. However, this repetition is highly concentrated in a few repeated sentence types—42% of repeated utterance occurrences are accounted for by only five distinct utterances: *"that's right", "what is that", "what's that", "what are you doing", "what is it"*. In both UBL^1 and UBL^{10}, the UBL system is being run under conditions for which it was not designed. When run with parameters initialised according to co-occurrence scores between logical constants and words (as described in Section 4.6) UBL achieves an average recall of 47.0% over all tests compared to 45.0% for LanAcq^{MWE} which has the most similar lexical search space. It should also be noted that LanAcq requires far more computational resources than UBL and cannot therefore be usefully run on either the GeoQuery or Atis datasets presented in Chapters 4 and 5.

Despite the frequent repetition of a few utterance types, the parsing task illustrated in Figure 6.2 is hard due to data sparsity. 32.9% of utterances parsed in testing contain a word that has never been seen before and another 11.1% of sentences contain a word that has only been seen once. This explains the great positive effect of word-guessing particularly in the earlier stages of training. It also motivates the requirement for a learning procedure that can learn word-meanings on the basis of one or very few observations. This *fast mapping* ability has been recognised in child language acquisition and is examined in greater depth later. The performance of all approaches illustrated in Figure 6.2 increases over time as more data is seen. This is especially true for those

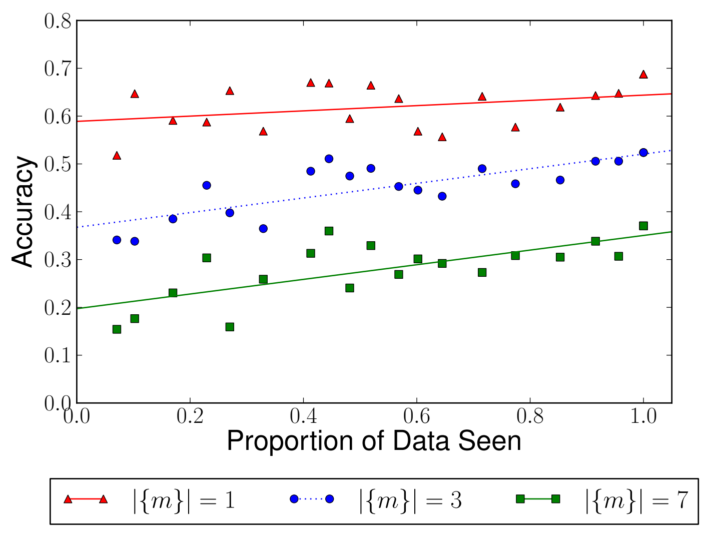


Figure 6.3: Parse Accuracy for $\mathrm{LanAcq}^{\mathrm{G}}$ with Propositional Uncertainty

models that do not have access to word-guessing. The effect of word guessing decreases for later files as a smaller proportion of the words are unknown, and also the utterances in the later files are more complex. Figure 6.3 illustrates the performance of $\mathrm{LanAcq}^{\mathrm{G}}$ when trained on data in which each utterance is paired with a context set $\{m\}$ of varying size. Even with 3 possible interpretations for each utterance $\mathrm{LanAcq}^{\mathrm{G}}$ learns a parser of similar quality to UBL^{10}. With 7 interpretations per training utterance, the parser performance is understandably lower but increasing over time as the model accumulates evidence. UBL cannot, in its current form, be trained with multiple logical-forms per training instance.

6.8.2 Word Learning

The most studied aspect of the language acquisition task from the perspective of computational modelling is *word learning* - learning the meanings of words. The word learning task is easily observable and lends itself to psychological experiments in which children are presented with unknown words and their reaction is observed. One of the strongest conclusions to have been drawn from these experiments is that children are capable of *fast mapping* (Carey and Bartlett (1978) and much subsequent work). Fast mapping is the process by which a new word of a type that has been seen before is learnt on the basis of one, or very few exposures. Figures 6.4 and 6.5 show the marginal probabilities of the correct semantic analyses for a selection of quantifiers and nouns that are calculated using parameter values representing the mode of the distribution

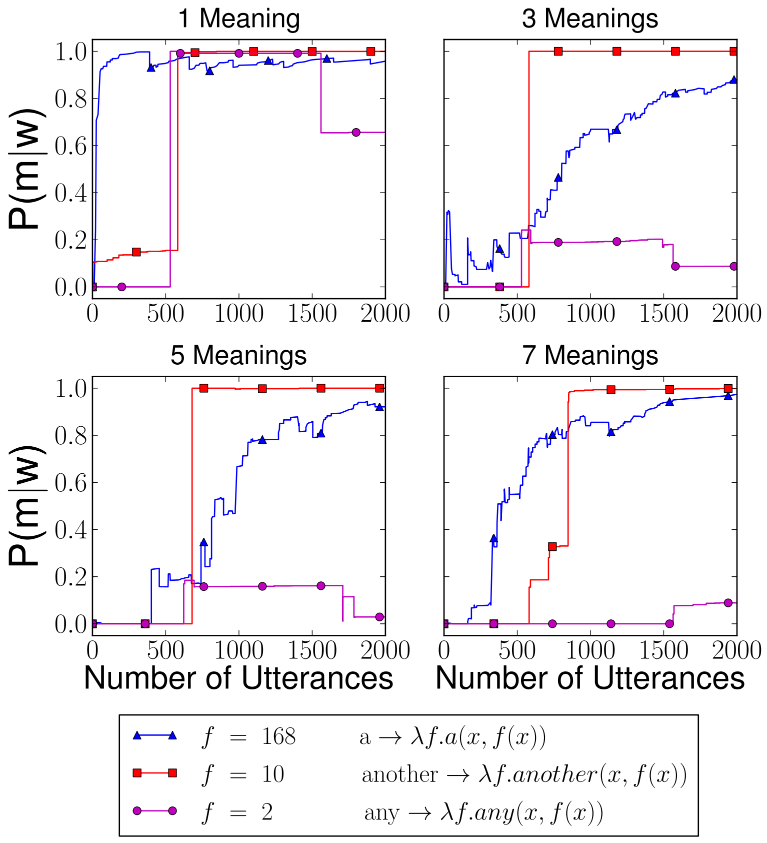


Figure 6.4: Probability of correct meaning for selection of quantifiers over time

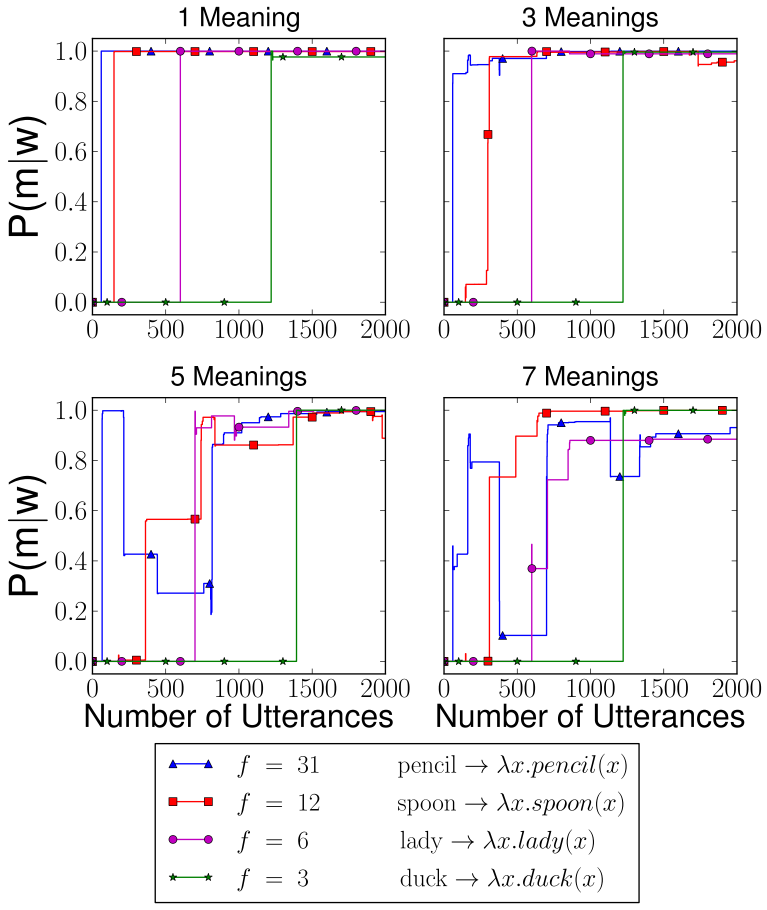


Figure 6.5: Probability of correct meaning for selection of nouns over time

over parameters. The words covered in these plots have widely varying frequencies f and each of the plots illustrates learning from data in which each utterance is paired with a context set containing 1, 3, 5 or 7 logical-form candidates. All markers in these plots are designed to ease viewing in black-and-white. Their position is meaningless.

In Figure 6.4 the very commonly occurring quantifier ‘a’ is learnt gradually over the course of training whereas the rarer ‘another’ appears to be learnt with a big jump in probability on the basis of a single exposure (with context sets containing 3 and 5 logical-forms) or a few exposures (with context sets containing 1 and 7 logical-forms) and the rarer still ‘any’ does not seem to be learnt at all over the course of the first 2000 training instances when there is any degree of propositional uncertainty.

In the model presented here, fast mapping can be represented as a sudden jump in the probability of a word analysis on the basis of a single training pair. If we assume that any word analysis with probability > 0.8 has been learnt, it is clear that in all training scenarios, at least some words are being learnt through a procedure that can be viewed as *fast mapping*. LanAcq’s ability to mimic fast mapping is most obvious in Figure 6.5. Here many of the nouns are learnt via a sharp increase in probability on the basis of a single exposure. This is due to the effect of *structural bootstrapping* from knowledge about syntax and word-meanings in the model. There is a virtuous interaction between commonly seen structures which accumulate high probability and the low frequency structures with which they are seen. If a noun is seen with a well known determiner and verb, the parsing model is likely to assign high probability to the derivation in which the noun has the correct logical interpretation. This high probability derivation will then result in a large pseudo-count update for that word analysis in LanAcq’s parameter update step. The best illustrated example of this is the word ‘duck’ which is learnt on the basis of a single example in all training scenarios. This example is the sentence “that’s a duck” for which the parsing model already has very strong beliefs about the correct analysis. Conversely, if the parsing model does not know about the syntactic and semantic context in which the word occurs, LanAcq cannot learn the correct word meaning. This is why the word ‘any’ is not learnt when there is propositional uncertainty—because it co-occurs with the new nouns ‘bread’ and ‘squirrel’ and the purely *syntactic bootstrapping* effect of quantifier-noun order is not strong enough to force the correct analysis over the analysis that lexicalises the quantified noun (e.g. $any(x, squirrel(x))$). Over time, as evidence for the correct analysis accumulates it will, however, be learnt correctly.

It should be noted that, since these graphs present probabilities calculated using parameters that represent the mode of the distribution over parameters, they do not illustrate the model's certainty in any one analysis. Rather they are intended to show that significant and sharp changes in the predictive probability are possible. Many of the word meanings learnt on the basis of a few observations can be unlearnt again on the basis of a few more. This is evident from the instability of the predictive probability for the word 'pencil' when LanAcq is trained with context sets containing 5 and 7 logical-forms and also for the word 'any' when learnt without propositional uncertainty.

This word-learning analysis does not give a quantitative measure of LanAcq's success but it does show that the model is, in general, capable of learning big changes in local predictive distributions on the basis of a few (possibly uncertain) observations—correlating with observations of word-learning in children.

6.8.3 Word Order Learning

Figure 6.6 illustrates the probabilities of the 6 possible transitive verb orders. These are calculated by summing over all lexical items containing transitive verb semantics and sampling in the space of parse trees that could have created them. With no propositional uncertainty the correct SVO word order is learnt very quickly and stabilises. As the degree of propositional uncertainty increases, the rate at which this word order is learnt decreases. However, even in the face of 6 distracting logical-forms per training utterance, LanAcq can learn the correct word order. The distribution over word orders also exhibits initial uncertainty followed by a sharp convergence to the correct choice. One should be careful in forming comparisons between this depiction of syntactic learning and targeted evaluations of a few childrens' acquisition of complex linguistic phenomona. However, the recovery from initial incorrect decisions (e.g. from having learnt OVS word order early in training) correlates with Crain and Thornton (1998)'s demonstration, through the use of elicited utterances, that a single child may believe in many different word orders before settling on the correct hypothesis for the target language. The abrupt manner in which LanAcq can acquire belief in the correct word order means that it is not subject to the criticisms that Thornton and Tesan (2007) levelled at statistical models of language acquisition in general (and that of Yang (2002) in particular)—that their learning rates are far more gradual than those observed in children.

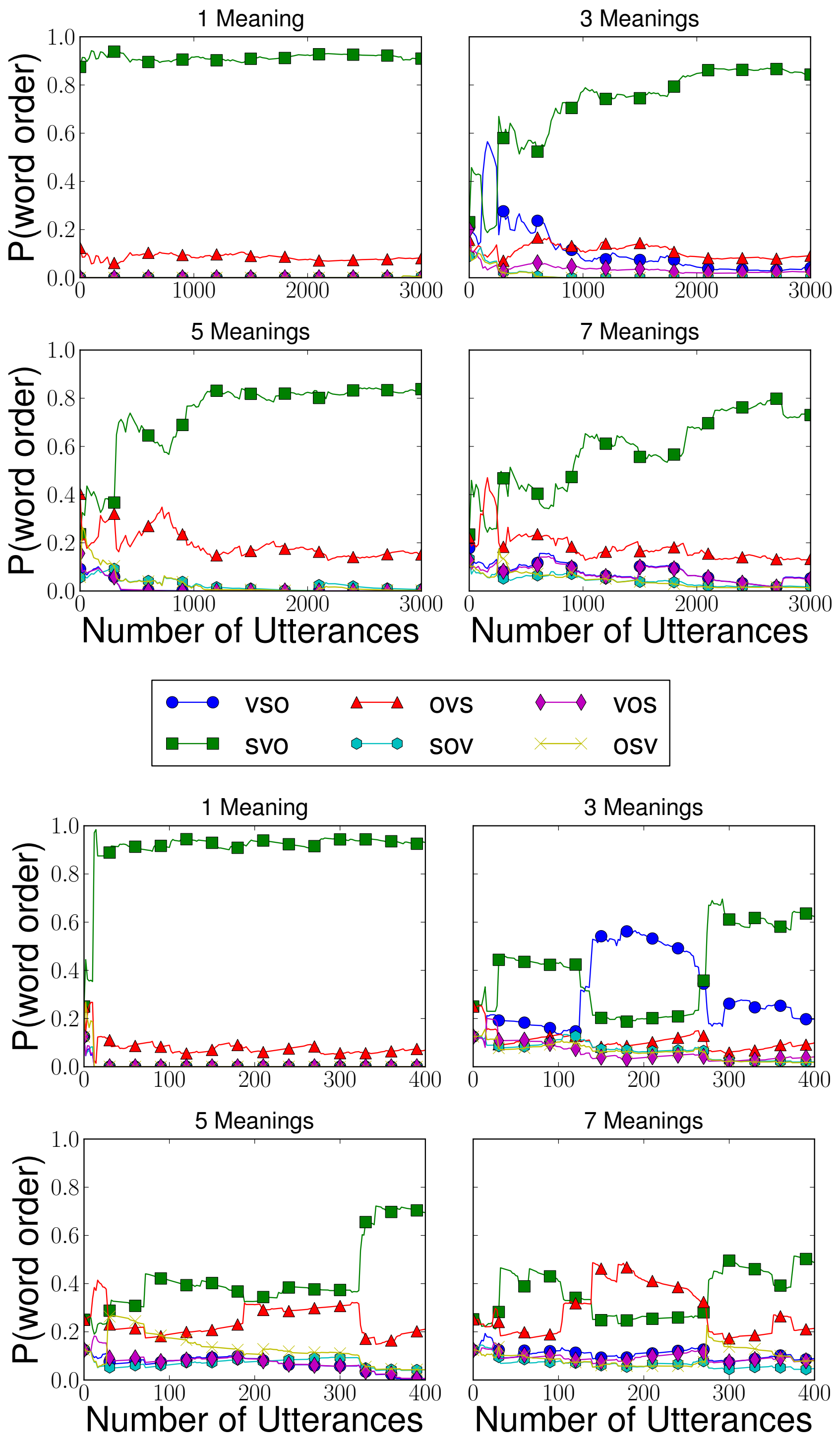


Figure 6.6: Probability of order of subject object and verb over time

6.9 Discussion and Future Work

The previous section analysed the model of language acquisition on the basis of its ability to learn a parser that can return logical-forms representing the meanings of child directed utterances. The generative probabilistic model of the grammar can also be used to produce (sentence, logical-form) pairs or sentences from a logical-form input. The learning approach could, in theory, be evaluated on the basis of its ability to learn a system of sentence generation that mirrors that of the child. This evaluation is very appealing as we can observe the child's productive ability and can therefore compare the model's competence directly to the child's competence. But it is also very hard to draw conclusions from, using the current data and evaluation metrics available to us. As the data in the Eve corpus is collected over a few short timespans, the overlap between child-spoken and child-directed language is not complete. Any evaluation based on child directed utterances should therefore distinguish between phenomona arising as a result of the learning procedure and those arising as a result of incomplete data. Any evaluation of the generation task should also focus on the grammatical rules and regularities of interest. Current machine translation metrics do not provide such an analysis.

The qualitative discussion of LanAcq's word-order learning ability was linked to experimental observations of syntactic acquisition in children by Crain and Thornton (1998) and Thornton and Tesan (2007). This link was made with a caveat—the rules illustrated did not describe the particular syntactic phenomona observed. This is because it is difficult to formalise marginal probability distributions that describe the complex syntactic parameters (such as Inflection-Negation and V2) focused on by the experimental literature and previous computational models of syntactic acquisition by e.g. Gibson and Wexler (1994); Sakas and Fodor (2001); Yang (2002); Villavicencio (2002) (all reviewed in the following section). Future work could focus on developing evaluations that measure the model's ability to correctly learn the information encoded in these parameters. Such an evaluation could potentially be made through an empirical observation of the syntactic variation in the distribution of sentences generated by the probabilistic grammar.

Further qualitative analysis of learning rates could also examine different well attested phenomona in child langauge acquisistion such as U-shaped learning (reviewed in Marcus et al. (1992)). U-shaped learning, most commonly seen in the acquisition of tensed verbs, occurs when children first learn all tensed verbs separately (and therefore correctly), then learn and over-use the rules governing tense such as adding 'ed' to signal past tense in English (resulting in less correct productions corresponding to the bottom of the U), then finally learning all irregular verbs separately from positive evidence of their existence. While the model that I have presented here does not support morphology, it does follow a trajectory in which it first memoizes large complex structures, then learns to use (and possibly over-use) rules. As the general structure of this model could be adapted to support a simple model of morphology, future experiments could examine the ability of the learning algorithm to model the U-shaped learning curves described above.

Although the probabilistic model is Bayesian, the overall approach is different from that used to train many of the Bayesian models proposed in cognitive science and language acquisition (Xu and Tenenbaum, 2007; Goldwater et al., 2009; Frank et al., 2009; Griffiths and Tenenbaum, 2006, 2005). These models are intended as *ideal observer* analyses, demonstrating what would be learned by a probabilistically optimal learner. The learner presented here uses more cognitively plausible but approximate online learning algorithm, and thus has no guarantee of optimality. In that it sacrifices optimality for cognitive plausibility, the learning approach can be related to other recent work in investigating cognitively plausible approximate Bayesian learners by Pearl et al. (2010); Sanborn et al. (2010); Shi et al. (2010), although that other work also enforces strict limitations on processing power and memory.

Although the learning algorithm is online and has no language specific initialisation there are still aspects that are cognitively implausible. In particular, the training algorithm generates all parses consistent with each training instance, which can be both memory and processor intensive for longer sentences. It is also unlikely that children do this once they have learnt at least some of the target language as this knowledge will persuade them to eschew some of the less probable analyses. Future work could retain the core function of parse proposal with a restricted version of $\mathcal{T}$ but by using more efficient parameter estimation methods could avoid the enumeration of all parses. One possibility would be an approximate online VBEM algorithm in which the expectations in Equation 6.20 are calculated according to a high probability subset of the parses $\{t\}$. Another option would be particle filtering, which several authors have investigated as a cognitively plausible method for approximate Bayesian inference (Levy et al., 2009; Sanborn et al., 2010; Shi et al., 2010).

Despite the limitations mentioned above, the system presented here makes an important contribution to task of modelling language acquisition computationally. It is the first to learn syntax and semantics concurrently from realistic data. The systems presented by Villavicencio (2002) and Buttery (2006) learnt categorial grammars from sentences where all word meanings were known and those of Siskind (1992) learnt only from a tiny toy corpus with unscalable computational methods. The system that I have presented is also the first model of language acquisition to be evaluated by parsing sentences onto their meanings, in contrast to the work mentioned above and that of Gibson and Wexler (1994); Siskind (1992); Sakas and Fodor (2001); Yang (2002). These all evaluate their learners on the basis of their ability to set a small number of predefined syntactic parameters. The system presented here represents the grammar as a large number of rules, each with an associated probability. This view of grammar contrasts with that taken by all of the approaches mentioned above, all of which aim to learn a single deterministic representation of the grammatical rules allowed.

Finally, this work addresses a misunderstanding about statistical learners—that their learning curves must be gradual (Thornton and Tesan, 2007). By demonstrating sudden learning of word order and fast mapping, our model shows that statistical learners can account for sudden changes in children's grammars. Future work could extend these results by examining other learning behaviors and testing the model on other languages.

6.10 Related Models of Language Acquisition

This section presents an overview of previous attempts to model a child's acquisition of syntax, word-meanings or both syntax and word-meanings. All of these approaches treat the tasks of word-meaning learning and syntax learning separately—either modelling one of the two tasks or modelling the integration of both in a pipelined system. When both are learnt they are learnt with distinct, task specific, mechanisms. This is in contrast to the unified approach presented here—where all syntactic and lexical knowledge is learnt as part of a single process.

The majority of previous models of word-meaning learning represent the meaning of a context as a pre-segmented bag of semantic symbols. The only previous work that has learnt word meanings from unsegmented logical-forms is that of Siskind (1996). The majority of previous syntax learning approaches express the syntactic grammar in terms of a pre-defined set of syntactic parameters in the principles-and-parameters framework of Chomsky (1981) that was introduced as a means of formulating a compact abstraction over diverse linguistic phenomena across languages. The parameter setting account of language acquisition (c.f. Hyams (1986)) poses the problem of learning a language as the one of correctly setting a few 'switch-like' parameters that control the range of allowed syntactic constructions. The algorithms of Gibson and Wexler (1994); Sakas and Fodor (2001); Yang (2002); Villavicencio (2002); Buttery (2006) all aim to learn the setting of a number of 'switches' defining a deterministic grammar that correctly covers the target language. The probabilistic approach set forward in this chapter aims, instead, to learn a distribution over possible derivations. If the grammaticality of a sentence with respect to a certain language ever needs to be judged, this would be done by placing a threshold on the probability of 'grammatical' sentences. It is not, however, clear why one would ever want a binary indication of grammaticality when a continuous probabilistic indication of sentence quality is available.

In this section I describe these previous models of language acquisition, dividing them into: those that model the acquisition of syntax from *triggering sentences* in which there is explicit information governing the setting of one or more syntactic parameters; those that learn word-meanings and syntax from sentences paired with compositional logical-forms; those that learn word-meanings from sentences paired with sets of semantic symbols; and those that learn syntax from sentences alone.

6.10.1 Learning from Triggers

There have been a number of algorithms developed to learn grammars that have been parametrically defined in the tradition of the Principles and Parameters theory (Chomsky, 1981) from explicit triggers apparent in the surface form of input sentences. Gibson and Wexler (1994) present the Triggering Learning Algorithm (TLA) which is then improved upon by the Structural Triggers Learner of Sakas and Fodor (2001). Both of these algorithms assume that there is some form of linguistic 'trigger' set available to the child from the strings of the language. Sakas and Fodor (2001) admit that the concept of triggers is hard to nail down but explain them as "A word string of the target language, perceived by the learner, which 'automatically' flips a parameter switch to the correct value." All of the models reviewed in this subsection assume that the language learner knows enough about the syntactic categorisation of words in a sentence to recognise a trigger despite not having yet learnt a parsing model that could project the sentence onto its underlying semantics. As an example, in order to learn the parameters required to license SVO word order in English, the learner needs not only to encounter a transitive verb-phrase but also to label each of the three words in this phrase with its correct thematic role. The TLA, STL and variational approach of Yang (2002) presented below all assume that this information is given by some process that available to the child prior to acquisition of the syntax.

This assumption seems far-fetched. The learner should reasonably be expected to learn both the parsing rules and categorisation of the words. And the only reasonable candidate for something that gives the information required to categorise words is semantics. I refer to the assumption that the word categories are known as the *a priori assumption of syntactic classification* and view it as a key weakness of all trigger based accounts of language acquisition that do not also account for the acquisition of word categories. The unified approach used by LanAcq learns word categorisation and syntax at the same time.

The TLA of Gibson and Wexler (1994) incrementally learns a three-parameter binarised parametric grammar on the basis of unparseable sentences. Each unparseable sentence is treated as a trigger, starting a stochastic search through the parameter space which proceeds by randomly toggling each parameter and accepting the change if the resulting grammar manages to parse the hitherto unparseable sentence. This approach has a key drawback, in addition to its reliance on the a priori assumption of syntactic classification, in its search strategy which easily gets stuck in local maxima and suffers from ambiguity in the triggers. The problem of ambiguity in the triggers is solved by Sakas and Fodor (2001) who force their STL to learn the smallest correct grammar at all points by enforcing the *subset principle*. The subset principle states that, when faced with a trigger, the learner should always choose the Universal-Grammar compatible language that fulfils the requirements of the trigger and simultaneously covers the smallest superset of the input sentences seen to date. This enforces a single choice for each trigger but the problem of local maxima remains.

Yang (2002) presents a “variational” model of syntactic acquisition designed to avoid the problem of getting stuck in local maxima as a result of misleading triggers. The term variational in this sense is used to contrast Yang’s approach with transformational learning models such as the TLA and STL which maintain a single hypothesis grammar. Instead, Yang considers a “variational theory in which language acquisition is the change in the distribution of I-language grammars, the principled variations in human language" (Yang, 2002, pp. 23)[2]. The variational approach presented by Yang uses the same notion of grammar as the approaches of Gibson and Wexler (1994) and Sakas and Fodor (2001) but poses the task of language acquisition as one of grammar competition. The learner has access, during learning, to many differently parameterised grammars, each of which is associated with a weight. The current highest weighted grammar is used to parse each new training sentence and all grammars are then re-weighted according to a linear reward which rewards the current grammars weight (and penalises all others) if it is successful and penalises the current grammars weight (and rewards all others) if it is not.

This variational approach is far more robust to noise than deterministic trigger based approaches but it still requires the a priori assumption of syntactic classification. Furthermore, for a grammar space with n parameters, the variational approach requires 2^n distinct grammars to compete. For any realistic value of n (and Fodor (1998) suggests that at least 20 are required), this approach will quickly become intractable as each weight update is performed on the basis of an observation of the competence of the current highest scoring grammar alone (despite the fact that it is applied to all grammars). This learning strategy has the advantage that the calculation of the learning update resulting from a single sentence is very computationally cheap (requiring only a single attempt to parse the sentence under the current favourite deterministic grammar). However, the treatment of the many possible grammars as separately defined structures necessarily results in a very slow learning procedure which is, over the course of learning an entire language, computationally intensive. Yang 2002, pp.31 admits this problem and suggests two possible solutions, one that borrows the STL's mechanisms of enforcing the subset condition and one that uses ordered *learning cues* to limit the set of parameters that may be set. However, acknowledging further problems with both solutions, Yang sticks with the slow learning *Naive Parameter Learner* that treats grammars distinctly. This variational learner has been criticized by Thornton and Tesan (2007) who observed radical 'step like' changes in the grammars of four children that do not correlate with the gradual projected learning rates presented by e.g. Yang 2002, pp. 30. This criticism is not one that can be applied to the Bayesian learner presented in this chapter as I showed in Section 6.8.

[2]Note that the term 'variational' in this sense is distinct from its use in 'variational-Bayes'. In the sense of variational-Bayes the word variational comes from the use of the calculus of variations to show that the maximisation of the variational approximation to the free energy is equivalent to the minimisation of the KL divergence between the variational approximation and the true posterior.

6.10.2 Learning from Sentence and Logical-Form

Models of language acquisition from sentences paired with representations of their meaning do not necessarily require the a priori assumption of syntactic classification as they can identify thematic roles by first learning an alignment from word to word-meaning.

Siskind (1992) presents a unified model of word and syntax learning from a small artificial corpus of (sentence, logical-form) pairs. This system learns a cascaded parser architecture that proposes parse structures using a parameterised $\bar{X}$ theory and then refines the set of allowed structures with separate modules that model movement, case structure and theme-referent assignments. The grammar presented by Siskind is deterministic and assumes that all words are monosemous. The set of parameters determining a word's meaning and use in the grammar are learnt from a corpus of (sentence, logical-form) pairs via a divide-and-conquer search strategy that finds, for each training instance, the set of parses consistent with the current parameterisation of the grammar and learns from these a superset of the current parameter set that allows the training instance to be parsed. This superset is not necessarily unique, leaving the approach open to the *subset problem*. Furthermore, (Siskind, 1992, pp. 80) admits that a number of ad-hoc restrictions were necessarily adopted as part of his linguistic theory in order to facilitate the search strategy. Siskind evaluates his approach by comparing the set of learnt grammatical parameters to a gold-standard target. He shows that it is possible to learn word categorisation and word-order rules in a number of cases but that the grammar learnt, while able to parse the artificial training corpus, is not exactly the same as the target English grammar (also able to parse the training corpus).

Waldron (1999) presents a system for learning categorial grammars from utterances paired with a set of associated semantic predicates and syntactic primitives. Villavicencio (2002) and Buttery (2006) extend this system into a more complete model of language acquisition from utterances paired with logical-forms. Waldron's system assumes that syntactic categories are known for words with primitive categories (such as N, NP). Then syntactic parses are proposed for sentences in which all word-meanings are known. This is done by hypothesising the set of non-primitive category assignments that could be used to piece together a parse that includes the known category assignments. Preferred word categories are learnt by accumulating statistics over a training corpus. However, since the syntactic categories assigned to words are unconstrained by the type of that word's logical-form, it is common for this approach to learn highly complex hypotheses for simple words. Villavicencio 2002, pp. 145 illustrates this deficiency with the example word-category pair $\mathrm{red} \vdash (((\mathsf{S}\backslash\mathsf{NP})\backslash(\mathsf{NP}\backslash\mathsf{N}))\backslash((\mathsf{S}/\mathsf{NP})/\mathsf{NP})/\mathsf{N})$ that is learnt for the word 'red' in the phrase 'red ball'. This complex syntactic analysis would require a similarly complex logical-form in any model that enforces the principle of categorial type transparency.

Villavicencio (2002) adapts Waldron's system and addresses the problem of categorial over-generation in two ways. First, she uses the principle of categorial type transparency to restrict the set of possible syntactic category assignments for each (word, meaning) pair. Second, she defines a categorial 'Universal Grammar' in terms of a hierarchical system of categorial types and uses this to constrain the order in which categories may be hypothesised. A fragment of this hierarchy is illustrated in Figure 6.7.

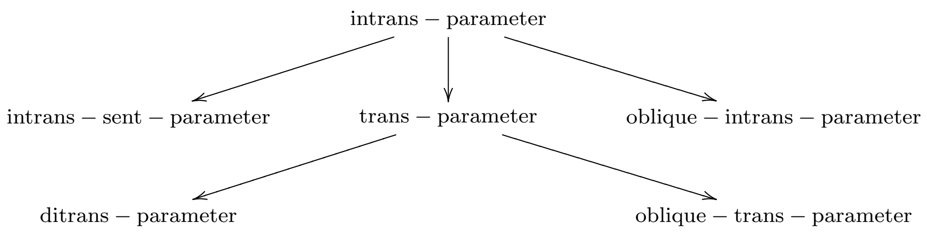

intrans − parameter
intrans − sent − parameter
trans − parameter
oblique − intrans − parameter
ditrans − parameter
oblique − trans − parameter

Figure 6.7: Villavicencio's Categorical Type Hierarchy

The path through the categorial type hierarchy constrains the order in which new category types are added to the grammar (only once intransitive verbs have been added to the grammar can transitive verbs be hypothesised). Each category type has associated category sign which defines the slash directions in syntactic realisation of that category type. These directional signs govern word order and are, where possible, inherited from categories higher in the type hierarchy unless contradictory evidence is seen in the training data. Villavicencio's categorial grammar is learnt from "triggering sentences" in which a semantic representation has been learnt for all words and a syntactic parse can be proposed. In this setting, a triggering sentence is "a sentence paired with a logical form, with corresponding valid semantics and syntactic assignments, which provide evidence for one or more parameters". Triggers are identified with a *trigger identification module*. Grammatical parameters are binary switches governing the set of categorial types allowed in the grammar and the word order associated with each of those types. While Villavicencio's grammatical parameters are binary they are learnt by updating a *parameter confidence weight* on the basis of matching triggering sentences. The parameter is then deemed to be on if this weight exceeds a predetermined threshold and off otherwise.

Buttery (2006) identifies the existence of a trigger identification mechanism that is distinct to the parsing mechanism used to parse and produce sentences as a weakness of Villavicencio's approach. By unifying the mechanisms of parsing and parse proposal during learning, Buttery removes the need for a specialised system of trigger identification along with the predefined set of syntactic parameters that it uses. Buttery retains the categorial type hierarchy of Villavicencio for its inheritance properties but makes no restrictions on the categorial types allowed at any point. Despite removing the distinction between parse proposal for grammar acquisition and parsing for sentence interpretation, Buttery (2006) retains the requirement of Villavicencio (2002) that the correct semantic analysis of every word in a sentence must be known before any syntactic analyses may be attempted. Buttery 2006, pp. 102 points out that this dependence on knowledge of a word's semantics may be excessive and that the use of the pipelined model of learning in which word meanings are learnt separately from syntactic categories slows the pace of grammar acquisition.

Villavicencio (2002) trained her model on a 1,517 utterance subset of the 12,105 child-directed utterances in the Sach's corpus, collected by Sachs (1983), which she has annotated with minimal recursion semantics (Copestake et al., 1998). The model was trained by first using Waldron's cross-situational word learner to learn semantics for all words in 63.6% of these utterances. Then these utterances for which all word meanings are known were sent as input to the syntax learner. Waldron's original system only achieves the correct syntactic analysis of 4.7% of the sentences. Villavicencio performs a more targeted evaluation, looking at the setting of the 18 word-order parameters embedded in her inheritance hierarchy. Of these 8 are correctly set which gives, through the default inheritance of word-order parameters, an average of 13.5 correct parameters. This more targeted evaluation is one that cannot be simply ported to syntax learners that do not use the same parameterisation of the grammar. This is why I cannot do a side by side comparison of the grammar learnt by LanAcq with that of Villavicencio (2002). It also prevented Buttery (2006) from comparing her learner to Villavicencio's.

Buttery (2006) evaluates her learner in an ideal learning setting and shows that it will generally outperform both the TLA and the STL. Buttery also examined the performance of her CGL when learning word-meanings and syntactic parameters in the face of indeterminate meaning representations (akin to the propositional uncertainty represented in the context set of potential sentence meanings in my experiments). In this scenario Buttery's system learns word meanings with high precision but reduced recall. The learning of syntactic parameters in Buttery's system was also shown to be robust to ambiguous training data due to the statistical handling of errors in the learning algorithm that accumulates evidence over the course of the training set. This is an advance over the approaches of Gibson and Wexler (1994); Sakas and Fodor (2001); Villavicencio (2002) which all get stuck in local maxima. Robustness to noise in the training input is also a great advantage of the probabilistic approach that I have presented in this chapter.

All the previous approaches presented in this subsection adopt a principles-and-parameters based view of generative grammar in which the target is a single deterministic parametrisation of the rules allowed to parse the target language. Under such a grammar a parse is either in the language or out of it. This contrasts with the approach presented in this chapter for which the target is a probabilistic description of the lexicon and parsing rules available to the grammar. This probabilistic approach does not give a binary indication of grammaticality for any parse (provided that parse is covered by the 'universal grammar' introduced in Section 6.1) but rather gives a relative (probabilistic) indication of a parse's 'goodness'. Many of the search problems encountered by approaches of Gibson and Wexler (1994); Sakas and Fodor (2001); Yang (2002); Villavicencio (2002) stem from the assumption that the grammar is described by parameters that set hard constraints. This leads to problems when a parameter is incorrectly set (such as in Gibson and Wexler (1994); Sakas and Fodor (2001)) or requires a slow, constrained search strategy to be adopted in order to avoid such incorrect assignments (such as the one used in Villavicencio (2002)).

6.10.3 Learning Word Meanings

Computational models of word learning aim to model the acquisition of a mapping from word to meaning. As with the model presented here, these models are trained on data that pairs child-directed utterances with some semantic representation that approximates the child's inference about the possible meaning of the utterance.

In order to learn the mapping from word to meaning, this approximation need not be compositional. The work of Yu and Ballard (2007); Frank et al. (2008); Fazly et al. (2010) pairs each input utterance with a set of semantic symbols that could represent the meaning of all or some of the words. In Yu and Ballard (2007) and Frank et al.

(2008) the child-directed utterances and set of semantic symbols are extracted from videos of child-caregiver interactions. In Fazly et al. (2010) the training data is generated by pairing child-directed utterances from the CHILDES database with a set of semantic symbols drawn from an input-generation lexicon containing a single meaning per word (ignoring homonymous words). Fazly et al. (2010) also mimic the child's uncertainty about the meaning of the sentence by adding incorrect distractors into the set of semantic symbols used to model meaning.

One drawback of using sets of atomic symbols to approximate meaning is that it requires one to assume that the correct segmentation of the meaning is known. This assumption may be reasonable when learning only object names as in Yu and Ballard (2007) and Frank et al. (2008). In the broader context of word learning, it is very unlikely that the child has access to this segmentation, especially as it differs between languages. Siskind (1996) avoids making this overly strong assumption and was the first person to demonstrate the feasibility of cross-situational word learning from unsegmented compositional meaning representations, albeit from synthetic data rather than real child-directed utterances.

6.10.4 Learning Exemplars

A final related strand of work on modelling language acquisition has learnt descriptions of syntax from raw word strings (Bod, 2009). Related work by O'Donnell et al. (2009); O'Donnell et al. (2011) has investigated the tradeoff between memoization and description via a hierarchical grammar. All of these approaches are initialised with a set of possible rules that allow many possible analyses (similar to the combinatorially defined "universal grammar" used by LanAcq). They then learn the most efficient representation of the data, aiming in doing so to mimic the learning task facing a child. Bod (2009) evaluates his syntax learner on the Eve corpus, reporting syntactic parse scores. Bod also mentions that the search space available to his syntactic learner could be further restrained by the semantics of the training sentences. However, the dependency grammar that his learner acquires does not directly define the manner in which semantics are composed, unlike CCG.

Chapter 7

Conclusions

This thesis has explored the task of learning probabilistic grammars that are capable of parsing sentences onto logical representations of their meanings. The motivations for doing this were two-fold: to induce semantic parsers that can return logical interpretations of sentence meaning for use in downtream computational tasks; and to model the acquisition of such grammars by children. The key hypothesis put forward by this thesis was that the correct probabilistic grammar could be learnt from a training corpus of sentences paired with logical forms via a language independent combinatorial function $\mathcal{T}$ that defines the set of parses consistent with each pair and a language independent learning algorithm.

Any reasonable model of language acquisition must not make assumptions specific to the language being modelled. Therefore the mapping from input pairs to sets of possible parses $\mathcal{T}$ must be language independent for modelling language acquisition. A language independent method of inducing probabilistic grammars for semantic parsing is also desirable. As well as being language independent, the generality of the combinators used in $\mathcal{T}$ allows the approaches to support multiple compositional logical representations of meaning. This flexibility with regards to meaning representation is desirable from the perspective of modelling language acquisition—because we do not know how to accurately represent the contextually afforded meanings available to the child. The ability to support multiple meaning representations is also desirable in a system designed to induce grammars for semantic parsers—because the target meaning representation is often ad-hoc and task specific.

Chapter 3 defined the language independent combinatorial function $\mathcal{T}$ in general terms, so that it would generate all possible parses of any given (sentence, logical-form) pair.

Chapter 4 then described an efficient method of semantic parser induction that incrementally expands a CCG lexicon by applying a restricted form of $\mathcal{T}$ to (sentence, logical-form) pairs, and simultaneously learns a probabilistic parsing model. By learning the parsing model and CCG lexicon in tandem, the learning algorithm in Chapter 4 is able to acquire an accurate parsing model while only ever considering a small subset of the lexical items allowed. The learning algorithm was used to induce semantic parsers for multiple languages and meaning representation formalisms and these semantic parsers were shown to achieve state of the art performance in all evaluations. I have argued that the development of the learning algorithm and methodology presented in Chapter 4 could be seen as a step towards the creation of a black-box system for semantic parser induction that could easily be trained on new and diverse datasets. The work presented in Chapter 4 was published as Kwiatkowski et al. (2010).

Chapter 5 addressed a drawback of using traditionally defined CCG lexicons for the task of semantic parsing. Namely, that because each lexical item contains so much structure describing its syntactic and semantic usage, a single word may be associated with many separate lexical items, each of which suits a different context. This is a problem for domains in which the training data is sparse and does not cover all usages of all words. Chapter 5 introduced a new and more efficient representation of the lexicon that separately stores information about the semantics of words and the syntactic usage of word-types. This *factored* representation was shown to increase recall significantly in a natural language dialogue domain, in which the sentences exhibited a high degree of syntactic variation. With lexical generalisation comes a greater lexical search space. I analysed the trade off between the benefits and drawbacks of the more powerful factorised lexical representation. I also argued that, with a different training scenario, the factored representation of the lexicon could have great potential in domain adaptation for semantic parsers as it stores language-specific and domain-specific knowledge separately.

Chapter 6 addressed a different problem from the previous two chapters: that of modelling a child's acquisition of word-meanings and syntax. For this task training data was created that paired child-directed utterances with a set of logical-forms approximating the *context* within which the utterances were made. The probabilistic model of the grammar and training algorithm presented in Chapter 6 also differed from those used for semantic parsing. The generative model of the grammar covers all sentences, logical-forms and derivations ever possible by use of an infinite Dirichlet process prior. This probabilistic model, and the online training algorithm used to train it, were motivated by the need for psycholinguistic plausibility in a computational model of language acquisition. I showed that the online training algorithm used in Chapter 6 outperformed the state of the art semantic parser induction algorithm described in Chapter 4 when given similar initialisations. This is because it searches for lexical items in the complete space of possible parses whereas the semantic parser induction algorithm performs a greedy search, only ever examining a subset of the possible lexical items supported by the training data. I also showed that the model of language acquisition is capable of learning both word-meanings and word-order rules in ways that mirror observations of these acquisitive actions in children—on the basis of few training examples and with sharp changes in levels of belief.

The experiments presented in Chapters 4, 5 and 6 all support the key hypothesis stated in the introduction and restated at the beginning of this chapter—that probabilistic parsers can be learnt from corpora of (sentence, logical-form) pairs via the combinatorial function $\mathcal{T}$.

The semantic parser induction algorithms presented in this thesis require training data that pairs sentences with hand-annotated logical-forms. This data is expensive to generate and its limited availability is the main hindrance to the extension of the methods presented here. Future work should address efficient methods of grammar expansion that make use of more widely available forms of supervision in addition to the high precision but rare logically annotated data used here.

The model of child language acquisition has addressed several deficiencies in previous computational accounts of word and syntax learning. Its contribution is to show that there is enough information in the structure of the semantics and combinators of CCG to support fully general language-independent learning without invoking any notion of trigger. It is, however, still open to criticism from the perspective of its computational complexity as all different analyses of each training pair are entertained even when a lot of linguistic regularities have been learnt. Future work should investigate the application of more efficient and realistic learning algorithms that do not require enumeration of all possible parses. Additionally, future work should address the problem of more accurately representing the context available to a child with a representation that could serve as (or be used to generate) input for a linguistic learner of the type presented here. This has been partially achieved by Frank et al. (2009) who trained a word learner on annotated videos but who did this only for a small set of objects that needed to be named. As more, richer, data and annotation tools of the type collected and developed by Roy et al. (2006), Kubat et al. (2007) and Roy and Roy (2009) become available, the creation of a realistic corpus for the full linguistic acquisition task may become possible.

Appendix A

Appendices

A.1 The Eve Corpus

The Eve corpus collected by Brown (1973) contains $14,124$ English utterances spoken to a single child, Eve, between the ages of 18 months and 27 months. The utterances in this corpus have been hand annotated by Sagae et al. (2004) with labelled syntactic dependency graphs and morphosyntactic part-of-speech tags. An example dependency graph is shown in Figure A.1.

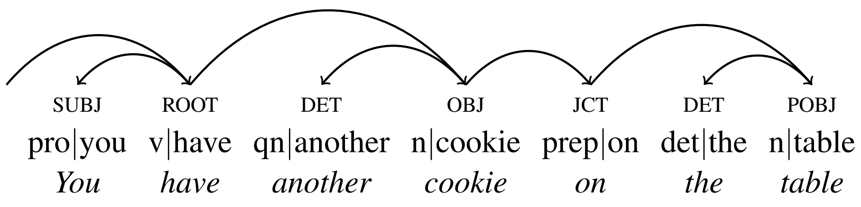


Figure A.1: Syntactic dependency graph from Eve corpus.

These parses represent syntactic information. However, the directed dependencies in the parse can also be viewed as a weak proxy for the predicate-argument structure of a semantic representation. By treating the morphosyntactic part-of-speech tags as nodes in a dependency tree and discarding the linear order of these nodes it is possible to generate a representation of the sentence's predicate argument structure without explicitly encoding word order information.

I generate logical forms of the type discussed in Section 2.1 from these unordered predicate-argument trees by means of a deterministic mapping. This mapping makes use of a closed dictionary of typed semantic constants, predicates, functions and logical connectives.

The mapping from dependency tree to logical-form first retrieves the set of logical constants, predicates, functions and boolean operators consistent with the morphosyntactic tags in the unordered dependency tree. For example, the tags

$$\{\mathrm{pro|you,\ v|have,\ qn|another,\ n|cookie,\ prep|on,\ det|the,\ n|table}\}$$

from the tree in Figure A.1 return the following set of typed logical components.

$$\{you,\ have(x,y,e),\ another(x,p(x)),\ cookie(x),\ on(x,e),\ the(x,p(x)),\ table(x)\}$$

These logical components are then composed into a sentential logical-forms according to a well defined set of logical-form templates. Logical-form templates are hand coded representations of typed logical structure that contain placeholders for predicates, constants, functions and quantifiers. Each logical-form template is licensed by a related commonly occuring substructure in the syntactic dependency tree.

The remainder of this section describes the set of logical-form templates along with the syntactic dependency graphs that license them. In combination these templates provide a mapping from syntactic dependency graph to well typed logical-form that generates logical-forms for 41% of the child-directed utterances in the Eve corpus, yielding a corpus of $5,832$ (utterance, logical-form) pairs. This corpus shall be released shortly.

Quantifiers Nouns and Noun Modifiers are treated in the annotations of Sagae et al. (2004) as trees rooted at the noun node (Figure A.2). Collapsing this tree structure into the predicate-argument structure

$$n|ball(det|a,\ adj|nice,\ adj|big)$$

gives an incorrect scoping of the determiner and requires a predicate of arity 3 to represent the semantics of *ball*. Instead, all nouns and adjectives are treated as arity one predicates with type $\langle e,t\rangle$. These predicates are connected in a conjunction when the dependency tree structure signals that they refer to the same entity. Determiners, quantifiers and possessives introduce variables of type e and outscope predicates with which they occur. The syntactic dependency graph in Figure A.2 is mapped onto the logical structure

$$a(x,\ nice(x) \wedge big(x) \wedge ball(x))$$

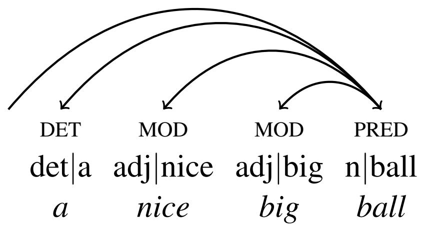


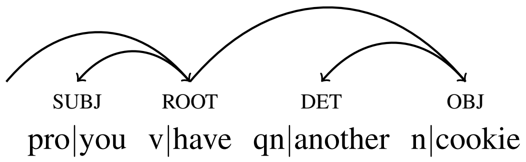
Figure A.2: Noun, determiner and adjective tree

where the determiner a introduces and binds the variable x.

Verbs in the Sagae et al. (2004) annotations may take any subset of subject, object, indirect object, preposition, complementizer as arguments. The number of arguments assigned to each verb in the Eve corpus varies between instances of that verb according to the context in which it is seen. In the mapping to logical form, each verb is classified into one or more of intransitive, transitive and ditransitive. After normalising out differences in grammatical tense and aspect, there are 109 intrasitive, 189 transitive and 18 ditransitive verbs in the corpus.

Each verb also takes, as an argument, an event variable bound by a λ-term at the head of the verb phrase (see Section 2.1.2. This allows the attachment of prepositional phrases and adverbs without the creation of new verbal predicates.

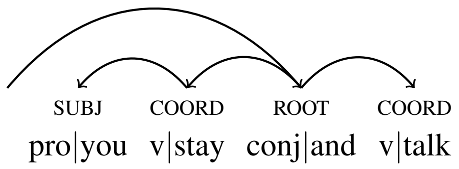
Verb phrases are built by selecting the suitable predicate from the verb dictionary and then adding logical arguments for the subject, object and indirect-object represented in the partial parse tree. For example, the partial tree



is used to build the logical-form $\lambda e.have(you, another(x, cookie(x)), e)$ by first selecting the transitive verb predicate for $have$ from the verb dictionary and then adding the logical-forms that have already been built for the verb's syntactic dependents in the parse tree.

Often, one or more of a verbal predicates arguments does not occur in the sentence's surface form. In particular, 11.6% of the child-directed utterances in the Eve corpus have null-subjects. This is due, in a large part, to the frequency of imperative sentences such as 'drink the water' or 'go find your hat'. In these cases I reject the sentence rather than attempting to guess the correct argument or using a (semantically incorrect) empty argument slot.

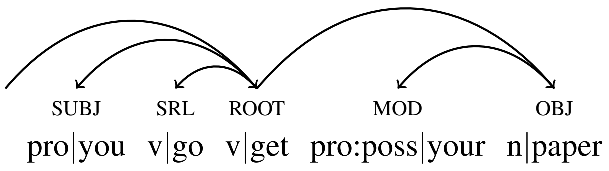
Conjunctions in the Sagae et al. (2004) dependency trees cannot encode the semantic dependencies shared by the conjuncts. In order to model the fact that you is the subject of both $stay$ and $talk$ in the phrase below, we require a DAG structured representation. The deterministic transformation recognises structures like this and shares



subjects and objects across conjuncts when one (or more) of them is missing in one of the conjuncts. For the tree above, the procedure creates the logical-form

$$\lambda ev.stay(you, ev) \wedge talk(you, ev)$$

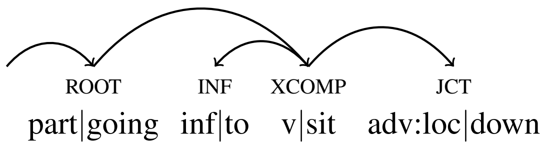
Serial verbs and infinitives are modelled as verb arguments in the Sagae et al. (2004) annotations. For example, the serial verb *go* is hanging off the verb *get* in the tree:



The deterministic mapping recognises these constructions and maps them onto logical forms that put the serial verb in a conjunction with the main verb, sharing its subject and event. E.g.

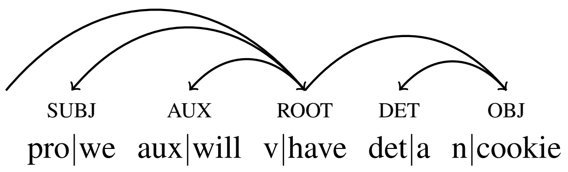
$$\lambda ev.go(you, ev) \wedge get(you, your(x, paper(x)), ev)$$

Infinitives are treated in a similar way, mapping:



onto $\lambda ev.go(you, ev \wedge sit(you, ev) \wedge down(ev)$, where the subject you has been added automatically as described above.

Auxilliaries are represented in the logical representation as they often add a temporal or intensional aspect to the sentence. In the tree structured syntactic annotations auxilliaries modify the main verb:



In the target representation, they take the main verb as an argument. The tree above is mapped onto the logical-form $\lambda ev.will(have(we, a(x, cookie(x)), ev))$.

Adverbial adjuncts and prepositional phrases are attached to verbs using event variables. Doing this allows the attachment of multiple adjuncts without the creation of new verbal predicates or arbitrary nesting of the adjuncts outside the verbal predicate. An example preposition attachment is illustrated in the logical-form below which is a full translation of the dependency graph in Figure A.1.

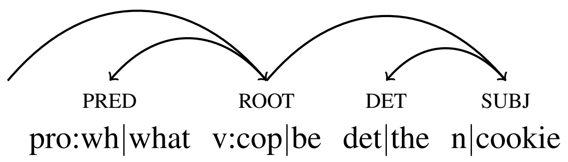
$$\text{You have a cookie on the table}$$
$$\lambda ev.have(you,\ a(y,\ cookie(y)),\ ev) \wedge on(the(z,\ table(z))\ , ev)$$

Copula can either be used to describe equality as in *Bonzo is a dog* or predication as in *Bonzo is brown*. This duality in meaning is well understood and prompted the following outburst from Bertrand Russell.

"The is of "Socrates is human" expresses the relation of subject and predicate; the is of "Socrates is a man" expresses identity. It is a disgrace to the human race that it has chosen to employ the same word "is" for these two entirely different ideas—a disgrace which a symbolic logical language of course remedies.” (Russell, 1920, Chapter XVI)

In the symbolic representation of meaning generated for the sentences in the Eve corpus, the two different uses of the copula are treated differently. Equality is handled using a predicate $eq(x, y)$ giving the logical-form $eq(bonzo, a(\lambda x.dog(x)))$ for the sentence *Bonzo is a dog*. Predication is modelled by applying the predicate directly to the subject, giving the logical form $brown(bonzo)$ for *Bonzo is brown*.

Wh Questions are generated by replacing the morphosyntactic tag related to the wh-word with a variable of type e bound at the root of the sentential logical expression. For example, the dependency tree:



is translated into the logical expression $\lambda x.eq(x,\ the(y, cookie(y)))$. ‘Where?’ questions are modelled with a predicate $eqLoc$, giving the training pair:

$$\text{where is the bicycle}$$
$$\lambda x.eqLoc(the(y, bicycle(y)), x)$$

‘How?’ and ‘why’ questions occur relatively frequently in the Eve corpus but are harder to represent logically. I do not attempt to model these.

Yes/No Questions and Negation are marked using a question operator $Q(x)$ and a negation operator $not(x)$ respectively. Negation is modelled explicitly in the dependency tree. Yes/No questions are recognised as non-wh utterances ending in a question mark.

Vocatives and Communicators occur a lot in the child directed utterances of the Eve corpus. These do not have any obvious meaning in the theory of semantics set forward in Section 2.1. Theay are therefore not represented in the logical expressions generated for the Eve corpus.

A.1.1 Untranslated Utterances

The deterministic transformation described above sucessfully generates logical expressions for $5,832$ child directed utterances - 41% of those in the Eve corpus. This section analyses the distribution of sentences for which the transformation was possible and classifies, where possible, the 59% of cases for which it failed.

Of the $8,292$ of child-directed utterances that could not be translated, 2314 were one-word utterances that did not have any clear semantic interpretation within our model of meaning. These account for 16% of the total utterance set and, of this 16%, 79% of the utterances rejected are accounted for by 10 distinct utterance types: 'what'; 'okay'; 'alright'; 'no'; 'yeah'; 'hmm'; 'yes'; 'uhhuh'; 'mhm'; and 'thankyou'. Sentences with null subjects that could not be resolved by adding an implicit you subject make up 5.2% of the child-directed data. Other null verbal arguments make up a further 3.6%. When a verbal predicate in a logical expression is missing a required subject or object, the logical expression - and therefore training pair - is rejected.

941 utterances - or 6.7% of all utterances - contain noun constructions that we cannot easily translate into a logical expression. These constructions are often well formed and could be logically represented with due work. However, each distinct type of construction occurs in only a few cases and there is no easily recognizable common structure on which to hook a deterministic mapping to logic. Examples of noun constructions that we fail to translate are 'box of books' and 'top to a jar'.

Finally, 3.2% of utterances contain Wh words that we cannot represent. These include all 'why', 'how' and 'when' sentences. Another 1.5% of utterances are rejected because they contain adverbs modifying adjectives such as 'much better'. Given our representation of adverbs (predicates with a single Davidsonian event argument), these cannot be represented.

This leaves at least 22% of the data unaccounted for. This is approximate because more than one of the problems listed above may occur in a single utterance. No other problems may occur with the unrepresentable one-word utterances. The uncategorisable 22% contains hard constructions not allowed for by either the translation script or analysis. This is not suprising as any easily categorisable construction should, for that reason, be easy to translate into a logical expression. I believe that the 41% of the data successfully transformed is representative of the full dataset and is therefore a

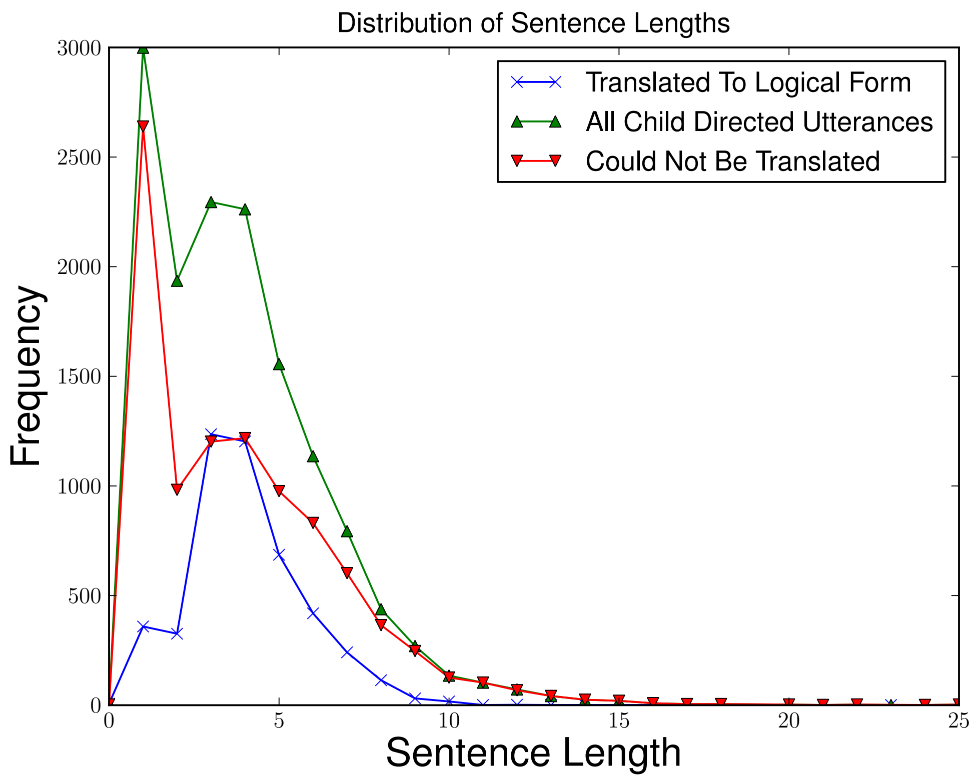


Figure A.3: Word length distribution in original utterance-set, translated utterance-set and all utterance-set.

reasonable approximation to the input available to a child.

A.1.2 Distribution of Translated Sentences

Figure A.3 shows the frequency of utterances of different length in the original, translated and rejected corpora. The translation to logical-form rejects the majority of one-word utterances and also cannot translate many of the utterances over 10 words long. These longer utterances would, anyway, be rejected by the training algorithm presented in Chapter 6. Overall, the vast majority of utterances lie in the range between the lengths of 2 and 8 words. In this range, the distribution of utterance lengths in the translated set takes a similar shape to the distribution of utterance lengths in the original Eve corpus.

A.2 GeoQuery Type Hierarchy

UBL and FUBL make use of a type hierarchy to govern allowed logical structures within the domain of interest. Each of the types added to entity e, integer i and truth-value t following types are sub-types of the entity type e. Each predicate has a type limit on its allowed arguments. As an example the type hierarchy and allowed predicate types for the GeoQuery domain are given here:

Type	Type parent	Entity class
n	e	name
lo	e	location
s	lo	state
c	lo	city
r	lo	river
l	lo	lake
m	lo	mountain
co	lo	country
p	lo	place
to	lo	town

Predicate	Type
state:t	$\langle s, t\rangle$
city:t	$\langle c, t\rangle$
town:t	$\langle to, t\rangle$
place:t	$\langle p, t\rangle$
lake:t	$\langle l, t\rangle$
mountain:t	$\langle m, t\rangle$
capital:t	$\langle c, t\rangle$
capital:c	$\langle s, c\rangle$
river:t	$\langle r, t\rangle$
major:t	$\langle lo, t\rangle$
loc:t	$\langle r, \langle c, t\rangle$
loc:t	$\langle r, \langle s, t\rangle$
loc:t	$\langle l, \langle s, t\rangle$
loc:t	$\langle p, \langle s, t\rangle$
loc:t	$\langle c, \langle s, t\rangle$
loc:t	$\langle m, \langle s, t\rangle$
loc:t	$\langle to, \langle s, t\rangle$
loc:t	$\langle lo, \langle co, t\rangle$
loc:t	$\langle p, \langle c, t\rangle$
loc:t	$\langle r, \langle p, t\rangle$

Bibliography

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.

Alishahi, A., Fazly, A., and Stevenson, S. (2008). Fast mapping in word learning: what probabilities tell us. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, CoNLL '08, pages 57–64, Stroudsburg, PA, USA. Association for Computational Linguistics.

Artzi, Y. and Zettlemoyer, L. (2011). Bootstrapping semantic parsers from conversations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 421–432, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Baker, M. (2005). Mapping the terrain of language acquisition. *Language Learning and Development*, 1:93–129.

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

Beal, M. J. (2003). Variational algorithms for approximate Bayesian inference. Technical report, Gatsby Institute, UCL.

Blackburn, P. and Bos, J. (2003). Computational semantics. *Theoria*, 18(1):27–45.

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

Bod, R. (2009). From exemplar to grammar: Integrating analogy and probability in language learning. *Cognitive Science*, 33(4).

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

Bottou, L. (1991). Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France. EC2.

Branavan, S., Zettlemoyer, L., and Barzilay, R. (2010). Reading between the lines: Learning to map high-level instructions to commands. In *Association for Computational Linguistics (ACL)*.

Brown, R. (1973). *A First Language: the Early Stages*. Harvard University Press, Cambridge MA.

Buttery, P. J. (2006). Computational models for first language acquisition. Technical Report UCAM-CL-TR-675, University of Cambridge, Computer Laboratory.

Carey, S. and Bartlett, E. (1978). Acquring a single new word. *Papers and Reports on Child Language Development*, 15(1-2):17–29.

Chen, D. L., Kim, J., and Mooney, R. J. (2010). Training a multilingual sportscaster: Using perceptual context to learn language. *Journal of Artificial Intelligence Research*, 37:397–435.

Chen, D. L. and Mooney, R. J. (2008). Learning to sportscast: A test of grounded language acquisition. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, Helsinki, Finland.

Chen, D. L. and Mooney, R. J. (2011a). Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-2011)*, pages 859–865.

Chen, D. L. and Mooney, R. J. (2011b). Panning for gold: Finding relevant semantic content for grounded language learning. In *Proceedings of Symposium on Machine Learning in Speech and Language Processing (MLSLP 2011)*.

Chomsky, N. (1981). *Lectures on Government and Binding*. Foris, Dordrecht.

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363.

Church, A. (1940). A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68.

Clark, S. and Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 103, Morristown, NJ, USA. Association for Computational Linguistics.

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

Clarke, J., Goldwasser, D., Chang, M.-W., and Roth, D. (2010). Driving semantic parsing from the world's response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning (CoNLL-2010)*, pages 18–27, Uppsala, Sweden.

Collins, M. (1997). Three generative lexicalized models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, Madrid*, pages 16–23, San Francisco, CA. Morgan Kaufmann.

Cooper, D. E. (1975). *Philosophy and the Nature of Language*. Longman.

Copestake, A., Flickinger, D., and Sag, I. (1998). Minimal recursion semantics: An introduction. In *ESSLLI*.

Crain, S. and Thornton, R. (1998). *Investigations in Universal Grammar*. MIT Press, Cambridge MA.

Creutz, M. and Lagus, K. (2007). Unsupervised models for morpheme segmentation and morphology learning. *ACM Trans. Speech Lang. Process.*, 4:3:1–3:34.

Dalrymple, M., Shieber, S., and Pereira, F. (1991). Ellipsis and higher-order unification. *Linguistics and Philosophy*, 14:399–452.

Davidson, D. (1967). The logical form of action sentences. In Rescher, N., editor, *The Logic of Decision and Action*. University of Pittsburgh Press, Pittsburgh.

Dowty, D. R., Wall, R. E., and Peters, S. (1980). *Introduction to Montague Semantics*, volume 11. Kluwer Academic Press.

Fazly, A., Alishahi, A., and Stevenson, S. (2008). A probabilistic incremental model of word learning in the presence of referential uncertainty. In *Proceedings of the 30th Annual Conference of the Cognitive Science Society*.

Fazly, A., Alishahi, A., and Stevenson, S. (2010). A probabilistic computational model of cross-situational word learning. *Cognitive Science*, 34(6):1017–1063.

Ferguson, T. S. (1973). A bayesian analysis of some nonparametric problems. *The Annals of Statistics*, 1(2):pp. 209–230.

Fodor, J. (1998). Unambiguous triggers. *Linguistic Inquiry*, 29(1):1–36.

Frank, M., Goodman, N., and Tenenbaum., J. (2009). Using speakers' referential intentions to model early cross-situational word learning. *Psychological Science*, 20(5):578–585.

Frank, M. C., Goodman, N. D., and Tenenbaum, J. B. (2008). A bayesian framework for cross-situational word-learning. In *Advances in Neural Information Processing Systems*.

Frege, G. (1892). Uber sinn und bedeutung. *Zeitschrift fur Philosophie und philosophische Kritik*.

Ge, R. and Mooney, R. J. (2005). A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 9–16, Stroudsburg, PA, USA.

Ghahramani, Z. and Attias, H. (2000). Online variational bayesian learning, 2000. Slides from talk presented at NIPS 2000 workshop on Online Learning, available at mlg.eng.cam.ac.uk/zoubin/papers/nips00w.ps.

Gibson, E. and Wexler, K. (1994). Triggers. *Linguistic Inquiry*, 25:355–407.

Goldwasser, D., Reichart, R., Clarke, J., and Roth, D. (2011). Confidence driven unsupervised semantic parsing. In *Association for Computational Linguistics (ACL)*.

Goldwater, S., Griffiths, T. L., and Johnson, M. (2009). A Bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112(1):21–54.

Griffiths, T. L. and Tenenbaum, J. B. (2005). Structure and strength in causal induction. *Cognitive Psychology*, 51:354–384.

Griffiths, T. L. and Tenenbaum, J. B. (2006). Optimal predictions in everyday cognition. *Psychological Science*, 17:767–773.

Hayes, B., Curtiss, S., Szabolcsi, A., Stowell, T., Stabler, E., Sportiche, D., Koopman, H., Keating, P., Munro, P., Hyams, N., and Steriade, D. (2001). *Linguistics: An introduction to linguistic theory*. Wiley-Blackwell.

He, Y. and Young, S. (2006). Spoken language understanding using the hidden vector state model. *Speech Communication*, 48(3-4).

Hockenmaier, J. and Steedman, M. (2002). Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Meeting of the ACL*, pages 335–342, Philadelphia, PA.

Hoffman, M., Blei, D. M., and Bach, F. (2010). Online learning for latent dirichlet allocation. In *Neural Information Processing Systems*.

Honkela, A. and Valpola, H. (2003). On-line variational bayesian learning. In *In Proc. of the 4th Int. Symp. on Independent Component Analysis and Blind Signal Separation*, pages 803–808.

Hyams, N. (1986). *Language Acquisition and the Theory of Parameters*. Reidel, Dordrecht.

Kate, R. J. and Mooney, R. J. (2006). Using string-kernels for learning semantic parsers. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics*.

Kate, R. J., Wong, Y. W., and Mooney, R. J. (2005). Learning to transform natural to formal languages. In *Proceedings of the National Conference on Artificial Intelligence*.

Klein, D. and Manning, C. D. (2003). A* parsing: Fast exact viterbi parse selection. In *Proceedings of the Human Language Technology Conference of the NAACL*.

Knight, K. (1989). Unification: A multidisciplinary survey. *ACM Computing Surveys*, 21:93–124.

Kubat, R., DeCamp, P., Roy, B., and Roy, D. (2007). Totalrecall: Visualization and semi-automatic annotation of very large audio-visual corpora. In *Proceedings of International Conference on Multimodal Interfaces*.

Kurihara, K. and Sato, T. (2006). Variational bayesian grammar induction for natural language. In *International Colloquium on Grammatical Inference*, pages 84–96.

Kushner, H. J. and Yin, G. (1997). *Stochastic approximation algorithms and applications / Harold J. Kushner, G. George Yin.* Applications of mathematics: 35. New York : Springer, c1997.

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2010). Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the Conference on Emperical Methods in Natural Language Processing*.

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2011). Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the Conference on Emperical Methods in Natural Language Processing*.

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2012). A probabilistic model of language acquisition from utterances and meanings. Submitted for publication.

Levy, R., Reali, F., and Griffiths, T. (2009). Modeling the effects of memory on human online sentence processing with particle filters. In *Advances in Neural Information Processing Systems 21*.

Liang, P., Jordan, M. I., and Klein, D. (2009). Learning semantic correspondences with less supervision. In *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*.

Liang, P., Jordan, M. I., and Klein, D. (2011). Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*.

Liang, P., Petrov, S., Jordan, M. I., and Klein, D. (2007). The infinite PCFG using hierarchical dirichlet processes. In *Empirical Methods in Natural Language Processing*, pages 688–697.

Lu, W. and Ng, H. T. (2011). A probabilistic forest-to-string model for language generation from typed lambda calculus expressions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1611–1622, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Lu, W., Ng, H. T., Lee, W. S., and Zettlemoyer, L. S. (2008). A generative model for parsing natural language to meaning representations. In *Proceedings of The Conference on Empirical Methods in Natural Language Processing*.

Marcus, G. F., Pinker, S., Ullman, M., and Hollander, M. (1992). Overregularization in language acquisition. *Monographs of the Society for Research in Child Development*, 57.

Mattys, S. and Jusczyk, P. (2001). Phonotactic cues for segmentation of fluent speech by infants. *Cognition*, 78:91–121.

Mattys, S., Jusczyk, P., Luce, P., and Morgan, J. (1999). Phonotactic and prosodic effects on word segmentation in infants. *Cognitive Psychology*, 38:465–494.

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

Montague, R. (1973). The Proper Treatment of Quantification in Ordinary English. *Approaches to Natural Language: proceedings of the 1970 Stanford workshop on Grammar and Semantics*, pages 221–242.

Neal, R. and Hinton, G. E. (1998). A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

O'Donnell, T., Goodman, N. D., and Tenenbaum, J. B. (2009). Fragment grammars: Exploring computation and reuse in language. Technical report, Massachusetts Institute of Technology.

O'Donnell, T. J., Snedeker, J., Tenenbaum, J. B., and Goodman, N. D. (2011). Productivity and reuse in language. In *Proceedings of the Cognitive Science Society*.

Pauls, A. and Klein, D. (2010). Hierarchical A* parsing with bridge outside scores. In *ACL (Short Papers)*, pages 348–352.

Pearl, L., Goldwater, S., and Steyvers, M. (2010). How ideal are we? incorporating human limitations into bayesian models of word segmentation. *bucld10*, pages 315–326.

Pinker, S. (1979). Formal models of language learning. *Cognition*, 7:217–283.

Poon, H. and Domingos, P. (2009). Unsupervised semantic parsing. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Poon, H. and Domingos, P. (2010). Unsupervised ontology induction from text. In *Association for Computational Linguistics (ACL)*.

Portner, P. H. and Partee, B. H., editors (2002). *Formal Semantics*. Linguistics: Essential Readings. Wiley-Blackwell, Malden.

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

Roy, B. C. and Roy, D. (2009). Fast transcription of unstructured audio recordings. In *Proceedings of Interspeech*.

Roy, D., Patel, R., DeCamp, P., Kubat, R., Fleischman, M., Roy, B., Mavridis, N., Tellex, S., Salata, A., Guiness, J., Levit, M., and Gorniak, P. (2006). The human speechome project. In *Proceedings of Annual Meeting of the Cognitive Science Society*.

Russell, B. (1920). *Introduction to mathematical philosophy*. Muirhead library of philosophy. Allen & Unwin.

Sachs, J. (1983). Talking about the there and then: the emergence of displaced reference in parent-child discourse. *Children's language*, 4:1–28.

Saffran, J., Aslin, R., and Newport, E. (1996). Statistical learning by 8-month-old infants. *Science*, 274:1926–1928.

Sagae, K., MacWhinney, B., and Lavie, A. (2004). Adding syntactic annotations to transcripts of parent-child dialogs. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*. Lisbon, LREC.

Sakas, W. and Fodor, J. D. (2001). The structural triggers learner. In Bertolo, S., editor, *Language Acquisition and Learnability*, pages 172–233. Cambridge University Press, Cambridge.

Sanborn, A. N., Griffiths, T. L., and Navarro, D. J. (2010). Rational approximations to rational models: Alternative algorithms for category learning. *Psychological Review.*, 117(4):1144–1167.

Sato, M. (2001). Online model selection based on the variational bayes. *Neural Computation*, 13(7):1649–1681.

Shi, L., Griffiths, T. L., Feldman, N. H., and Sanborn, A. N. (2010). Exemplar models as a mechanism for performing bayesian inference. *Psychonomic Bulletin & Review*, 17(4):443–464.

Siskind, J. M. (1992). *Naive Physics, Event Perception, Lexical Semantics, and Language Acquisition*. PhD thesis, Massachusetts Institute of Technology.

Siskind, J. M. (1996). A computational study of cross-situational techniques for learning word-to-meaning mappings. *Cognition*, 61(1-2):1–38.

Steedman, M. (2000). *The Syntactic Process*. MIT Press, Cambridge, MA.

Stirling, C. (2010). Introduction to decidability of higher-order matching. In *International Conference on Foundations of Software Science and Computation Structures*.

Tarski, A. (1933). *The concept of truth in the languages of the deductive sciences*. Muirhead library of philosophy. Allen & Unwin. expanded English translation in Tarski 1983, pp. 152–278.

Thompson, C. A. and Mooney, R. J. (1999). Automatic construction of semantic lexicons for learning natural language interfaces. In *AAAI/IAAI*, pages 487–493.

Thornton, R. and Tesan, G. (2007). Categorical acquisition: Parameter setting in universal grammar. *Biolinguistics*, 1.

Titov, I. and Klementiev, A. (2011). A bayesian model for unsupervised semantic parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1445–1455, Portland, Oregon, USA. Association for Computational Linguistics.

Tomasello, M. (1995). Joint attention as social congnition. In Morre, C. and P.Dunham, editors, *Joint attention: Its origins and role in development*, pages 103–130.

Tomasello, M. (1999). *The Cultural Origins of Human Cognition*. Harvard University Press, Cambridge, MA.

Villavicencio, A. (2002). The acquisition of a unification-based generalised categorial grammar. Technical Report UCAM-CL-TR-533, University of Cambridge, Computer Laboratory.

Waldron, B. (1999). Learning grammar from corpora. Master's thesis, University of Cambridge.

Wang, C., Paisley, J., and David (2011). Online Variational Inference for the Hierarchical Dirichlet Process. In *In Proc. of Artificial Intelligence and Statistics*.

Wexler, K. (1998). Very early parameter setting and the unique checking constraint: A new explanation of the optional infinitive stage. *Lingua*, 106:23–79.

Wong, Y. W. and Mooney, R. (2006). Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL*.

Wong, Y. W. and Mooney, R. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Association for Computational Linguistics*.

Xu, F. and Tenenbaum, J. B. (2007). Word learning as Bayesian inference. *Psychological Review*, 114:245–272.

Yang, C. (2002). *Knowledge and Learning in Natural Language*. Oxford University Press, Oxford.

Yu, C. and Ballard, D. H. (2007). A unified model of early word learning: Integrating statistical and social cues. *Neurocomputing*, 70(13-15):2149 – 2165.

Zelle, J. M. and Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*.

Zettlemoyer, L. S. and Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Zettlemoyer, L. S. and Collins, M. (2007). Online learning of relaxed CCG grammars for parsing to logical form. In *Proc. of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Zettlemoyer, L. S. and Collins, M. (2009). Learning context-dependent mappings from sentences to logical form. In *Proceedings of The Joint Conference of the Associa-*

tion for Computational Linguistics and International Joint Conference on Natural Language Processing.

Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 116–, New York, NY, USA. ACM.