Situated Understanding and Learning of Natural Language

Yoav Artzi

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2015

Reading Committee:

Luke S. Zettlemoyer, Chair

Dieter Fox

Patrick Pantel

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Situated Understanding and Learning of Natural Language

Yoav Artzi

Chair of the Supervisory Committee:
Associate Professor Luke S. Zettlemoyer
Computer Science & Engineering

Robust language understanding systems have the potential to transform how we interact with computers. However, significant challenges in automated reasoning and learning remain to be solved before we achieve this goal. To accurately interpret user utterances, for example when instructing a robot, a system must jointly reason about word meaning, grammatical structure, conversation history and world state. Additionally, to learn without prohibitive data annotation costs, systems must automatically make use of weak interaction cues for autonomous language learning.

We present a framework that uses situated interactions to learn to map sentences to rich, logical meaning representations. Our approach induces a Combinatory Categorial Grammar (CCG), while relying on various learning cues, such as easily gathered demonstrations and even raw conversations without any additional annotation effort. It achieves state-of-the-art performance on a number of tasks, including robotic interpretation of navigational directions and learning to understand user utterances in dialog systems. Such an approach, when integrated into complete systems, has the potential to achieve continuous, autonomous learning by participating in interactions with users.

We first describe an approach to induce a CCG from automatically recorded conversations. Next, we show that jointly reasoning about language meaning and system response improves instruction following, and describe an approach to induce compact grammars in such scenarios. Finally, with the goal of studying linguistic phenomena not addressed by existing corpora, we present a broad-coverage CCG parser to recover rich formal representations. Together, our techniques lower or eliminate annotation overhead, improve situated language understanding and enable grammar induction for recovering logical forms on a scale not possible before.

CONTENTS

Contents i

List of Figures v

List of Tables ix

List of Algorithms xi

Chapter 1: Introduction 1

1.1 Challenges 3

1.2 Previous Approaches 5

1.3 Overview of Approach 7

1.4 Contributions 15

1.5 Thesis Outline 16

Chapter 2: Background 17

2.1 Simply-typed Lambda Calculus 17

2.2 Combinatory Categorial Grammar 20

2.3 Factored Lexicons 26

2.4 Weighted CCG 26

2.5 Supervised Learning with GENLEX 27

2.6 Skolem Terms 29

Chapter 3: Related Work 30

3.1 Combinatory Categorial Grammar 30

3.2 Semantics 31

3.3 Learning from Linguistic Annotation for Semantic Parsing 32

3.4 Learning from Demonstrations for Semantic Parsing 36

3.5 Other Forms of Supervision for Semantic Parsing 38

3.6 Non-learning Approaches for Semantic Parsing 39

Chapter 4: Bootstrapping Semantic Parsers from Conversations 40

4.1 Introduction 40

4.2 Problem 42

4.3 Overview of Approach 43

4.4 Measuring Loss 46

4.5 Learning 50

4.6 Data Sets 53

4.7 Experimental Setup 54

4.8 Results 56

Chapter 5: Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions 58

5.1 Introduction 58

5.2 Technical Overview 60

5.3 Spatial Environment Modeling 61
5.4 Modeling Instructional Language 63
5.5 Joint Parsing and Execution 69
5.6 Learning 72
5.7 Experimental Setup 75
5.8 Results 78

Chapter 6: Learning Compact Lexicons for CCG Semantic Parsing 80
6.1 Introduction 80
6.2 Learning 81
6.3 Global Voting for Lexicon Learning 86
6.4 Experimental Setup 88
6.5 Results 89

Chapter 7: Broad-coverage CCG Semantic Parsing with AMR 93
7.1 Introduction 93
7.2 Technical Overview 95
7.3 Mapping Sentences to Logical Form 97
7.4 Learning 101
7.5 Experimental Setup 106
7.6 Results 107

Chapter 8: Conclusion 112
8.1 Future Work 113

Appendix A: Pseudocode for Voting Procedures 115

A.1 Vote Aggregation for Lexemes 115

Appendix B: AMR to Lambda Calculus Conversion 118

Appendix C: AMR Specification Function $\mathcal{S}$ 120

Bibliography 122

LIST OF FIGURES

1.1 Schematic diagram of a map environment and examples of navigation instructions paired with their logical forms and demonstrations of system response, navigation traces in this case. The environment includes an agent (marked in green and facing left), two hallways and two chairs. This type of environment is described in Chapter 5. 2

1.2 A hypothetical conversation between a user and a natural language calendar system. The arrows show information the system was able to recover after failing to fully understand the first user request and how it aligns to this request. This alignment can be used to learn a semantic parser without any annotation effort. . . 7

1.3 Schematic diagrams of two similar environments. Given the instruction above, different logical form representations are required to move to the intersection in front of the agent. In the environment on the left, the prepositional phrase *in a red hall* is coordinated and applied both to the chair and lamp. In contrast, on the right, a different attachment is required, and the phrase is not coordinated, applying only to the lamp. 11

1.4 Example sentence and its AMR, underspecified logical form (ULF) and complete logical form representation (LF). The colored symbol **p** represents a distant reference introduced by *he* and resolved to the entity introduced by *Akkermans*. In ULF, this reference is is unresolved and represented by ID, which is specified to **3** in LF. 14

2.1 Partial typing hierarchy for a travel planning domain. This domain includes three basic types: numeral n, truth-value t and entity e. Edges indicate inheritance. Transportation tr inherits from e. Flight fl inherits from air travel fla, which inherits from tr. Ground transportation gt is a type of transportation tr. Airport ap and city ci are both locations an inherit from loc, which inherits from e. 18

2.2 Example CCG parse tree for the sentence *walk to the red chair in the intersection*. . 22

2.3 Example CCG parse tree for the phrase *square blue or round yellow pillow*. 23

2.4 Lexical entries generated by GENLEX for the sentence x and logical form z given two predefined templates. 28

4.1 Conversational excerpt from a DARPA Communicator travel-planning dialog. Each system statement is labeled with representations of its speech act and logical meaning, in parentheses. User utterances have no labels. Conversations of this type provide the training data to learn semantic parsers for user utterances. 42

4.2 An example CCG parse. This parse shows the construction of a logical form with an array-typed variable $x_{[]}$ that specifies a list of flight legs, indexed by $x[1]$ and $x[2]$. The top-most parse steps introduce lexical items while the lower ones create new nonterminals according the CCG combinators ($>$, $<$, etc.), see Steedman (2000) for details. 45

4.3 Conversation reflecting an interaction as seen in the DARPA Communicator travel-planning dialogs. 47

5.1 A sample navigation instruction set. Each instruction is paired with its lambda-calculus meaning representation. 59

5.2 Schematic diagram of a map environment and examples of formal representation of spatial phrases. 62

5.3 A CCG parse with a prepositional phrase. 67

5.4 A CCG parse showing adverbial phrases and topicalization. 68

6.1 Lexical entries for the word *chair* as learned with no corpus-level statistics. Our approach is able to correctly learn only the top two bolded entries. 81

6.2 Four rounds of CONSENSUSVOTE for the string *chair* for four training samples. For each sample, we specify the set of lexemes generated in the Round 1 column, and update this set after each round. At the end, the highest voted new lexeme according to the final votes is returned. In this example, MAXVOTE and CONSENSUSVOTE lead to different outcomes. MAXVOTE, based on the initial sets only, will select $\langle$*chair*, $\{\text{easel}\}\rangle$. 86

7.1 A sentence (x) paired with its AMR (a), underspecified logical form (u) and fully specified logical form (z) representations. 96

7.2 A complete derivation for the sentence *Obama promised to discuss lifting the embargo*. 98

7.3 A visualization of the factor graph constructed for the derivation in Figure 7.2. Only a subset of the factors are included and the set of possible assignments is only specified for placeholders (for REL only a subset is listed). 98

7.4 Results from parsing the sentence: *Cyber space essentially has no borders.* The system failed to recover the manner of have-03, and it incorrectly applied the negation to border rather than to have-03. The SMATCH F1 for this example is 0.80. . . 109

7.5 Results from parsing the sentence: *Akkermans was working with Storimans and suffered minor injuries*. The system correctly analyzed the coordination, but it incorrectly introduced a suffer-01 predicate and inferred that *Storimans* fills the role of a job rather than a coworker. The SMATCH F1 for this example is 0.70. . . . 110

7.6 Development SMATCH F1 without early updates (●) and with early updates (■). . . . 111

LIST OF TABLES

4.1 Data set statistics for Lucent and BBN systems. 53

4.2 Mean exact-match results for cross fold evaluation on the development sets. 55

4.3 Exact- and partial-match results on the test sets. 56

5.1 Corpora statistics (lower-cased data). 76

5.2 Cross-validation development accuracy and standard deviation on the oracle corpus. 77

5.3 Cross-validation accuracy and standard deviation for the SAIL corpus. 77

5.4 Oracle corpus test accuracy and standard deviation results. 77

6.1 Ablation study using cross-validation on the ORACLE corpus training data. We report mean precision (P), recall (R) and harmonic mean (F1) of execution accuracy on single sentences and sequences of instructions and mean lexicon sizes. Bold numbers represent the best performing method on a given metric. 90

6.2 Our final results compared to previous work on the SAIL and ORACLE corpora. We report mean precision (P), recall (R), harmonic mean (F1) and lexicon size results and standard deviation between runs (in parenthesis) when appropriate. *Our Approach* stands for batch learning with a consensus voting and pruning. Bold numbers represent the best performing method on a given metric. 91

6.3 Example entries from a learned ORACLE corpus lexicon using batch learning. For each phrase we report the number of lexical entries without voting (CONSENSUSVOTE) and pruning and with, and provide a few examples. Struck entries were successfully avoided when using voting and pruning. 92

7.1 Test SMATCH results. 108

7.2 Development SMATCH results. 108

B.1 AMR to lambda calculus conversion cases. 119

C.1 The specification function $\mathcal{S}$ used in our experiments. 121

LIST OF ALGORITHMS

1 Loss-driven learning algorithm. 51

2 Validation-driven situated learning algorithm. 73

3 Batch algorithm for maximizing $\mathscr{L}(\theta, \Lambda, D)$. See Section 6.2.1 for details. 83

4 GENENTRIES: Algorithm to generate lexical entries from one training sample. See Section 6.2.2 for details. 84

5 COMPUTEUPDATE: Algorithm to compute the gradient and the set of lexical entries to prune for one datapoint. See Section 6.2.3 for details. 85

6 The main learning algorithm. 102

7 GENENTRIES: Procedure to generate lexical entries from one training sample. See Section 7.4.1 for details. 103

8 MAXVOTE: Voting strategy to select the highest voted lexical entries. See SectionA.1.1 for details. 116

9 CONSENSUSVOTE: Voting strategy to select lexical entries preferred by most datapoints through multiple rounds of voting. See Section A.1.2 for details. 117

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Luke Zettlemoyer, for his guidance, advice, support and teaching throughout my studies. Luke has been my biggest champion and supporter. I have benefited greatly from his advice on every aspect of my research and academic career. Luke has been my role model throughout my studies and will remain so as I continue with my academic career. I have been extremely fortunate to have him as my mentor and friend.

Thank you also to all my mentors outside of UW: to Patrick Pantel, your friendship, support and advice broadened my perspective; to Michael Gamon, for many wonderful conversations; to Dipanjan Das, for giving me new perspective on my research; and to Slav Petrov, for the opportunity to work along some of the brightest researchers in the field. In addition to Luke and Patrick, I wish to thank my supportive and inspiring committee members: Dieter Fox, for your insightful discussions and unlimited support; Dan Roth, for many thought provoking conversations throughout my studies; and Dan Weld, for his insights and advice during my job search. I have been fortunate and honored to have the support of Raymond Mooney. Ray, the groundbreaking research by you and your students over the last two decades have been fundamental in the formation of my research outlook. Thank you for the endless support.

I wish to extend special thanks to Nicholas FitzGerald, Kenton Lee and Tom Kwiatkowski, with whom I had numerous deep discussions and collaborations that contributed significantly to my research. To Mark Yatskar, your accurate critique and superb technical understanding have had tremendous impact on my work. I wish to thank you for being my close colleague and dear friend throughout my time in Seattle. I have also been lucky to widely collaborate with numerous talented people: Regina Barzilay, Maya Cakmak, Eunsol Choi, Jesse Dodge, Max Forbes and Nate Kushman. I have been privileged to work and learn from all of you. To Mark Steedman, I thank

you for many thoughtful discussions, which have significantly influenced my research. I was also fortunate to have made wonderful friends in the UW community: Robert Gens, Chloé Kiddon, Paris Koutris, Ricardo Martin and Tony Fader. Thank you all for fruitful discussions and support. To all my friends and Smith, thank you for many wonderful, fun and thought provoking evenings.

UW is home to a thriving research community, which I have benefited greatly from. I wish to extend special thanks to: Shiri Azenkot, Emily Bender, Yejin Choi, Janara Christensen, Luheng He, Raphael Hoffmann, Gina-Anne Levow, Victoria Lin, Xiao Ling, Mike Lewis, Cynthia Matuszek, Daniel Perelman, Hoifung Poon, Alan Ritter, Gabriel Schubiner, Noah Smith, Adrienne Wang, Wei Xu, Congle Zhang and Leila Zilles. Interactions and discussions with all of these people have helped to shape my ideas about how to do research. I wish to thank Oren Etzioni for believing in me and convincing me that UW is the best place for me. I benefited greatly from the openness of CSE. I wish to extend special gratitude to Pedro Domingos, Michael Ernst, Hank Levy, Franzi Roesner and Zach Tatlock for their advice. Thank you also to all the people that made my graduate studies such a smooth process: Elise DeGoede, Lindsay Michimoto and Chiemi Yamaoka-Vismale. Seattle is home to one of the most exciting research communities in the world. I wish to especially thank Bill Dolan and the Microsoft Research NLP group for many wonderful interactions.

At the base of my graduate studies is my undergraduate education in the School of Computer Science, Tel Aviv University. I especially wish to thank Daniel Cohen-Or and Ronitt Rubinfeld. Your mentorship and support have been instrumental in preparing me for my graduate degree.

This degree would not have been possible without a strong family tradition that values education and inspirational figures to look up to. I wish to thank my father, Zvi Artzi, who has inspired and encouraged me throughout my life and my studies. His dedication, integrity and diligence have been inspirational in my work and research. Following in his footsteps and upholding myself to his standards has been a motivating force throughout my studies and will continue to be throughout my career. I also feel fortunate to have been able to look back and benefit from a long family tradition that values hard work and education. I would like to thank my grandparents, Israel “Isidor” Artzi

and Nathan Svarinski, alehem ha-shalom. They both have been incredible sources of inspiration. I am honored to be able to follow in their footsteps. I am grateful to the endless support of my wife, Julija Lazutkaite. This degree would not have happened without your endless support, care and personal sacrifice. I am privileged for having shared this journey with you. I also wish to thank the limitless support, care and worry of my mother, Malca Artzi, siblings, Eran and Ronnie Artzi, and grandmothers, Lea Artzi and Hinda Svarinski. Special thanks to Hila Grinberger and Kobi Bensimon, my Seattle family, for support, encouragement and familial wisdom. The support of my family, both in Seattle and Israel, has been paramount in enabling me to pursue this degree.

My graduate career and the work was supported in part by a Microsoft PhD Fellowship, an Amazon AWS in Education Grant, a Yahoo Key Scientific Award, the Dora Zee Ling CSE Endowed Fellowship and funding from DARAPA and NSF.

DEDICATION

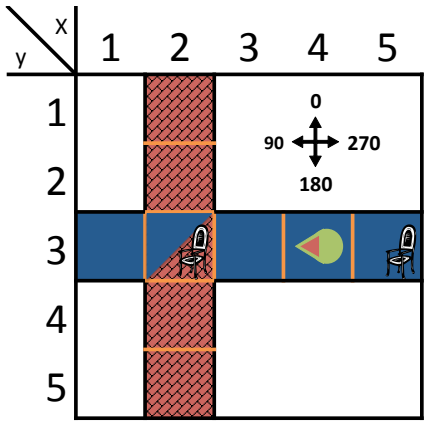
To Jul

Chapter 1

INTRODUCTION

Robust and ubiquitous conversational interfaces have the potential to transform how users access and use computers. However, developing natural language learning and understanding algorithms for interactive scenarios, such as robot control and question answering, remains a challenging problem. To accurately interpret user requests, for example when instructing a robot, a system must jointly reason about word meaning, grammatical structure, conversation history and world state. While challenging, such situated interactions also provide promising opportunities for both learning and interpretation. For example, during learning, a robot can experiment with different hypotheses in the environment and observe the results, and while following instructions, the robot can similarly prioritize interpretations given its perception of the world. In both cases, interpretations that result in infeasible actions are likely to be wrong, while those that correlate well with the perceived environment are more likely to be correct.

In this thesis, we develop a framework for learning to map sentences to system responses via a latent logical meaning representation, a procedure often referred to as *semantic parsing*. We study multiple aspects of this problem: learning without linguistic annotation, incorporating situated cues when reasoning about language meaning and jointly reasoning about compositional and non-compositional aspects of language meaning. We use Combinatory Categorial Grammar (CCG; Steedman, 1996, 2000; Steedman and Baldridge, 2003) to map sentences to lambda calculus representation of their meaning. Learning requires solving two problems: inducing the structure of the grammar and estimating the model parameters. Rather then relying on labor-intensive linguistic expertise, we demonstrate learning from system failures and subsequent recovery efforts in dialog systems and from human demonstrations in a situated learning regime. When integrated into a complete system, we show how incorporating situated cues to jointly reason about language



(1) *move forward twice to the chair*
$\lambda a.\text{move}(a) \wedge \text{dir}(a, \text{FORWARD}) \wedge \text{len}(a, 2) \wedge \text{to}(a, \iota x.(\lambda x.\text{chair}(x)))$
$\langle \text{MOVE}, \text{MOVE} \rangle$

(2) *at the intersection turn left to face the brick hall*
$\lambda a.\text{pre}(a, \iota(\lambda x.\text{intersection}(x))) \wedge \text{turn}(a) \wedge \text{dir}(a, \text{LEFT}) \wedge$
$\text{post}(a, \text{front}(\text{YOU}, \iota(\lambda x.\text{blue}(x) \wedge \text{hall}(x))))$
$\langle \text{MOVE}, \text{MOVE}, \text{LEFT} \rangle$

(3) *move to the chair in the first junction*
$\lambda a.\text{move}(a) \wedge \text{to}(a, \iota(\lambda x.\text{sofa}(x))) \wedge \text{intersect}(\text{order}(\lambda y.\text{intersection}(y), \text{frontdist}, 1), x)$
$\langle \text{MOVE}, \text{MOVE} \rangle$

Figure 1.1: Schematic diagram of a map environment and examples of navigation instructions paired with their logical forms and demonstrations of system response, navigation traces in this case. The environment includes an agent (marked in green and facing left), two hallways and two chairs. This type of environment is described in Chapter 5.

meaning and system response results in significantly better performance, for example, when following navigation instructions. Finally, in response to challenges presented in recently annotated resources, we introduce a new, scalable CCG induction approach and a joint model of both compositional and non-compositional semantics.

As an example in the navigation domain, consider the map diagram and instructions in Fig-

ure 1.1. Each instruction is presented with its logical form meaning representation and a demonstration of the desired system behavior. While annotating such logical forms is labor intensive and requires linguistic expertise, navigation traces can be easily obtained by recording the actions of non-expert users in the environment. For each instruction, its logical form represents the constraints on the type of actions the agent is expected to perform. For example, for Instruction 1, the logical form expresses constraints about the type of action ($\mathrm{move}(a)$), its direction ($\mathrm{dir}(a, \mathrm{FORWARD})$), length ($\mathrm{len}(a, 2)$) and destination ($\mathrm{to}(a, \iota(\lambda x.\mathrm{chair}(x)))$). Its demonstration ($\langle \mathrm{MOVE}, \mathrm{MOVE} \rangle$) indicates the expected behavior of the agent from its current position: execute the MOVE action two times. However, in a different position, the agent's response is likely to be different (e.g., different number of steps).

Recovering such logical representations and reasoning about their execution is widely applicable. Logical forms can represent questions in natural language interfaces to databases, where the execution will be a database query. In dialog systems, logic can represent the meaning of user requests, which are then evaluated to plan the system response. While we focus on specific applications in this thesis, we expect that the lessons learned will be useful for the general design and deployment of natural language interfaces.

1.1 Challenges

Responding appropriately to natural language requests and queries requires reasoning at multiple levels, starting from deciding about the meaning of words and continuing with combining them to obtain the meaning of complete sentences. For a system to be able to learn how to make such decision from easily gathered data, such as non-expert annotations, it must be able to reason across many training examples to identify the best sequence of linguistic decisions. The challenges addressed in this thesis are classified into five categories:

(C1) Lexical: The system must be able to identify the meaning of words. For example, the word *move* (Instruction 1, Figure 1.1) suggests that the predicate move may appear in the logical form, while the word *first* (Instruction 3) is associated with the order construction, which

indicates that a given set of entities ($\lambda y.\text{intersection}(y)$, which is introduced by *junction*) is to be indexed by the function frontdist and the first element is to be returned.

(C2) Parse selection: Given the meaning of all words in the sentence, the system must then select the best way to combine them and return the most likely meaning representation of the sentence. For example, to correctly respond to the instruction *move past a sofa or a chair and the hatrack in the red hall*, a system must choose from the many possible combinations of word meaning to select the most appropriate logical form.

(C3) Non-compositional Semantics: The parse structure of a sentence does not recover all of its semantic meaning. For example, pronouns express distant references that are not fully resolved in the parse structure. To correctly respond to user requests, such as *face the chair and move to it*, the system must resolve distant references, for example, to understand that *it* and *the chair* refer to the same object.

(C4) Situated signals: The system must use situational cues to reason about language meaning. For example, consider the phrase *the chair* in Instruction 1. It may refer to each of the two chairs in the environment. However, the system must select the more likely reference given the agent's state and the complete sentence to correctly follow the instruction.

(C5) Annotation: While logical form annotation provides a strong learning signal, annotating many sentences with such rich representations is labor-intensive and requires specific linguistic expertise. The system, therefore, must be able to learn from significantly weaker forms of supervision and rely on many sentences to generalize. For example, learning from demonstrations, as illustrated in Figure 1.1, requires the system to learn different representations for Instructions 1 and 3, despite their identical annotation.

1.2 Previous Approaches

Building systems that can understand natural language has been studied extensively. In this section, to motivate the focus on this thesis, we briefly review three strands of research to highlight the state of the art and common approaches prior to our work. First, we discuss a learning approach that relies on linguistic annotation to learn to map sentences to logical meaning representations. We then discuss an alternative approach for language understanding that focuses on building complete systems, requires less annotation, but limits the expressivity of recovered representations. Finally, we briefly characterize existing efforts to recover non-compositional semantics. In Chapter 3, we expand this discussion and provide a more complete overview of related work.

The dominant paradigm for learning to map sentences to logical forms in previous work assumed access to training data of sentences annotated with logical forms. For example, in the navigation domain in Figure 1.1, each sentence will have to be manually annotated with the logical form representation provided. While annotators are not required to specify every decision the system is expected to make, it is necessary for them to posses extensive knowledge of the domain (e.g., the representation of flights for travel planning) and substantial linguistic background. Zettlemoyer and Collins (2005) proposed a two-stage learning approach to induce CCG from sentences annotated with lambda calculus logical expressions. Their approach was able to learn the meaning of words and the decisions required to combine them together to complete logical forms. However, the annotation effort required made it extremely challenging to build systems for real-life applications. Each such system would have required training annotators with linguistic expertise in the intricacies of the domain and then have them annotate thousands of sentences, a process that is likely to take months.

In contrast, Branavan et al. (2009) focused on learning to execute instructions in the Microsoft Windows domain without linguistic annotation. Instead, they used reinforcement learning with a carefully designed reward function to measure task completion during learning. Roughly speaking, instead of annotating sentences with linguistic structures, they learned from demonstrations of executions. Such annotation is significantly easier to get. It only requires the annotator to be

familiar with the system and follow the given instructions. Their actions can then be recorded to guide learning. This effort does not require any linguistic knowledge and in many cases, including the Windows domain, relies on skills that are relatively common. However, they focused on recovering relatively simple representations. Given a sentence, the learned model mapped spans in the sentence to small frame-like representations (e.g., $\text{left\_click}(\text{OK\_button})$). Each frame included the type of the action and its arguments. Since the data they collected displayed relatively simple language, this representation was sufficient for a significant portion of their corpus. However, it was unable to represent much of the phenomena commonly observed in more complex corpora, such as those studied by Zettlemoyer and Collins (2005).

Recovering non-compositional meaning has been studied extensively. Co-reference resolution, where the goal is to identify all text spans that refer to the same entity, is the most popular task addressing those aspects of meaning. Since a complete overview of the literature on this problem is beyond the scope of this thesis, we refer the reader to Ng (2010) for a recent survey. In the semantic parsing literature, recovering non-compositional semantics has received relatively little attention. A notable exception is Zettlemoyer and Collins (2009). They recover context-dependent logical forms in conversational interactions in the travel planning domain and focus on cross-sentential aspects of non-compositional meaning.

The techniques we develop in this thesis are designed combine the benefits of the above threads of research. We demonstrate learning of expressive CCG grammars, such as those induced by Zettlemoyer and Collins (2005), while relying on easier to obtain annotation. In Chapter 4, we show how the interaction generated when a system tries to recover in conversations provides a learning signal to learn a CCG without any explicit annotation effort. Following Branavan et al. (2009), we describe in Chapter 5 an approach to use demonstrations of desired system behavior to induce similar grammars, enabling us to recover significantly more expressive representations of language meaning. We also study the recovery of non-compositional semantic meaning. However, in contrast to most existing work, we study it within a semantic parsing framework, where we recover representations that capture both compositional and non-compositional meaning. Our focus on intra-sentence dependencies is complementary to Zettlemoyer and Collins (2009), which

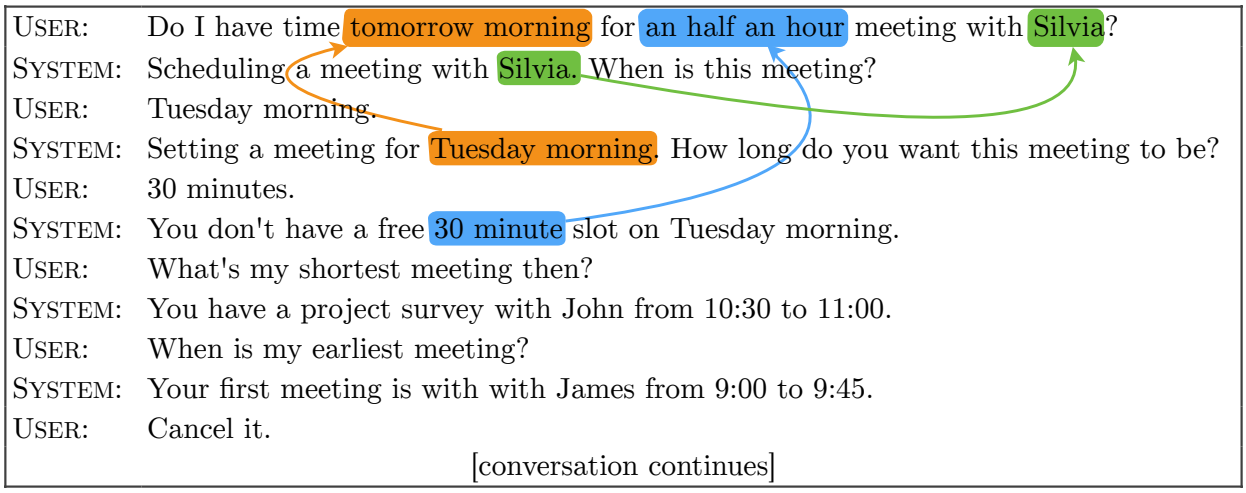


Figure 1.2: A hypothetical conversation between a user and a natural language calendar system. The arrows show information the system was able to recover after failing to fully understand the first user request and how it aligns to this request. This alignment can be used to learn a semantic parser without any annotation effort.

emphasizes inter-sentence dependencies. Additionally, we study this problem in the context of broad-coverage semantic parsing, in contrast to their domain-specific focus.

1.3 Overview of Approach

In this thesis, we develop a framework for learning and inference of CCG semantic parsers. We focus on three major aspects of the problem: the type of supervision required for learning, the cues considered during inference and the scalability of our learning techniques. The remainder of this section provides an overview of the primary contributions of this thesis.

1.3.1 Learning without Linguistic Annotation

We first consider learning a CCG semantic parser from recorded logs of conversations between users and an automated dialog system (Chapter 4). Such systems allow users to perform pre-specified tasks, such as scheduling, through natural language interaction. For example, Figure 1.2 shows a hypothetical conversation between a user and a natural language calendar system. In such

conversations, the system often fails to understand the user. However, since it is in an interaction with the user it must attempt to recover by asking for specific questions and clarifications. In the example conversation, the system failed to recover the time and duration of the meeting, but it quickly tried to recover this information by asking specific questions. We observe that this recovery contains informative cues about the meaning of the initial user request the system failed to understand. Learning will then require recordings of such conversations along with the internal system state, thereby removing the need for expert labeling or any explicit annotation effort.

The first contribution of this thesis is a learning algorithm for learning from recordings of such conversations. The algorithm takes user utterances paired with their conversations. For example, the utterance *Do I have time tomorrow morning for an half an hour meeting with Silvia?* will be paired with the conversation in Figure 1.2. The output is a weighted CCG that can map new sentences to their logical form meaning representation. CCG has been studied extensively and was designed to model a wide range of linguistic phenomena while providing a transparent interface between syntax and semantics. Using a weighted CCG gives our models the ability to reason about the probabilities of different logical forms for a given sentence. Such probabilities give us a principled way to select the best analysis, reason about ambiguity and robustly recover from noise in the data.

During learning, we recover two parts of the weighted CCG: a lexicon and a parameter vector. The lexicon pairs words with CCG categories, which include both syntactic and semantic information. For example, the entry $meeting \vdash N : \lambda x.\text{meeting}(x)$ indicates that the word *meeting* may syntactically be a noun, indicated by the syntax N, with the meaning represented by $\lambda x.\text{meeting}(x)$, a lambda calculus expression defining the set of all meetings in the system. The parameter vector includes real-valued weights, one for each feature in the system. Combining the lexicon and the parameters, we are able to solve both the lexical and parse selection challenges mentioned above (Section 1.1). The lexicon represents the meaning of words and the parameter vector solves the parse selection problem by scoring each parse.

Weighted CCG induction without annotated logical forms is extremely challenging. The learning algorithm must induce lexical entries and estimate the correct parameters without exactly

knowing what structures it should recover. Instead, we give it access to a loss function which measures the alignment between hypothesized logical forms and system utterances. This choice signifies a departure from previous research, where learning assumed access to annotated logical forms (Zettlemoyer and Collins, 2005), while still jointly solving both the grammar induction and parameter estimation learning problems.

Recorded conversations allow us to learn without any annotation effort. However, conversations are not always available, for example, when a system is first deployed before users had a chance to use it or in systems that rely on one-sided interactions. In such scenarios, demonstrations of system behavior provide an interesting opportunity for learning. For example, consider the robot navigation system in Figure 1.1. While annotating instructions with their logical form representation is labor-intensive and requires linguistic expertise, even non-expert users can demonstrate what the system is expected to do given an instruction. For instance, in the first instruction, given the starting position, a non-expert can easily describe the path the system must take given the instruction *move forward twice to chair*. Generating the training data then requires collecting instructions in specific situations and asking annotators to follow them in the environment, for example, by controlling the robot or by walking on their own. For the example sentence, the recorded demonstration in the given environment is $\langle \text{MOVE}, \text{MOVE} \rangle$.

We treat learning from such demonstrations as a special case of using a loss function (Chapter 5), as in the case of learning conversations. We use a simple binary loss function that indicates if a given hypothesis fits the demonstration or not. To evaluate logical forms using this loss function we constantly execute hypothesized logical forms during training in the environment. This process maps logical forms to their execution, which can then be directly compared to the provided demonstration. We use this learning signal to induce a weighted CCG, including both the lexicon and the parameter vector.

Learning the lexicon from demonstrations creates an interesting challenge. When using conversations (Chapter 4), the recorded state of the system provides the set of symbols that may be used for new lexical entries. Our algorithm then uses this set to hypothesize the set of potential lexical entries and select the ones to add to the lexicon. However, demonstrations lack any sym-

bolic supervision and therefore we are forced to consider the full space of lexical entries, including all symbols that are available to the system. This creates an extremely large set, and previous techniques to select the correct entries are overwhelmed and fail to learn an effective lexicon. Therefore, we propose a new approach (Chapter 5) to gradually prune such large sets of potential lexical entries with a coarse-to-fine parsing algorithm.

1.3.2 Situated Reasoning

The meaning of language is closely related to its context. For example, in instructional language, the meaning of a sentence may vary depending on the current state of the world. Consider the navigation domain and the two similar environments in Figure 1.3. Given the instruction *move until you see a chair and a lamp in the red hall*, multiple reading are available. Specifically, the system must decide if the prepositional phrase *in the red hall* is coordinated to modify both *chair* and *lamp*, or if it modifies *lamp* only. The two logical forms in the figure illustrate the different choices. In the environment on the left, where both the lamp and chair are in the brick hallway, the prepositional phrase may modify both, as seen in the left logical form. However, in the environment on the right, such a reading will fail to execute. Therefore, the correct logical form does not coordinate *in the red hall*. Given a grammar that can generate both readings, the agent must use its observations to select the appropriate logical form and execute it in the environment.

Furthermore, in learned models, where limited and noisy training data often results in grammars that lead to many wrong parses, using contextual cues can provide further evidence to better choose the correct logical form representation. For example, a learned grammar may include two entries for the word *chair*:

$$chair \vdash N : \lambda x.\text{chair}(x)$$
$$chair \vdash N : \lambda x.\text{hatrack}(x)$$

The first entry correctly represents *chair* as the set of chairs in the environment, while the second incorrectly maps it to the set of hatracks in the environment. Considering the set of observed objects

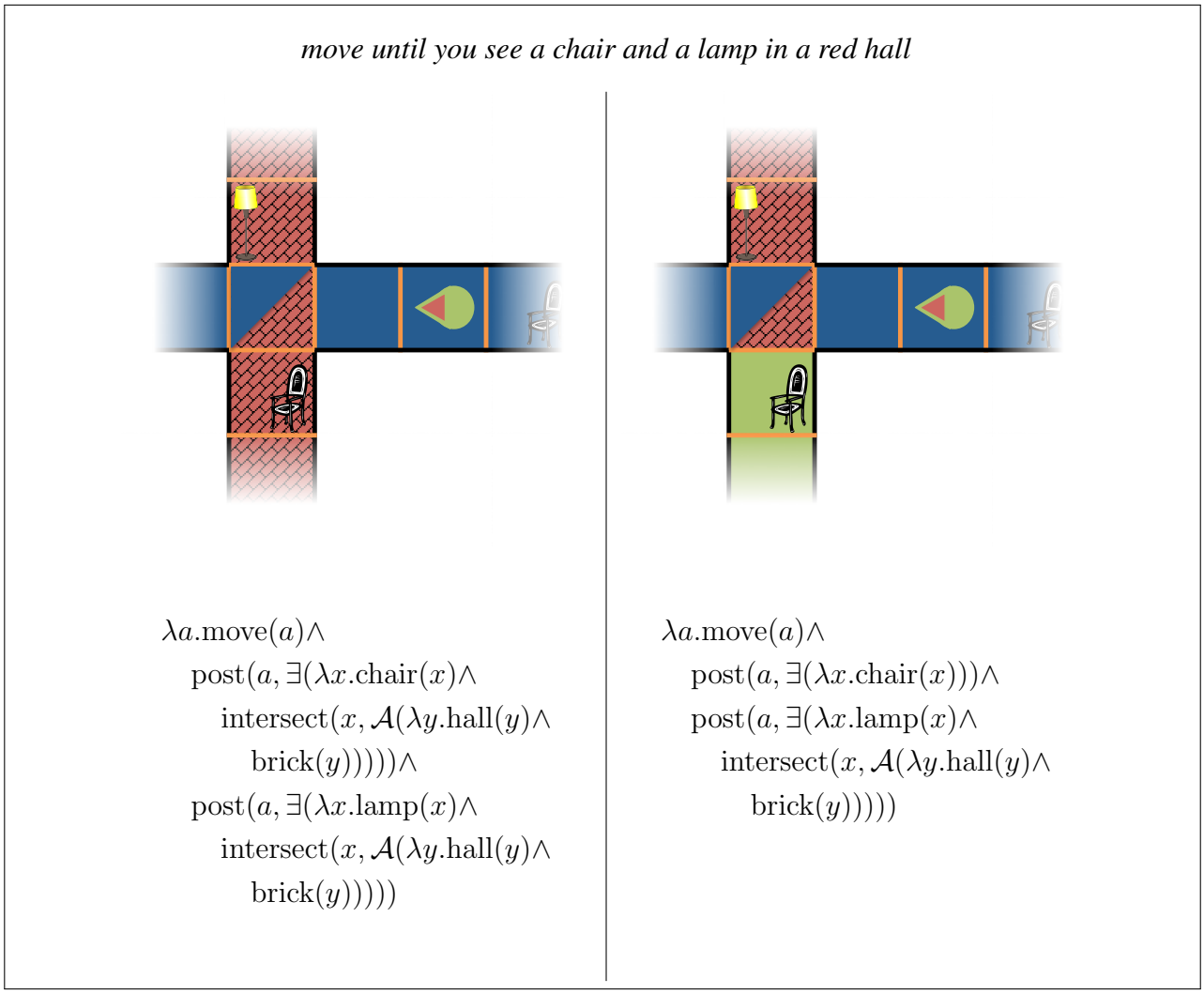


Figure 1.3: Schematic diagrams of two similar environments. Given the instruction above, different logical form representations are required to move to the intersection in front of the agent. In the environment on the left, the prepositional phrase *in a red hall* is coordinated and applied both to the chair and lamp. In contrast, on the right, a different attachment is required, and the phrase is not coordinated, applying only to the lamp.

allows an agent to effectively discard parses incompatible with the environment. For example, in the environments in Figure 1.3, no hatrack is observed. Therefore, the agent can easily choose to ignore these parses regardless of their score, drastically narrowing down the space of competing logical forms to consider.

To consider such contextual cues, we jointly reason about sentence meaning and its execution in the environment (Chapter 5). During parsing, we execute all competing logical forms in the environment and observe the results. When the agent is unable to successfully execute a logical form, for example, when it includes a condition about hatracks and none are observed, this logical form is discarded regardless of its score. This allows our system to augment its learned parameters with contextual cues. We demonstrate how this approach leads to significant improvements in performance.

1.3.3 Learning Compact Lexicons

Learning the grammar enables the system to recover the semantics of complex sentences. However, this process also results in large and noisy lexicons, especially when learning without annotated logical forms. For example, in the navigation domain, while the algorithm correctly learns that the word *chair* may be mapped to the categories $N : \lambda x.\text{chair}(x)$ and $N : \lambda x.\text{sofa}(x)$, it also learn many other entries, including erroneous uses of *chair* as adjective $ADJ : \lambda x.\text{chair}(x)$, noun phrase $NP : \mathcal{A}(\lambda x.\text{corner}(x))$ or adverbial phrase $AP : \lambda a.\text{len}(a, 3)$. The reason behind such bad learning decisions is the greedy nature of our learning algorithm. The learner considers one sample at a time and decides if to update the lexicon based on this sample alone. Instead, we propose to consider global corpus statistics when adding entries to the lexicon and show how this results in more compact lexicons and increased performance.

We present a batch algorithm focused on controlling the size of the lexicon when learning CCG semantic parsers (Chapter 6). Because we make updates only after processing the entire training set, we can take corpus-wide statistics into account before each lexicon update. To explicitly control the size of the lexicon, we adopt two complementary strategies: *voting* and *pruning*. First, we consider the lexical evidence each sample provides as a vote towards potential entries. We describe two voting strategies for deciding which entries to add to the model lexicon. For example, from the example entries for the word *chair* above, we will first introduce $N : \lambda x.\text{chair}(x)$ into the lexicon since it is supported by a large number of training samples. In a later iterations, we will introduce $N : \lambda x.\text{sofa}(x)$, which is required by fewer training samples, but is still supported

by multiple samples. Once the correct entries are learned, the erroneous ones are less likely to be considered, since the sentences they were originally derived from will already have an existing high quality analysis. Second, even though we use voting to only conservatively add new lexicon entries, we also prune existing entries if they are no longer necessary for parsing the training data. These steps are incorporated into the learning framework, allowing us to apply stricter criteria for lexicon expansion while maintaining a single learning algorithm. We demonstrate how this process results in up to 70% reduction in lexicon size. The learned lexicons are significantly less noisy and lead to better task-completion performance.

1.3.4 Broad-coverage Semantic Parsing

Recently, the Abstract Meaning Representation (AMR; Banarescu et al., 2013) was proposed as a general-purpose meaning representation language for broad-coverage text, and work is ongoing to study its use for a variety of applications, such as machine translation (Jones et al., 2012) and summarization (Liu et al., 2015). For example, Figure 1.4 shows the AMR for the sentence *Akkermans was working with the Storimans and he suffered minor injuries*. The AMR meaning bank provides a large new corpus that, for the first time, enables us to study the problem of grammar induction for broad-coverage semantic parsing. However, it also presents significant challenges for existing algorithms, including much longer sentences, more complex syntactic phenomena and increased use of non-compositional semantics, such as within-sentence coreference. We present a scalable CCG grammar induction approach to learn a joint model of both compositional and non-compositional semantics (Chapter 7).

We map sentences to AMR structures in a two-stage process. First, we use CCG to construct lambda-calculus representations of the compositional aspects of the sentence. To use CCG for AMR parsing we define a simple encoding for AMRs in lambda calculus. However, using CCG to construct such logical forms requires a new mechanism for non-compositional reasoning, for example to model the long-range anaphoric dependency introduced by *he* in Figure 1.4. To represent such dependencies while maintaining a relatively compact grammar, we follow Steedman's (2011) use of generalized Skolem terms, a mechanism to allow global distant references in lambda calcu-

Sentence: *Akkermans was working with the Storimans and he suffered minor injuries.*

AMR:
(*a*/and
:op1 (*w*/work
:ARG0 (*p*/person
:name (*n*/name :op1 "Akkermans"))
:ARG3 (*p2*/person
:name (*n2*/name :op1 "Storimans")))
:op2 (*i*/injure-01
:ARG1 **p**
:degree (*m*/minor)))

ULF:
$\mathcal{A}_1(\lambda a.\text{and}(a) \wedge$
$\text{op1}(\mathcal{A}_2(\lambda w.\text{work}(w) \wedge$
$\text{ARG0}(w, \mathcal{A}_3(\lambda p.\text{person}(p) \wedge$
$\text{name}(p, \mathcal{A}_4(\lambda n.\text{name}(n) \wedge \text{op}(n, \text{AKKERMANS}))))) \wedge$
$\text{ARG3}(w, \mathcal{A}_5(\lambda r.\text{person}(r) \wedge$
$\text{name}(r, \mathcal{A}_6(\lambda q.\text{name}(q) \wedge \text{op}(q, \text{STORIMANS}))))))) \wedge$
$\text{op2}(a, \mathcal{A}_7(\lambda i.\text{injure-01}(i) \wedge$
$\text{ARG1}(i, \mathcal{R}(\text{ID})) \wedge$
$\text{degree}(i, \mathcal{A}_8(\lambda m.\text{minor}(m))))))$

LF:
$\mathcal{A}_1(\lambda a.\text{and}(a) \wedge$
$\text{op1}(\mathcal{A}_2(\lambda w.\text{work}(w) \wedge$
$\text{ARG0}(w, \mathcal{A}_3(\lambda p.\text{person}(p) \wedge$
$\text{name}(p, \mathcal{A}_4(\lambda n.\text{name}(n) \wedge \text{op}(n, \text{AKKERMANS}))))) \wedge$
$\text{ARG3}(w, \mathcal{A}_5(\lambda r.\text{person}(r) \wedge$
$\text{name}(r, \mathcal{A}_6(\lambda q.\text{name}(q) \wedge \text{op}(q, \text{STORIMANS}))))))) \wedge$
$\text{op2}(a, \mathcal{A}_7(\lambda i.\text{injure-01}(i) \wedge$
$\text{ARG1}(i, \mathcal{R}(\mathbf{3})) \wedge$
$\text{degree}(i, \mathcal{A}_8(\lambda m.\text{minor}(m))))))$

Figure 1.4: Example sentence and its AMR, underspecified logical form (ULF) and complete logical form representation (LF). The colored symbol **p** represents a distant reference introduced by *he* and resolved to the entity introduced by *Akkermans*. In ULF, this reference is is unresolved and represented by ID, which is specified to **3** in LF.

lus. We then allow the CCG derivation to mark when non-compositional reasoning is required with underspecified placeholders. Figure 1.4 shows an underspecified logical form (ULF) that would be

constructed by the grammar with the placeholder ID indicating an unresolved anaphoric reference. These placeholders are resolved by a factor graph model that is defined over the output logical form and models which part of it they refer to, for example to find the referent for a pronoun. In the example, the final logical form (LF) fully specifies the reference using the identifier 3.

We also introduce a new CCG grammar induction algorithm which incorporates ideas from previous algorithms (Zettlemoyer and Collins, 2005; Kwiatkowski et al., 2010) in a way that scales to the longer sentences and more varied syntactic constructions observed in newswire text. During lexical generation, the algorithm first attempts to use a set of templates to hypothesize new lexical entries. It then attempts to combine bottom-up parsing with top-down recursive splitting to select the best entries and learn new templates for complex syntactic and semantic phenomena, which are re-used in later sentences to hypothesize new entries. While previous algorithms (e.g., Zettlemoyer and Collins, 2005) have assumed the existence of a grammar that can parse nearly every sentence to update its parameters, this does not hold for AMR Bank. Due to sentence complexity and search errors during parsing, our model cannot produce fully correct logical forms for a significant portion of the training data. To learn from as much of the data as possible and accelerate learning, we adopt an early update strategy that is able to generate effective updates from partially correct analyses.

This new learning technique is able to induce CCG lexicons at scale. When combined with our modeling contributions, it results in new state-of-the-art performance for AMR parsing, as evaluated on the AMR Bank.

1.4 Contributions

This thesis presents new modeling and learning developments for recovering logical form representations of natural language sentences. The primary contributions include:

- A loss-driven learning algorithm for weighted CCG induction and parameter estimation without linguistic annotation.

- A technique to consider situated cues when mapping sentences to context-dependent logical form representations of their meaning.

- An approach to learn compact CCG lexicons by considering global statistics for lexicon learning decisions.

- A joint model for compositional and non-compositional semantics to map sentences to logical form meaning representations.

- A scalable learning algorithm for broad-coverage semantic parsing that combines grammar induction and parameter estimation.

1.5 Thesis Outline

Chapter 2 provides background material, including lambda calculus, CCG, factored lexicon representation, weighted CCG, supervised CCG induction and generalized skolem terms. Chapter 3 reviews related work, including work on CCG, formal semantics and learning for semantic parsing. Chapter 4 describes our approach for learning from conversations. Chapter 5 extends this approach to learn from demonstrations and discusses situated reasoning. Chapter 6 presents algorithms to learn compact CCG lexicons. Chapter 7 describes our approach for broad-coverage semantic parsing, including our joint model for compositional and non-compositional semantics and our learning algorithms. Finally, Chapter 8 concludes and outlines possible directions for future work.

Chapter 2

BACKGROUND

2.1 Simply-typed Lambda Calculus

We represent the meaning of sentences using simply-typed lambda calculus logical expressions. Lambda calculus (Church, 1932) is a formal system to represent computation. It uses function abstraction, application, variable binding and substitution. It is based on the principle of compositionality (Boole, 1854), which defines the meaning of complex expressions to be determined by the meaning of their sub-expressions and the rules combining them. Simply-typed lambda calculus (Church, 1940) extends lambda calculus by adding a typing hierarchy with a single inheritance relation. We present here a short informal overview of simply-typed lambda calculus and refer the reader to Chapter 2 in Carpenter (1997) for a formal introduction.

A type is (a) a primitive symbol α or, (b) a combination $\langle \alpha, \beta \rangle$ of two types, α and β, with the operator $\langle , \rangle$. A type that is a combination of two types is called *complex*. We use four base atomic types: entity e, truth-value t, numeral n, event ev and skolem ID id.[1] Base types are inherited in a domain-specific typing hierarchy. For example, in a travel domain, loc may be used for locations and will then inherit e, ci may then be used for cities and will inherit loc. Figure 2.1 shows a partial typing hierarchy for a travel planning domain. Atomic types are combined to create complex types. Functions from entities (e-typed objects) to boolean t-typed objects (representing *true* or *false*) have the type $\langle e, t \rangle$, a functional type where the domain of the function is e-typed and the range is t-typed. We say that functions with the type $\langle \alpha, t \rangle$, where α is a type, define a set of α-typed objects that includes all objects for which the function evaluates to true. The $\langle , \rangle$ operator can be used to create arbitrarily complex types. For example, $\langle e, \langle e, t \rangle \rangle$-typed functions take a

[1] In Chapter 5, instead of n, we use the primitive type m for meta entities, including numbers and directions. Skolem IDs are defined in Section 2.6

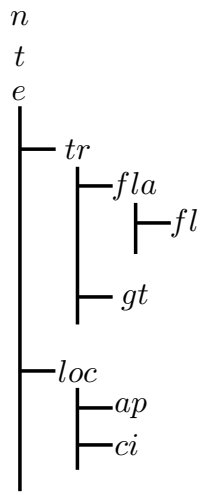


Figure 2.1: Partial typing hierarchy for a travel planning domain. This domain includes three basic types: numeral n, truth-value t and entity e. Edges indicate inheritance. Transportation tr inherits from e. Flight fl inherits from air travel fla, which inherits from tr. Ground transportation gt is a type of transportation tr. Airport ap and city ci are both locations an inherit from loc, which inherits from e.

e-typed argument to return a $\langle e,t\rangle$-typed function, $\langle\langle e,t\rangle,e\rangle$-typed functions expect $\langle e,t\rangle$-typed functions and return e-typed objects. The ability to define functions that consume other functions is especially useful for representing the meaning of determiners, which select an entity from a set of entities, and superlatives, which sort entities using a ranking function to select the one with a maximal value. We say that a logical form is *functional* if it has a complex type.

Simply-typed lambda calculus is defined using the following four base cases:

- A **logical constants** c_α, where c is an atomic symbol and α is its type. We often use constants to represent entities, such as NYC_{ci} and JFK_{ap}, or functions, such as $\mathrm{flight}_{\langle fl,t\rangle}$ and $\mathrm{departure\_date}_{\langle tr,\langle date,t\rangle\rangle}$.

- A **variable** x_α, where x is an atomic symbol and α is its type. We denote variables with lowercase letters x_α, y_α, z_α, etc.

- A **literal** $(P_{\langle\alpha,\beta\rangle})(A_{\alpha'})$, where the *predicate* $P_{\langle\alpha,\beta\rangle}$ and *argument* $A_{\alpha'}$ are logical expressions

and α' extends α. The type of the literal is β. For example, $(\text{city}_{\langle loc,t\rangle})(\text{NYC}_{city})$ is a literal. Literals represent function application, as we define below, and can be recursive to allow for functions that take multiple arguments. For example, $((\text{located\_in}_{\langle loc,\langle loc,t\rangle\rangle})(\text{JFK}_{ap}))(\text{NYC}_{ci})$ is a binary literal. We show later how to simplify the writing of such literals.

- A **lambda term** $\lambda x_\alpha.A_\beta$, where x_α is a variable of type α, A_β is a logical expression of type β and the lambda term has the type $\langle\alpha,\beta\rangle$. For example, $\lambda x_e.((\text{located\_in}_{\langle loc,\langle loc,t\rangle\rangle})(x_e))(\text{LA}_{ci})$ is a lambda term.

We use two basic logical operations: *application* and *composition*. Application is denoted as a literal (i.e., $(P_{\langle\alpha,k\beta\rangle})(A_{\alpha'})$). If $P_{\langle\alpha,\beta\rangle}$ is a lambda term $\lambda x_\alpha.B_\beta$, it can be reduced to $B_\beta[x_\alpha \mapsto A_{\alpha'}]$, where $[x_\alpha \mapsto A_{\alpha'}]$ indicates that all occurrences of x_α in B_β are substituted with $A_{\alpha'}$. Function composition can be derived using application. Given a $g_{\langle\alpha,\beta\rangle} = \lambda x_\alpha.G_\beta$ and $f_{\langle\beta,\gamma\rangle} = \lambda y_\beta.F_\gamma$, the composition $(f_{\langle\beta,\gamma\rangle} \cdot g_{\langle\alpha,\beta\rangle})_{\langle\alpha,\gamma\rangle}$ can be derived with the following steps:

1. Let A_α be a α-typed logical expression.

2. $g_{\langle\alpha,\beta\rangle}(A_\alpha) = (\lambda x_\alpha.G_\beta)(A_\alpha) = G_\beta[x_\alpha \mapsto A_\alpha]$

3. $f_{\langle\beta,\gamma\rangle}(g_{\langle\alpha,\beta\rangle}(A_\alpha)) = (\lambda y_\beta.F)(G_\beta[x_\alpha \mapsto A_\alpha]) = F[y_\beta \mapsto G_\beta[x_\alpha \mapsto A_\alpha]]$

4. $\lambda z_\alpha.f_{\langle\beta,\gamma\rangle}(g_{\langle\alpha,\beta\rangle}(A_\alpha))[A_\alpha \mapsto z_\alpha] = \lambda z_\alpha.F[y_\beta \mapsto G_\beta[x_\alpha \mapsto A_\alpha]][A_\alpha \mapsto z_\alpha] =$
   $\lambda z_\alpha.F[y_\beta \mapsto G_\beta[x_\alpha \mapsto z_\alpha]] = (f_{\langle\beta,\gamma\rangle} \cdot g_{\langle\alpha,\beta\rangle})_{\langle\alpha,\gamma\rangle}$

To simplify out notation, we use the following syntactic sugar:

- Literals with multiple arguments:

$$(\ldots((P_{\langle\alpha_1,\langle\alpha_2,\ldots\langle\alpha_n,\beta\rangle\ldots\rangle})(A^1_{\alpha'_1}))(A^2_{\alpha'_2})\ldots)(A^n_{\alpha'_n}) \equiv P_{\langle\alpha_1,\langle\alpha_2,\ldots\langle\alpha_n,\beta\rangle\ldots\rangle}(A^1_{\alpha'_1},\ldots,A^n_{\alpha'_n}) \ .$$

- Nested binary conjunctions $\wedge$ (and disjunctions $\vee$):

$$\wedge_{\langle t,\langle t,t\rangle\rangle}(A_t^1, \wedge_{\langle t,\langle t,t\rangle\rangle}(A_t^2, \wedge_{\langle t,\langle t,t\rangle\rangle}(A_t^3, \cdots \wedge_{\langle t,\langle t,t\rangle\rangle} (A_t^{n-1}, A_t^n)\dots))) \equiv A_t^1 \wedge A_t^2 \wedge \cdots \wedge A_t^n \quad .$$

- Negation literals:

$$\neg(A_t) \equiv \neg A_t \quad .$$

Additionally, we omit all typing information for all logical expressions. We use the terms *logical form* and *logical expression* interchangeably.

To illustrate how lambda calculus represents sentence meaning, consider, for example, the meaning representation for *walk to the red chair in the intersection*:

$$\lambda a.\mathrm{move}(a) \wedge \mathrm{to}(a, \iota(\lambda y.\mathrm{chair}(y) \wedge \mathrm{intersect}(y, \iota(\lambda x.\mathrm{intersection}(x))) \wedge \mathrm{color}(y, \mathrm{RED}))) \quad ,$$

a $\langle ev, t\rangle$-typed logical form. It defines a set of all events (ev-typed objects) for which the body of the outer lambda term evaluates to true. The body of the lambda term is a conjunction of t-typed literals, each representing a boolean function that tests a property of the considered event. The literal $\mathrm{move}(a)$ tests the type of the event and $\mathrm{to}(a, \dots)$ tests the final position of the agent to match the evaluation of $\iota(\lambda y.\mathrm{chair}(y) \wedge \mathrm{intersect}(y, \iota(\lambda x.\mathrm{intersection}(x))) \wedge \mathrm{color}(y, \mathrm{RED}))$. Roughly speaking, ι (iota) represents the definite determiner and selects a single entity from the set defined by $\lambda y.\mathrm{chair}(y) \wedge \mathrm{intersect}(y, \iota(\lambda x.\mathrm{intersection}(x))) \wedge \mathrm{color}(y, \mathrm{RED})$. The constants chair, $\mathrm{intersection}$, color and $\mathrm{intersect}$ test various properties of e-typed entities and RED denotes the red color. We discuss our representation of imperatives in Section 5.4.2 and the ι in Section 5.4.1.

2.2 Combinatory Categorial Grammar

CCG is a linguistically motivated categorial formalism for modeling a wide range of language phenomena (Steedman, 1996, 2000). In CCG, parse tree nodes are categories, which are assigned to phrases (single words or n-grams) and combined to create a complete derivation. For example,

$S/NP : \lambda x.\lambda a.\text{move}(a) \wedge \text{direction}(a, x)$ is a CCG category describing an imperative verb phrase. The syntactic type S/NP indicates the category is expecting an argument of type NP on its right, and the returned category will have the syntax S. A syntactic type can be an atomic symbol (S or NP in the example), or a combination of the form $A\backslash B$ or A/B, where A and B are syntactic types. We say that a category is *complex* if its syntactic type is a combination, otherwise we say it is *simple*. The slash symbol indicates a directionality constraint. Roughly speaking, the forward slash / indicates that the category expects an argument on the right, where a backward slash \ indicates it expects an argument on the left. The exact interpretation of the slashes is determined by the combinatory rule used, as we describe below. The logical form in the category, a simply-typed lambda calculus expression in our case, represents its semantic meaning.

A CCG is defined by a lexicon and a set of combinators. The lexicon provides a mapping from phrases to categories. Figures 2.2 and 2.3 show two CCG parses. Parse trees are read top to bottom. Parsing starts by matching categories to phrases in the sentence using the lexicon. For example, the lexical entry *walk* $\vdash S : \lambda a.\text{move}(a)$ pairs the token *walk* with the category $S : \lambda a.\text{move}(a)$. Each intermediate parse node is constructed by applying one of a small set of binary CCG combinatory rules or unary operators. Parsing concludes with a logical form that captures the meaning of the complete sentence. We include two rules for function application, one for each direction:

Forward application:	$A/B : f \quad B : g$	$\Rightarrow \quad A : f(g)$	$(>)$
Backward application:	$B : g \quad A\backslash B : f$	$\Rightarrow \quad A : f(g)$	$(<)$

And two rules for function composition:

Forward composition:	$A/B : f \quad B/C : g$	$\Rightarrow \quad A/C : \lambda x.f(g(x))$	$(>\mathbf{B})$
Backward composition:	$B\backslash C : g \quad A\backslash B : f$	$\Rightarrow \quad A\backslash C : \lambda x.f(g(x))$	$(<\mathbf{B})$

Our use of unary rules depends on the domain and each chapter specifies the unary rules used. For example, we often use rules to shift adjectives, prepositional phrases and adverbials to a modifier

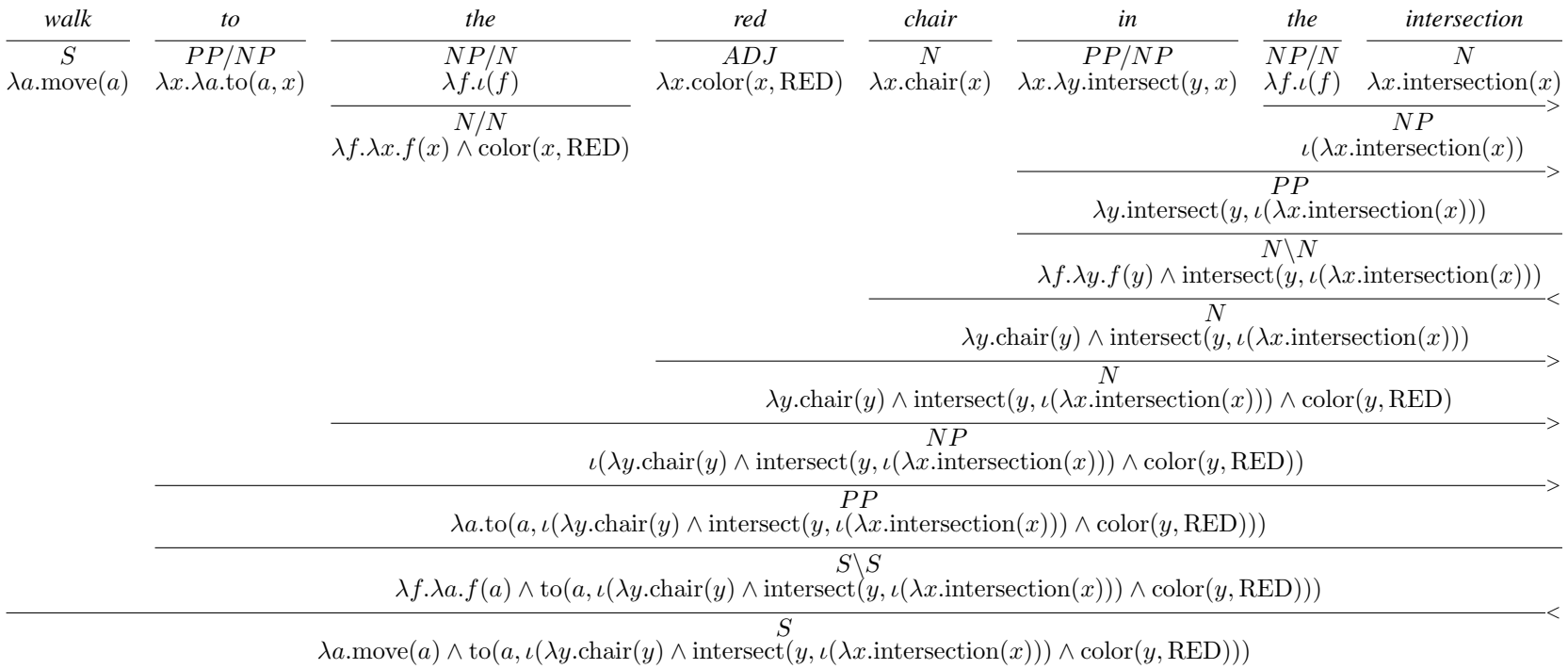


Figure 2.2: Example CCG parse tree for the sentence *walk to the red chair in the intersection*.

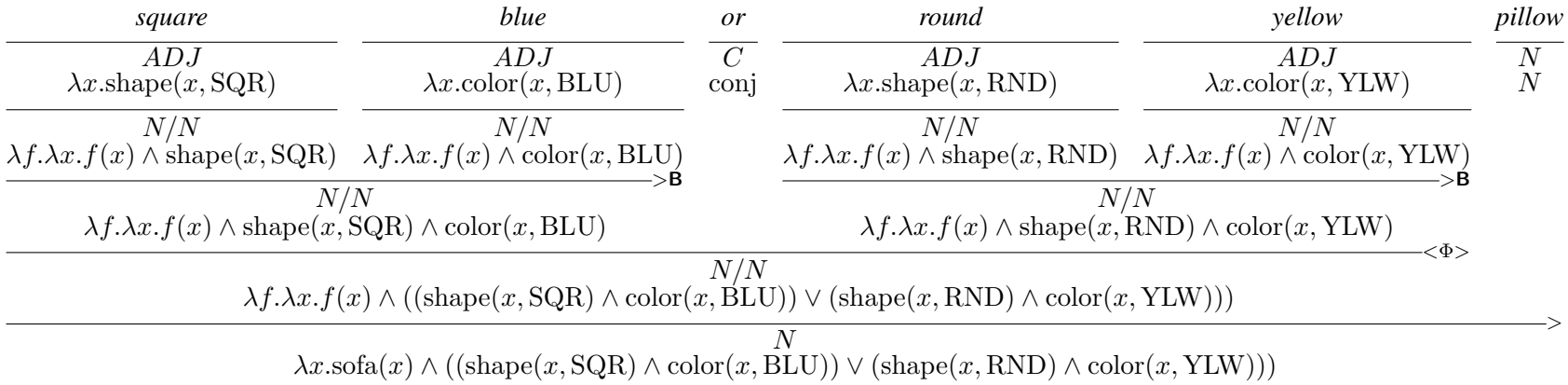


Figure 2.3: Example CCG parse tree for the phrase *square blue or round yellow pillow*.

form:

Adjective:	$ADJ : \lambda x.f(x) \;\Rightarrow\; N/N : \lambda g.\lambda x.g(x) \wedge f(x)$
Preposition:	$PP : \lambda x.f(x) \;\Rightarrow\; N\backslash N : \lambda g.\lambda x.g(x) \wedge f(x)$
Adverbial:	$AP : \lambda x.f(x) \;\Rightarrow\; S\backslash S : \lambda g.\lambda x.g(x) \wedge f(x)$
Topicalized adverbial:	$AP : \lambda x.f(x) \;\Rightarrow\; S/S : \lambda g.\lambda x.g(x) \wedge f(x)$

For example, in Figure 2.2, the category of the span *in the intersection*,

$$PP : \lambda y.\text{intersect}(y, \iota(\lambda x.\text{intersection}(x))) \;\;,$$

is shifted using a unary rule to

$$N\backslash N : \lambda f.\lambda y.f(y) \wedge \text{intersect}(y, \iota(\lambda x.\text{intersection}(x))) \;\;,$$

and is then combined with the category of *chair* using backward application ($<$). Figure 2.3 illustrates how function composition is used to combine adjectives to be coordinated. Both the categories for *square* and *blue* are shifted:

$$\textit{square}: \quad ADJ : \lambda x.\text{shape}(x, \text{SQR}) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge \text{shape}(x, \text{SQR}) \;\;,$$
$$\textit{blue}: \quad ADJ : \lambda x.\text{color}(x, \text{BLU}) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge \text{color}(x, \text{BLU}) \;\;,$$

and then combined using forward composition to:

$$N/N : \lambda f.\lambda x.f(x) \wedge \text{shape}(x, \text{SQR}) \wedge \text{color}(x, \text{BLU}) \;\;.$$

Similarly, *round* and *yellow* are shifted and composed. The coordination operation ($<\Phi>$) coordi-

nates the two composed pairs to:

$$N/N : \lambda f.\lambda x.f(x) \wedge ((\text{shape}(x, \text{SQR}) \wedge \text{color}(x, \text{BLU})) \vee (\text{shape}(x, \text{RND}) \wedge \text{color}(x, \text{YLW}))) \quad .$$

This trinary operation is equivalent to assigning *or* the category:

$$N/N\backslash(N/N)/(N/N) : \lambda f.\lambda g.\lambda h.\lambda x.h(x) \wedge (f(\lambda x.\text{true}, x) \vee g(\lambda x.\text{true}, x)) \quad ,$$

and then using two application operations: forward and backward.

Additionally, in Chapter 7, we use syntactic attributes for singular sg, plural pl and mass nouns nb. For example, a singular noun *sofa* is assigned the category $N_{[sg]} : \lambda x.\text{sofa}(x)$ where the subscript sg indicates it is a singular entity. The plural noun *sofas* will be assigned the category $N_{[pl]} : \lambda x.\text{sofa}(x)$, which only differs in the usage of the pl attribute, which indicates it is a plural entity. We also use syntactic agreement variables, which indicate an agreement constraint between two unspecified syntactic attributes. For example, the adjective *red* will be assigned the category $N_{[x]}/N_{[x]} : \lambda f.\lambda x.f(x) \wedge \text{color}(x, \mathcal{A}(\lambda y.\text{red}(y)))$, where x indicates that once this category consumes its argument, the syntax of the resulting category will be determined accordingly. For example, for the phrase *red sofa*, which is the result of forward application between *red* and *sofa*, the category will be $N_{[sg]} : \lambda x.\text{sofa}(x) \wedge \text{color}(x, \mathcal{A}(\lambda y.\text{red}(y)))$. Equivalently, the category for *red sofas* will be $N_{[pl]} : \lambda x.\text{sofa}(x) \wedge \text{color}(x, \mathcal{A}(\lambda y.\text{red}(y)))$. We limit the grammar to support a single agreement variable at most per category. See Hockenmaier (2003) for a detailed description of CCG syntactic attributes and agreement variables.

Using CCG gives us access to a well studied linguistic formalism. As we discuss in Chapter 3, CCG has been extensively applied, including to various linguistic phenomena and multiple languages, and its computational properties are well studied. Next, we discuss an extension to CCG, with the goal of making it more effective for learning form limited data.

2.3 Factored Lexicons

We adopt a factored representation for CCG lexicons (Kwiatkowski et al., 2011), where entries are dynamically generated by combining *lexemes* and *templates*. A lexeme is a pair that consists of a natural language phrase and a set of logical constants, while the template contains the syntactic and semantic components of a CCG category, abstracting over logical constants. For example, consider the lexical entry *walk* $\vdash S/NP : \lambda x.\lambda a.\text{move}(a) \wedge \text{direction}(a, x)$. Under the factored representation, this entry can be constructed by combining the lexeme $\langle$*walk*, $\{\text{move}, \text{direction}\}\rangle$ and the template $\lambda v_1.\lambda v_2.[S/NP : \lambda x.\lambda a.v_1(a) \wedge v_2(a, x)]$. This representation allows for better generalization over unseen lexical entries at inference time, allowing for pairings of templates and lexemes not seen during training.

2.4 Weighted CCG

Parsing with a CCG requires choosing appropriate lexical entries from an often ambiguous lexicon and the order in which operations are applied. In general, given a CCG, there are many parses for each sentence. To discriminate between competing parses, we use log-linear weighted CCG, inspired by Clark and Curran (2007b).[2]

Let $\mathcal{X}$ be the set of all possible sentences. Let $\mathcal{Y}$ be the space of CCG parse trees and $\mathcal{Z}$ the space of all possible logical forms. Given a sentence $x \in \mathcal{X}$, our goal is to generate a CCG parse $y \in \mathcal{Y}$, which includes a logical form $z \in \mathcal{Z}$ in its root. Let $\textsc{Gen}(x; \Lambda) \subset \mathcal{Y}$ be the set of all possible CCG parses given the sentence x and the lexicon Λ. In $\textsc{Gen}(x; \Lambda)$, multiple parse trees may have the same logical form; let $\mathcal{Y}(z) \subset \textsc{Gen}(x; \Lambda)$ be the subset of such parses with the logical form z at the root. Also, let $\theta \in \mathbb{R}^d$ be a d-dimensional parameter vector. We define the probability of the logical form z as:

$$p(z \mid x; \theta, \Lambda) = \sum_{y \in \mathcal{Y}(z)} p(y \mid x; \theta, \Lambda) \quad . \tag{2.1}$$

[2] In Chapters 4 and 5 we use linear weighted CCG, where the score is not exponentiated, while in Chapters 6 and 7 we use log-linear weighted CCG.

Above, we marginalize out the probabilities of all parse trees with the same logical form z at the root. The probability of a parse tree y is defined as:

$$p(y \mid x; \theta, \Lambda) = \frac{e^{\theta \cdot \phi(x,y)}}{\sum_{y' \in \text{GEN}(x;\Lambda)} e^{\theta \cdot \phi(x,y')}} \quad , \tag{2.2}$$

where $\phi(x, y) \in \mathbb{R}^d$ is a feature vector.

At inference time, given a sentence x, we find the best logical form z^* by solving:

$$z^* = \arg\max_z \ \ p(z \mid x; \theta, \Lambda) \ \ . \tag{2.3}$$

The above $\arg\max$ operation sums over all trees $y \in \mathcal{Y}(z)$, as described in Equation 2.1. We use a CKY chart for this computation. The chart signature in each span is a CCG category. Since exact inference is prohibitively expensive, we follow previous work (Zettlemoyer and Collins, 2005) and perform bottom-up beam search, maintaining only the k-best categories for each span in the chart. The logical form z^* is taken from the k-best categories at the root of the chart. The partition function in Equation 2.2 is approximated by summing the inside scores of all categories at the root.

2.5 Supervised Learning with GENLEX

Previous work (Zettlemoyer and Collins, 2005) introduced a function GENLEX(x, z) to map a sentence x and its logical form z to a large set of potential lexical entries. To generate entries, GENLEX dynamically generates new lexemes from x and z, which are then used to instantiate pre-defined lexical templates. Lexemes are generated by considering all subsets of logical constants from z and pairing them with all n-grams in x. Figure 2.4 shows an example set of generated entries for the sentence *go to the chair* given a small set of templates. All n-grams from the sentence x (2.4b) are combined with all constants from z (2.4c) to generate a set of lexemes (2.4d), which are finally paired with the templates (2.4e) to generate lexical entries (2.4f). The complete learning algorithm then simultaneously selects a small subset of these entries and estimates parameter values θ. In Chapter 4 we present a version of GENLEX that relies on system utterances instead of annotated

(a) **Training sample:** Training sentence x and its labeled logical form z for GENLEX(x, z).

x**:**	*go to the chair*
z**:**	$\lambda a.\text{move}(a) \wedge \text{to}(a, \iota(\lambda x.\text{chair}(x)))$

(b) **N-grams:** All the n-grams in x that can are used to generate new lexemes. In practice, we often cap the maximum n-gram length.

go	*go to*	*go to the*	*go to the chair*
to	*to the*	*to the chair*	
the	*the chair*		
chair			

(c) **Logical constants:** The set of logical constants extracted from z to generate new lexemes. We do not consider $\wedge$ and ι for factoring, and therefore ignore them when generating lexemes.

move	to	chair

(d) **Generated lexemes:** A subset of all the lexemes generated for x and z above. Given four tokens, three constants (move, to and chair) GENLEX will generate 80 lexemes. Each lexeme is a pairing of a subset of constants with a n-gram.

$\langle go, \{\text{move}, \text{to}, \text{chair}\}\rangle$	$\langle to\ the\ chair, \{\text{to}, \text{chair}\}\rangle$	$\langle go\ to\ the, \{\text{move}\}\rangle$
$\langle go, \{\text{move}\}\rangle$	$\langle to, \{\text{to}\}\rangle$	$\langle the\ chair, \{\text{to}\}\rangle$

(e) **Predefined templates:** A small set of predefined lexical templates used in GENLEX(x, z). In practice, we use a few dozen templates, often obtained from annotating a small set of sentences with lexical entries.

$\lambda v_1.[S : \lambda a.v_1(a)]$	$\lambda v_1.[AP/NP : \lambda x.\lambda a.v_1(a, x)]$

(f) **Generated entries:** All the lexical entries generated by pairing the lexemes and templates above. In realistic settings, even for such a short sentence we will generate many more lexical entries.

$go\ to\ the \vdash S : \lambda a.\text{move}(a)$	$go \vdash S : \lambda a.\text{move}(a)$
$to \vdash AP/NP : \lambda x.\lambda a.\text{to}(a, x)$	$the\ chair \vdash AP/NP : \lambda x.\lambda a.\text{to}(a, x)$

Figure 2.4: Lexical entries generated by GENLEX for the sentence x and logical form z given two predefined templates.

sentences. Chapter 5 describes how to generate lexical entries with predefined templates without access to any symbolic supervision. Finally, in Chapter 7 we show how to combine GENLEX with the unification-based approach of Kwiatkowski et al. (2010) to learn new templates.

2.6 Skolem Terms

Generalized Skolem terms for CCG were introduced by Steedman (2011) to simplify derivations and capture complex dependencies with relatively local quantification. We use a simplified version of the theory to represent entities and allow distant references in Chapter 7. Our representation of the indefinite determiner in Chapter 5 is also related. We formally define here our simplified version, and refer the reader to Steedman (2011) for the complete linguistic theory. Let $\mathcal{A}$ be a $\langle\langle e,t\rangle,e\rangle$-typed logical constant. We call $\mathcal{A}$ a generalized Skolem quantifier. Given a $\langle e,t\rangle$-typed logical expression C, the logical form $\mathcal{A}(C)$ is a generalized Skolem term. Since we do not make the necessary distinctions, we use the terms *generalized Skolem term* and *Skolem term* interchangeably. For example, $\mathcal{A}(\lambda x.\text{sofa}(x))$ and $\mathcal{A}(\lambda x.\text{sofa}(x) \wedge \text{located-in}(x, \mathcal{A}(\lambda y.\text{house}(y))))$ are both Skolem terms. Skolem terms may receive id-typed Skolem IDs. For example, in the logical form $\mathcal{A}_1(\lambda x.\text{sofa}(x))$ the number 1 is a Skolem ID. Skolem IDs are globally scoped, i.e., they can be referred from anywhere in the logical form without scoping constraints. To refer to Skolem IDs we define the $\langle id,e\rangle$-typed constant $\mathcal{R}$. For example, in the logical form

$$\begin{aligned}&\mathcal{A}_1(\lambda x.\text{love-01}(x)\\&\qquad\text{ARG0}(x, \mathcal{A}_2(\lambda y.\text{person}(y)\wedge\\&\qquad\qquad\text{name}(y, \text{JOHN})))\wedge\\&\qquad\text{ARG1}(x, \mathcal{R}(2)))\end{aligned} \tag{2.4}$$

the literal $\mathcal{R}(2)$ is a reference to the Skolem term $\mathcal{A}_2(\lambda y.\text{person}(y) \wedge \text{name}(y, \text{JOHN}))$, which has the Skolem ID 2.

We will use skolem terms to express distant dependencies, for example, as introduced by pronouns and other co-referring expressions. This is a relatively limited use of the theory discussed in Steedman (2011), which we hope to further integrate into our learning approach in future work.

Chapter 3

RELATED WORK

This chapter reviews several lines of related work. We start with a short review of work on CCG (Section 3.1) and semantic representations (Section 3.2). We then review work on learning for semantic parsing, the task of recovering the formal meaning representation of natural language sentences. We divide this review according to the type of supervision used. In Section 3.3 we review learning approaches that assume access to sentences paired with linguistic annotation. We describe techniques that assume complete annotation of the entire automated decision process as well as methods that assume partially annotated data (i.e., for latent variable learning). Section 3.4 discusses approaches that rely on demonstrations of desired system behavior. Such supervision often requires no linguistic expert knowledge and is significantly less labor intensive. Section 3.5 overviews other forms of supervisions, including distant supervision and unsupervised learning. Finally, we quickly review the long history of non-learning approaches (Section 3.6).

3.1 Combinatory Categorial Grammar

CCG is a categorial grammar formalism in which both function application and composition are allowed. CCG is a member of an equivalence class of mildly context-sensitive formalisms, including tree adjoining grammar, linear indexed grammar and head grammar (Weir, 1988; Joshi et al., 1990). Steedman (1996, 2000) and Steedman and Baldridge (2003) provide a comprehensive overview of CCG. The formalism has been widely studied and various extensions have been proposed. We list several examples. Baldridge (2002) describes an extension of the category structure for fine-grained lexical control. Hoffman (1995) and Bozsahin (1998) suggested extensions to recover the semantic meaning conveyed by word order in languages with flexible ordering, such as Turkish. Steedman (2011) discussed the syntax and semantics of quantifier-scope, negation, polar-

ity, coordination and pronominal bindings within CCG. While our use of CCG is relatively simple, these extensions and many others can be adopted in future work into our approach, for example, as we study other languages. Section 2.2 describes the subset of the formalism we use.

CCG has been extensively studied for broad-coverage syntactic parsing, where the goal is to recover a parse tree with the semantic representation omitted. Vijay-Shanker and Weir (1990) and Kuhlmann and Satta (2014) described polynomial-time extensions of CKY parsing for CCG recognition, both algorithms have a complexity bound of $\mathcal{O}(n^6)$, where n is the number of tokens in the sentence. Eisner (1996) and Hockenmaier and Bisk (2010) proposed normal-form constraints to avoid spurious ambiguity during parsing, thereby drastically reducing the number of derivations considered. CCGBank, a version of the Penn TreeBank (Marcus et al., 1993) with CCG parse tree annotations, was introduced by Hockenmaier and Steedman (2007). It was created using an automated process followed by manual corrections. The release of CCGBank was followed by extensive work on learning, including with supervised methods (e.g., Hockenmaier, 2003; Clark and Curran, 2007b,a; Lewis and Steedman, 2014), weakly supervised techniques (e.g., Garrette et al., 2015) and unsupervised approaches (e.g., Bisk and Hockenmaier, 2013, 2015; Bisk et al., 2015). Honnibal et al. (2010) proposed to heuristically re-bank CCGBank for improved noun phrase interpretation.

3.2 Semantics

Building formal representations of language meaning has been studied extensively. We briefly review here some of the most relevant work and refer the reader to Maienborn et al. (2011) for a more detailed historical overview of semantics. Montague (1970a,b, 1973) proposed to use lambda calculus (Church, 1932, 1940) to represent the meaning of sentences, where the constituents in the language are mirrored by constituents in the logic and function application is used to combine them. Davidson (1967, 1969) observed that adverbial modifiers present a similar behavior to adjectival ones. Specifically, an arbitrary number of adverbial modifiers can be included in a sentence and their ordering, while not fully unconstrained, is flexible. Therefore, Davidson proposed to introduce event variables and treat adverbials as conjunctive modifiers, where adverbs modify not the

verb itself, but instead an event variable, which is typed by the verb introduced predicate. This approach was further extended by Parsons (1990), who observed that passivization of transitive verbs often makes the subject optional, therefore requiring predicates with different arity than the ones used for active sentences. Parsons proposed a Neo-Davidsonian treatment of thematic roles, where instead of appearing as argument to the verb predicate, they are specified as conjunctive modifiers. Our semantic representation combines ideas from Neo-Davidsonian event semantics and the typed model theory introduced by Carpenter (1997). Our work on representing non-compositional aspects of language meaning (Chapter 7) is related to existing work on Discourse Representation Theory (Kamp and Reyle, 1993; Kamp et al., 2011), where first-order logic statements are embedded within discourse frames that define variable scoping. In contrast, we follow Steedman (2011) and rely on globally scoped variable-like Skolem IDs (Section 2.6) to capture such distant dependencies (Section 7.3.3).

3.3 Learning from Linguistic Annotation for Semantic Parsing

To the best of our knowledge, Miller et al. (1994) presented the first learning approach for mapping sentences to logical forms. In their approach, sentences were mapped to tree structured meaning representations via a learned generative grammar. Training data included sentences paired with fully observed derivation trees and the ATIS (Dahl et al., 1994) corpus was used for training and evaluation. Zelle (1995) and Zelle and Mooney (1996a) proposed to learn deterministic shift-reduce parsers with backtracking to map sentences to Prolog queries. Their approach was originally designed for case-role parsing (Zelle and Mooney, 1993), an early task that combined aspects of dependency parsing and semantic role labeling (Gildea and Jurafsky, 2002), and for treebank syntactic parsing (Zelle and Mooney, 1994). Similar to Miller et al. (1994), learning required sentences paired with fully annotated derivation trees. For evaluation, the GeoQuery corpus (Zelle and Mooney, 1996b) was introduced, which has been used extensively as a benchmark corpus in the field since then. Tang and Mooney (2000, 2001) proposed to use statistical relational learning to build a probabilistic model that allows for statistical parsing for this model. Following a similar model, Thompson and Mooney (2003) were the first to treat derivation decisions as latent, instead

learning from sentences paired with logical forms. In this scenario, learning required inducing the lexicon that maps words to the meaning representation. They treated lexicon induction as a separate learning problem to be solved prior to inducing the parser.

A common and extensively studied approach to semantic parsing is to treat it as a statistical machine translation (SMT) problem, where the source is natural language and the target is a formal language. Training in SMT-based approaches assumes access to pairs of sentence and logical forms. Epstein et al. (1996) were the first to apply SMT techniques to semantic parsing. Their approach relies on using a hierarchy of IBM-style alignment models. Papineni et al. (1997) extended this approach to use a feature-based model to score the alignments, rather than a simple generative model. Motivated by scalability issues observed in the feature-rich approach, Ramaswamy and Kleindienst (2000) proposed to use a two-stage process, where the first model selects which SMT model to use. Wong and Mooney (2006) proposed to use word alignment for lexical learning and a synchronous context-free grammar (CFG) for parsing. While this approach was first proposed to recover variable-free logic representations, it was later extended to recover lambda-calculus meaning representations (Wong and Mooney, 2007). Matuszek et al. (2010) applied this approach to execute navigation instructions. Recently, Andreas et al. (2013) demonstrated the effectiveness of simple SMT tools (Och and Ney, 2003; Koehn et al., 2007) for semantic parsing.

Various other techniques were proposed to induce parsers from sentences paired with annotated logical forms. Ge and Mooney (2005) designed a two-stage process based on syntactic parsing: first, a syntactic parse tree is constructed with semantic labels on non-terminals, and then a compositional function is used to combine the labels into a single logical form. Ge and Mooney (2006) studied global re-ranking for this model. Kate et al. (2005) treated the learning problem as inducing rules that transform the sentences into a logical form, similar to how sentences are gradually "reduced" in CFG. Kate and Mooney (2006) proposed an alternative parsing model where local decisions are made using a support vector machine (SVM) with a string kernel. Lu et al. (2008) presented a generative model with a dynamic programming decoding algorithm. He and Young (2005, 2006) proposed to use a sequence model with a stack to construct hierarchical representations. In their model, words are consumed in order and can be pushed to a stack to be processed

later. This approach is reminiscent of shift-reduce parsing.

Zettlemoyer and Collins (2005) introduced the first approach for CCG induction for semantic parsing. Learning alternated between expanding the CCG lexicon and updating the parameters, where lexicon expansion relied on a set of manually defined templates to hypothesize new lexical entries. Training assumed access to sentences paired with their lambda-calculus meaning representation. The advantage of this approach was in its combination of grammar induction and parameter estimation. This created an interesting interaction between the two: as the model parameters improved new grammatical structures could be explored, which in turn triggered better estimation of the parameters. Zettlemoyer and Collins (2007) extended this approach to handle ungrammatical spoken language by introducing relaxed CCG operations, which allow for combinations that violate directionality constraints. Our grammar induction techniques rely on a similar template-based procedure. However, our approach does not assume access to annotated logical forms.

While using templates has been shown to be effective for CCG induction, it requires to manually define the templates and is challenging to generalize across languages. Kwiatkowski et al. (2010) proposed a unification-based approach that splits existing lexical entries to identify entries that generalize better. The process requires no hand-defined templates and is initialized by creating lexical entries for all the labeled logical forms. It was to be shown effective for existing corpora in English as well as in other languages. A similar unification-based learning technique was later applied to model child language acquisition (Kwiatkowski et al., 2012). With the goal of learning grammars that generalize better, Kwiatkowski et al. (2011) adopted a factored representation of the lexicon. The factored representation relied on, potentially never observed, pairs of lexemes and templates, thereby not requiring to observe every word in every syntactic role during training. Throughout this thesis (except Chapter 4) we adopt this factored representation of the lexicon. Kushman and Barzilay (2013) studied the problem of recovering regular expressions from sentences describing them. Similar to Kwiatkowski et al. (2011), their learning algorithm relied on splitting lexical entries for better generalization. Although these approaches improved performance, they still failed to use existing linguistic resources, which could potentially improve learning with limited data. With this limitation in mind, Wang et al. (2014) proposed to use Wik-

tionary[1] to augment the learning algorithm with existing linguistic knowledge. This approach has been shown to induce compact and linguistically motivated lexicons and is currently the state of the art for the ATIS corpus. In Chapter 7 we present a lexical induction procedure that combines unification-based splitting and templates with CCGBank syntactic parsing to learn new lexical entries for broad-coverage semantic parsing.

The recently released AMR meaning bank (Banarescu et al., 2013) provided a corpus for the development of broad-coverage semantic parsing. Flanigan et al. (2014) proposed a graph-based technique with Lagrangian relaxation to map sentence to AMRs. Wang et al. (2015) described an approach to transform dependency structures to AMRs. Both approaches rely on the heuristic alignment technique of Flanigan et al. (2014) to learn the mapping of words to their meaning. Pourdamghani et al. (2014) proposed to use machine translation techniques to recover such alignments. However, this approach still relies on surface form heuristics to initialize the process. In Chapter 7 we present an approach for CCG induction to map sentences to AMRs.

While the majority of work on semantic parsing considers sentences in isolation, using linguistic annotation to recover context-dependent meaning was also studied. Miller et al. (1996) presented the first approach to learn to correctly interpret sentences within conversations, albeit with a simpler compositional structure than their prior work (Miller et al., 1994). They presented a three-stage process. First, a simplified syntactic parse tree is recovered with a generative process similar to Miller et al. (1994). Frames and their slot values are then extracted from the parse. Finally, the frames are transformed to capture discourse-level information. Training assumed access to fully observed derivations. The approach was developed and evaluated on the ATIS corpus (Dahl et al., 1994), which has supported extensive followup work on learning to map sentences to shallow meaning representations for language understanding in dialog system. Tur et al. (2010) surveys the approaches applied to the task along with existing challenges. Zettlemoyer and Collins (2009) proposed a model to recover the context-dependent logical form representation of sentences within conversations. They proposed a two-stage process. First, a context-independent logical form is

[1] https://www.wiktionary.org/

recovered. Second, it is transformed to include contextual information. The work we present in Chapters 5 and 6 similarly considers contextual cues when recovering the meaning of sentences. However, we consider situated cues rather than linguistic discourse cues and learn without annotated logical forms.

3.4 Learning from Demonstrations for Semantic Parsing

While relying on sentences paired with logical forms reduced the need to annotate every derivation decision, partially facilitating the proliferation of methods, it still relied on a labor-intensive process and linguistic expertise to create the data. In contrast to logical form annotation, labeling the desired system response requires no such expertise and is significantly less labor intensive. Recently, such easily obtained supervision was studied for various tasks. Clarke et al. (2010) proposed to learn from sentences paired with the desired system response in the GeoQuery domain (Zelle and Mooney, 1996b). They formulated semantic parsing as constrained optimization and designed a learning algorithm that assumes access to sentences paired with the set of returned database entries, instead of the formal representation question meaning. This type of supervision was also used to learn to recover Dependency-based Compositional Semantic (DCS) representations for GeoQuery (Liang et al., 2011). This transition from supervised learning was quickly followed by increased interest in question-answering for large knowledgebases with a specific focus on FreeBase (Bollacker et al., 2008). Berant et al. (2013) used Google Suggest and crowdsourcing to collect a set of questions and their answers from FreeBase. They used λ-DCS, a variant of DCS, to learn to map sentences to logical forms that are then executed on FreeBase. Berant and Liang (2014) suggested to paraphrase the input question to bridge the lexical gap between the question and the knowledgebase. Recently, the WikiTableQuestions corpus (Pasupat and Liang, 2015) was proposed in response to the relatively simplistic compositional structure in WebQuestions.

CCG algorithms were also developed for learning from non-linguistic forms of supervision. Kwiatkowski et al. (2013) proposed to use question-answer pairs to induce the parameters for a two-stage model. In the first stage, the question is mapped to a domain-independent underspecified logical form using a manually designed grammar, and then the structure of the logical form

is modified to match the structure of the knowledgebase. The advantage of this approach is in its ability to recover complex compositional phenomena while automatically bridging the gap between the knowledgebase and the logical form. This is in contrast to most existing work, where it is necessary to manually design the space of logical forms and performance varies greatly based on the quality of this design. Choi et al. (2015) identified the potential for extracting information by recovering the compositional structure of noun phrases and proposed an extension of Kwiatkowski et al. (2013) for this task. The algorithms we present in Chapters 5 and 6 assume access to demonstrations of desired system behavior.

Learning from the desired system behavior naturally extends to instructional language. Initial approaches recovered shallow representations of language meaning. Branavan et al. (2009) and Vogel and Jurafsky (2010) proposed to cast language understanding as a shallow tagging problem and designed reinforcement learning (RL) algorithms to estimate its parameters. Branavan et al. (2010) extended their RL approach to model high-level instructions. The learning algorithm we present in Chapter 5 is inspired by their RL approach and our approach to implicit actions corresponds to high-level instructions, as observed in their work. Wei et al. (2009) and Kollar et al. (2010) also used shallow linguistic representations for instructions. Tellex et al. (2011) used a graphical model semantic representation to learn from instructions paired with demonstrations. In contrast, in Chapter 5, we model significantly more complex linguistic phenomena than these approaches, as required for the navigation domain. The navigation corpus we use in Chapter 5 was collected by MacMahon et al. (2006) and was extensively used to learn to execute instructions given system demonstrations (Chen and Mooney, 2011; Chen, 2012; Kim and Mooney, 2012, 2013). Goldwasser and Roth (2011) learned a semantic parser for game instructions from valid and invalid game states paired with instructions.

While most work on semantic parsing assumes a fixed logical ontology, several approaches were proposed to expand this ontology by learning new symbols and their underlying grounded meaning. Matuszek et al. (2012) studied the problem of learning new logical constants paired with vision classifiers to identify previously unknown properties of seen objects. This was done within the task of resolving referring expressions, where the system input is a natural language

expression along with a world state, and the system is required to identify the subset of objects in the world the expression refers to. Training assumed access to triplets of sentence, world state and referred objects. Their system was initialized using the approach of Kwiatkowski et al. (2011) and then alternated between updating its parsing parameters and learning new classifiers for color and shape. The new classifiers were assigned to newly created logical constants to represent previously unknown properties of objects. This approach was extended by Krishnamurthy and Kollar (2013) to learn new binary relations. Krishnamurthy and Mitchell (2015) proposed to use universal schema (Riedel et al., 2013) to learn new relations for semantic parsing for information extraction.

3.5 Other Forms of Supervision for Semantic Parsing

Integrating semantic parsers into complete natural language systems created interesting opportunities for unsupervised learning (Goldwasser et al., 2011; Poon, 2013). Instead of relying on annotation, learning in these approaches relies on a mixture of heuristics, surface-form similarity between words and their formal meaning representation and system response, for example, to reject invalid interpretations. Cai and Yates (2013a) proposed heuristics to use web data to expand a learned CCG lexicon for better coverage for FreeBase questions-answering, where the number of potential predicates is too large to be completely observed during training.

Distant supervision techniques were also shown to be effective for learning semantic parsers. Cai and Yates (2013b) developed a retrieval engine to obtain distantly supervised sentences paired with logical forms to train a semantic parser for FreeBase question-answering. Krishnamurthy and Mitchell (2012) studied the application semantic parsing to information extraction, where the goal is to populate a knowledgebase with new facts, rather than respond to queries. They followed the architecture of Hoffmann et al. (2011) and trained a semantic parser instead of a sentence level classifier to extract relational tuples. To train the model, following Hoffmann et al. (2011) they use distant supervision from the knowledgebase, which they augment with syntactic supervision from a CCGBank syntactic parser. Krishnamurthy and Mitchell (2014) used such supervision in conjunction with CCGBank to jointly train both a semantic parser and a CCGBank syntactic parser.

3.6 Non-learning Approaches for Semantic Parsing

Hand-engineered approaches, which do not rely on machine learning, have been extensively studied. A comprehensive survey of this work is out of the scope of this thesis. We therefore provide a short overview and refer the reader to Androutsopoulos et al. (1995) for a more complete survey of topics and approaches in earlier work. Travel planning, mostly using the ATIS corpus (Dahl et al., 1994), has been a common domain for developing and evaluating techniques for building natural language interfaces (e.g., Woods, 1968; Carbonell and Hayes, 1983; Ward and Issar, 1994; Levin et al., 2000; Popescu et al., 2004). In Chapter 4 we learn a semantic parser from similar travel planning conversations without any annotation effort. Bos (2008) proposed a heuristic approach to convert CCGBank parse trees to Discourse Representation Structures (Kamp and Reyle, 1993). In contrast, the work we present in Chapter 7 learns the grammar, attempts to recover representations grounded in broad-coverage ontological resources and was evaluated against existing corpora. Instructional language, which we discuss in Chapter 5, has been extensively studied in non-learning settings (e.g., Winograd (1972); Di Eugenio and White (1992); Webber et al. (1995); Bugmann et al. (2004); MacMahon et al. (2006) and Dzifcak et al. (2009)).

Chapter 4

BOOTSTRAPPING SEMANTIC PARSERS FROM CONVERSATIONS

This chapter addresses the problem of learning a semantic parser from conversational cues. We motivate our approach by identifying opportunities for learning in interactive dialog systems, where systems often fail to understand the user but try to recover and continue the interaction. Relying on such failures and recovery attempts, we describe a learning approach to induce a CCG, including both a lexicon and parsing parameters. Except automatically recorded interactions of non-experts with the system, our approach requires no expert annotation or any labeling effort. When applied to the task of learning a semantic parser for the travel domain, our approach demonstrates near supervised-learning performance on conversations from two dialog systems. This chapter is based on work originally described in Artzi and Zettlemoyer (2011).

4.1 Introduction

Conversational interactions provide significant opportunities for autonomous learning. A well-defined goal allows a system to engage in remediations when confused, such as asking for clarification, rewording, or additional explanation. The user's response to such requests provides a strong, if often indirect, signal that can be used to learn to avoid the original confusion in future conversations. In this chapter, we show how to use this type of conversational feedback to learn to better recover the meaning of user utterances, by inducing semantic parsers from unannotated conversational logs. We believe that this style of learning will contribute to the long term goal of building self-improving dialog systems that continually learn from their mistakes, with little or no human intervention.

Many dialog systems use a semantic parser to analyze user utterances (e.g., Allen et al., 2007; Litman et al., 2009; Young et al., 2010). For example, in a flight booking system, the sentence

Sentence: *I want to go to Seattle on Friday*

LF: $\lambda x.\text{to}(x, \text{SEA}) \wedge \text{date}(x, \text{FRI})$

might be mapped to the logical form (LF) meaning representation above, a lambda-calculus expression defining the set of flights that match the user's desired constraints. This LF is a representation of the semantic content that comes from the sentence, and would be input to a context-dependent understanding component in a full dialog system, for example to find the date that the symbol FRI refers to.

To induce semantic parsers from interactions, we consider user statements in conversational logs and model their meaning with latent variables. We demonstrate that it is often possible to use the dialog that follows a statement (including remediations such as clarifications, simplifications, etc.) to learn the meaning of the original sentence. For example, consider the first user utterance in Figure 4.1, where the system failed to understand the user's request. To complete the task, the system must use a remediation strategy. Here, it takes the initiative by asking for and confirming each flight constraint in turn. This strategy produces an unnatural conversation but provides supervision for learning the meaning of the original utterance. We can easily record representations of the meanings the system intended to convey at each step, as seen in Figure 4.1, and use this indirect supervision for learning.

Learning from this weak signal is challenging. In any specific conversation, the system's remediations can fail to recover aspects of the original user meaning and can introduce spurious constraints, for example when users change their goals mid conversation. To learn effectively, the model must accumulate evidence from many interactions to best recover the meaning of each specific sentence.

We will learn semantic parsers defined by a probabilistic CCG, which include both a lexicon and a weighted linear model for parse selection. The lexicon specifies the meanings of individual words and phrases, while the parameters of a parsing model define how to best combine word- and phrase-level meanings to analyze complete sentences. To learn without labeled meaning representations, we make use of a variant of the loss-sensitive Perceptron algorithm (Singh-Miller and

SYSTEM: *how can i help you?* (OPEN_TASK)
USER: *i would like to fly from atlanta georgia to london england on september twenty fourth in the early evening i would like to return on october first departing from london in the late morning*
SYSTEM: *leaving what city?* (ASK:$\lambda x.\text{from}(fl, x)$)
USER: *atlanta georgia*
SYSTEM: *leaving atlanta.* (CONFIRM:$\text{from}(fl, \text{ATL})$) *going to which city?* (ASK:$\lambda x.\text{to}(fl, x)$)
USER: london
SYSTEM: *arriving in london england.* (CONFIRM:$\text{to}(fl, \text{LON})$) *what date would you like to depart atlanta?* (ASK:$\lambda x.\text{from}(fl, \text{ATL}) \wedge \text{depart-date}(fl, x)$)
USER: *september twenty fourth in the early evening*
[conversation continues]

Figure 4.1: Conversational excerpt from a DARPA Communicator travel-planning dialog. Each system statement is labeled with representations of its speech act and logical meaning, in parentheses. User utterances have no labels. Conversations of this type provide the training data to learn semantic parsers for user utterances.

Collins, 2007). We define loss functions to provide a rough measure of (1) how well a candidate meaning for a utterance matches the conversation that follows it, and (2) how well the candidate matches our expectations about the types of things that are often said in the dialog's domain. These notions of loss drive not only the parameter estimation but also the grammar induction process that constructs the CCG lexicon. Experiments on conversation logs from the DARPA Communicator corpus (Walker et al., 2002) demonstrate the feasibility of our approach.

4.2 Problem

Our goal is to learn a function that maps a sentence x to a lambda-calculus expression z. We assume access to logs of conversations with automatically generated annotation of system utterance meanings, but no explicit labeling of each user utterance meaning.

We define a conversation $\mathcal{C} = (\vec{U}, O)$ to be a sequence of utterances $\vec{U} = [u_0, \ldots, u_m]$ and a set of conversational objects O. An object $o \in O$ is an entity that is being discussed, for example there would be a unique object for each flight leg discussed in a travel planning conversation. Each utter-

ance $u_i = (s, x, a, z)$ represents the speaker $s \in \{\text{USER}, \text{SYSTEM}\}$ producing the natural language statement x which asserts a speech act $a \in \{\text{ASK}, \text{CONFIRM}, \ldots\}$ with meaning representation z. For example, for the second system utterance in Figure 4.1 the question x =*Leaving what city?* is an a=ASK speech act with lambda-calculus meaning $z = \lambda x.\text{from}(fl, x)$. This meaning represents the fact that the system asked for the departure city for the conversational object $o = fl$ representing the flight leg that is currently being discussed. We will learn from conversations where the speech acts a and logical forms z for user utterances are unlabeled. Such data can be generated by recording interactions, along with each system's internal representation of its own utterances.

Finally, since we will be analyzing sentences at a specific point in a complete conversation, we define our training data as a set $D = \{(j_i, \mathcal{C}_i) : i = 1 \ldots N\}$. Each pair is a conversation $\mathcal{C}_i$ and the index j_i of the user utterance x in $\mathcal{C}_i$ whose meaning we will attempt to learn to recover. In general, the same conversation $\mathcal{C}$ can be used in multiple examples, each with a different sentence index. Section 4.6 provides the details of how the data was gathered for our experiments.

4.3 Overview of Approach

We will present an algorithm for learning a weighted CCG parser, as defined in Section 2.2, that can be used to map sentences to logical forms. The approach induces a lexicon to represent the meanings of words and phrases while also estimating the parameters of a weighted linear model for selecting the best parse given the lexicon.

Learning As defined in Section 4.2, the algorithm takes a set of N training examples, $\{(j_i, \mathcal{C}_i) : i = 1, \ldots, n\}$. For each example, our goal is to learn to parse the user utterance x at position j_i in $\mathcal{C}_i$. The training data contains no direct evidence about the logical form z that should be paired with x, or the CCG analysis that would be used to construct z. We model all of these choices as latent variables.

To learn effectively in this complex, latent space, we introduce a loss function $\mathcal{L}(z, j, \mathcal{C}) \in \mathbb{R}$ that measures how well a logical form z models the meaning for the user utterance at position j in $\mathcal{C}$. In Section 4.4, we will present the details of the loss we use, which is designed to be

sensitive to remediations in $\mathcal{C}$ (system requests for clarification, etc.) but also be robust to the fact that conversations often do not uniquely determine which z should be selected, for example when the user prematurely ends the discussion. Then, in Section 4.5, we present an approach for incorporating this loss function into a complete algorithm that induces a CCG lexicon and estimates the parameters of the parsing model.

This learning setup focuses on a subproblem in dialog; semantic interpretation. We do not yet learn to recover user speech acts or integrate the logical form into the context of the conversation. These are important areas for future work.

Evaluation We will evaluate performance on a test set $\{(x_i, z_i) : i = 1, \ldots, M\}$ of M sentences x_i that have been explicitly labeled with logical forms z_i. This data will allow us to directly evaluate the quality of the learned model. Each sentence is analyzed with the learned model alone; the loss function and any conversational context are not used during evaluation. Parsers that perform well in this setting will be strong candidates for inclusion in a more complete dialog system, as motivated in Section 4.1.

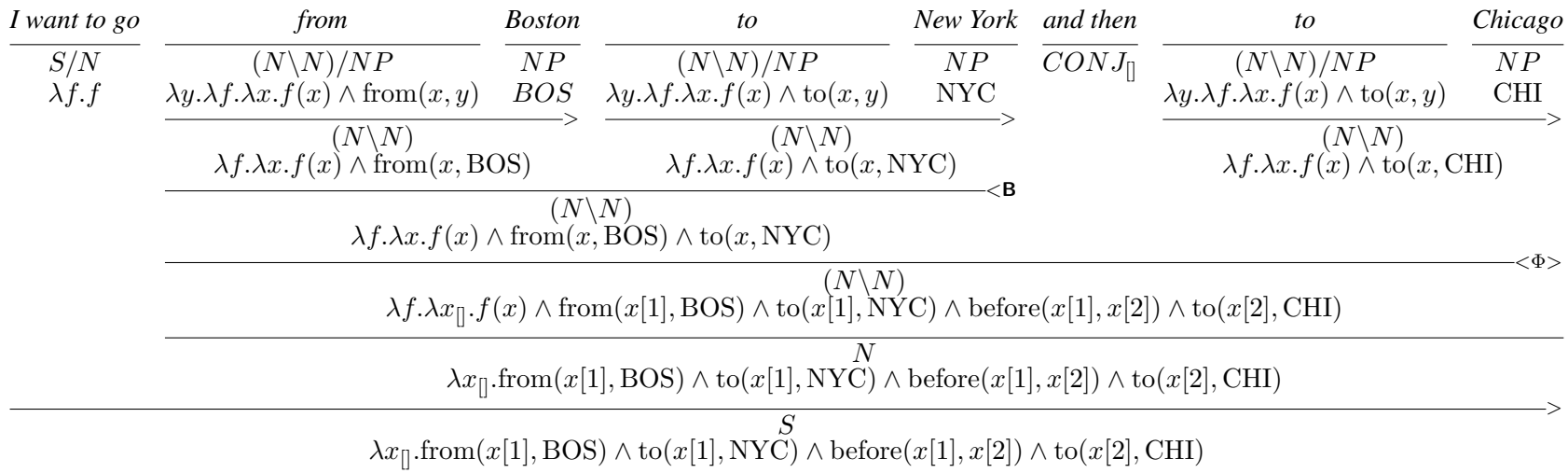


Figure 4.2: An example CCG parse. This parse shows the construction of a logical form with an array-typed variable $x_{[]}$ that specifies a list of flight legs, indexed by $x[1]$ and $x[2]$. The top-most parse steps introduce lexical items while the lower ones create new nonterminals according the CCG combinators (>, <, etc.), see Steedman (2000) for details.

4.4 Measuring Loss

In Section 4.5, we will present a loss-sensitive learning algorithm that models the meaning of user utterances as latent variables to be estimated from conversational interactions.

We first introduce a loss function to measure the quality of potential meaning representations. This loss function $\mathcal{L}(z, j, \mathcal{C}) \in \mathbb{R}$ indicates how well a logical expression z represents the meaning of the j-th user utterance in conversation $\mathcal{C}$. For example, consider the first user utterance ($j = 2$) in Figure 4.3, which is a request for a return trip from Boston to New York. We would like to assign the lowest loss to the meaning representation (d) in Figure 4.3 that correctly encodes all of the stated constraints.

We make use of a loss function with two parts: $\mathcal{L}(z, j, \mathcal{C}) = \mathcal{L}_c(z, j, \mathcal{C}) + \mathcal{L}_d(z)$. The conversation loss $\mathcal{L}_c$ (defined in Section 4.4.1) measures how well the candidate meaning representation fits the conversation, for example incorporating information recovered through conversational remediations as motivated in Section 4.1. The domain loss $\mathcal{L}_d$ (described in Section 4.4.2) measures how well a logical form z matches domain expectations, such as the fact that flights can only have a single origin. These functions guide the types of meaning representations we expect to see, but in many cases will fail to specify a unique best option, for example in conversations where the user prematurely terminates the interaction. In Section 4.5, we will present a complete, loss-driven learning algorithm that is robust to these types of ambiguities while inducing a weighted CCG parser from conversations.

4.4.1 Conversation Loss

We will use a conversation loss function $\mathcal{L}_c(z, j, \mathcal{C})$ that provides a rough indication of how well the logical expression z represents a potential meaning for the user utterance at position j in $\mathcal{C}$. For example, the first user utterance ($j = 2$) in Figure 4.3 is a request for a return trip from Boston to New York where the user has explicitly mentioned both legs. The figure also shows five options (a-e) for the logical form z. We want to assign the lowest loss to option (d), which includes all of the stated constraints.

Example Conversation:

(1) SYSTEM: *how can i help you?* (OPEN_TASK)

(2) USER: *i would like to fly from boston arriving to new york and back from new york to boston*

(3) SYSTEM: *leaving boston* (CONFIRM:$\text{from}(fl1, \text{BOS})$) *on what date?* (ASK:$\lambda x.\text{depart-date}(fl1, x)$)

(4) USER: *september twenty four*

(5) SYSTEM: *leaving boston* (CONFIRM:$\text{from}(fl1, \text{BOS})$) *on september twenty four* (CONFIRM:$\text{depart-date}(fl1, \text{sept\_24})$) *to new york* (CONFIRM:$\text{to}(fl1, \text{NYC})$) *is new york your final destination?* (ASK:$\neg\exists x.\text{fr}(x, \text{NYC})$)

(6) USER: *yes i want to return to boston*

(7) SYSTEM: *sorry, i didn't understand.* (APOLOGY) *where are you flying to from new york?* (ASK:$\lambda x.\text{fr}(fl2, \text{NYC}) \wedge \text{to}(fl2, x)$)

[conversation ends]

Candidate Logical Expressions for Utterance #2:

(a) $\lambda x.\text{to}(x, \text{BOS}) \wedge \text{from}(x, \text{NYC})$

(b) $\lambda x.\text{from}(x, \text{BOS}) \wedge \text{to}(x, \text{NYC})$

(c) $\lambda x.\text{to}(x, \text{BOS}) \wedge \text{to}(x, \text{NYC})$

(d) $\lambda x[].\text{from}(x[1], \text{BOS}) \wedge \text{to}(x[1], \text{NYC}) \wedge \text{before}(x[1], x[2]) \wedge \text{return}(x[2])$
$\wedge \text{from}(x[2], \text{NYC}) \wedge \text{to}(x[2], \text{BOS}))$

(e) $\lambda x[].\text{from}(x[1], \text{BOS}) \wedge \text{to}(x[1], \text{NYC}) \wedge \text{before}(x[1], x[2]) \wedge \text{return}(x[2])$
$\wedge \text{from}(x[2], \text{BOS}) \wedge \text{to}(x[2], \text{NYC})$

Figure 4.3: Conversation reflecting an interaction as seen in the DARPA Communicator travel-planning dialogs.

The loss is computed in four steps for a user utterance x at position j by (1) selecting a subset of system utterances in the conversation $\mathcal{C}$, (2) extracting and computing loss for semantic content from selected system utterances, (3) aligning the subexpressions in z to the extracted semantic content, and (4) computing the minimal loss value from the best alignment. In Figure 4.3, the loss for the candidate logical forms is computed by considering the segment of system utterances up until the conversation end. Within this segment, the matching for expression (d) involves mapping the origin and departure constraints for the first leg (BOS - NYC) onto the earlier system confirmations while also aligning the ones for the second leg to system utterances later in the selected portion of the conversation. Finally, the overall score depends on the quality of the alignment, for example how many of the constraints match to confirmations. This section presents the full approach.

Segmentation For a user utterance at position j, we select all system utterances from $j-1$ until the system believes it has completed the current subtask, as indicated by a reset action or final offer. We call this selected segment $\bar{\mathcal{C}}$. In Figure 4.3, $\bar{\mathcal{C}}$ ends with a reset, but in a successful interaction it would have ended with the offer of a specific flight.

Extracting Properties A property is a predicate-entity-value triplet, where the entity can be a variable from z or a conversational object. For example, $\langle \mathrm{from}, fl, \mathrm{BOS} \rangle$ is a property where fl is a object from $\bar{\mathcal{C}}$ and $\langle \mathrm{from}, x, \mathrm{BOS} \rangle$ is a property from $z = \lambda x.\mathrm{from}(x, \mathrm{BOS})$. We define $P_{\bar{\mathcal{C}}}$ to be the set of properties from logical forms for system utterances in $\bar{\mathcal{C}}$. Similarly, we define P_z to be the set of properties in z.

Scoring System Properties For each system property $p \in P_{\bar{\mathcal{C}}}$ we compute its position value $\textsc{Pos}(p)$, which is a normalized weighted average over all the positions where it appears in a logical form. For each mention the weight is obtained from its speech act. For example, properties that are explicitly confirmed contribute more to the average than those that were merely offered to the user in a select statement.

We use $\textsc{Pos}(p)$ to compute a loss $\textsc{Loss}(p)$ for each system property $p \in P_{\bar{\mathcal{C}}}$. First, we define

$P^e_{\bar{\mathcal{C}}}$ to be all properties in $P_{\bar{\mathcal{C}}}$ with entity e. For entity e and position d, we define the entity-normalization function:

$$n_e(d) = \frac{d - \min_{p \in P^e_{\bar{\mathcal{C}}}} \textsc{Pos}(p)}{\max_{p \in P^e_{\bar{\mathcal{C}}}} \textsc{Pos}(p) - \min_{p \in P^e_{\bar{\mathcal{C}}}} \textsc{Pos}(p)} \quad .$$

For a given property $p \in P_{\bar{\mathcal{C}}}$ with an entity e we compute the loss value:

$$\textsc{Loss}(p) = n_e^{-1}(1 - n_e(\textsc{Pos}(p))) - 1 \quad .$$

Where n_e^{-1} is the inverse of n_e. This loss value is designed to, first, provide less loss for later properties so that it, for example, favors the last property in a series of statements that finally resolves a confusion in the conversation. Second, the loss value is lower for objects mentioned closer to the user utterance x, thereby preferring objects discussed sooner.

Matching Properties An alignment $\mathcal{A}$ maps variables in z to conversational objects in $\bar{\mathcal{C}}$, for example the flight legs $fl1$ and $fl2$ being discussed in Figure 4.3. We will use alignments to match properties of z and $\bar{\mathcal{C}}$. To do this we extend the alignment function $\mathcal{A}$ to apply to properties, for example $\mathcal{A}(\langle \text{from}, x, \text{BOS} \rangle) = \langle \text{from}, \mathcal{A}(x), \text{BOS} \rangle$.

Scoring Alignments Finally, we compute the conversation loss $\mathcal{L}_c(z, j, \mathcal{C})$ as follows:

$$\mathcal{L}_c(z, j, \mathcal{C}) = \min_{\mathcal{A}} \sum_{p_u \in P_z} \sum_{p_s \in P_{\bar{\mathcal{C}}}} s(\mathcal{A}(p_u), p_s) \quad .$$

The function $s(\mathcal{A}(p_u), p_s) \in \mathbb{R}$ computes the compatibility of the two input properties. It is zero if $\mathcal{A}(p_u) \neq p_s$. Otherwise, it returns $\textsc{Loss}(p_s)$.

We approximate the $\min$ computation in $\mathcal{L}_c$ over alignments $\mathcal{A}$ as follows. For a logical form z at position j, we align the outer-most variable to the conversational object in $\bar{\mathcal{C}}$ that is being discussed at j. The remaining variables are aligned greedily to minimize the loss, by selecting a single conversational object for each in turn.

Finally, for each aligned variable, we increase the loss by one for each unmatched property from P_z. This increases the loss of logical forms that include spurious information. However, since a conversation might stop prematurely and therefore won't discuss the entire user request, we only increase the loss for variables that are already aligned. For this purpose, we define an aligned variable to be one that has at least one property matched successfully.

4.4.2 Domain Loss

We also make use of a domain loss function $\mathcal{L}_d(z) \in \mathbb{R}$. The function takes a logical form z and returns the number of violations there are in z to a set of constraints on logical forms that occur commonly in the dialog domain. For example, in a travel domain, a violation might occur if a flight leg has two different destination cities. The set of possible violations must be specified for each dialog system, but can often be compiled from existing resources, such as a database of valid flight ticketing options.

In our experiments, we will use a set of eight simple constraints to check for violations in flight itineraries, which can have multiple legs. These include, for example, checking that the legs have unique origins and destinations that match across the entire itinerary. For example, in Figure 4.3 the logical forms (a), (b) and (d) will have no violations; they describe valid flights. Example (c) has a single violation: a flight has two origins. Example (e) violates a more complex constraint: the second flight's origin is different from the first flight's destination.

4.5 Learning

Algorithm 1 presents the complete learning algorithm. We assume access to training examples, $\{(j_i, \mathcal{C}_i) : i = 1, \ldots, N\}$, where each example includes the index j_i of a sentence x_i in the conversation $\mathcal{C}_i$. The algorithm learns a weighted CCG parser, described in Chapter 2, including both a lexicon Λ and parameters θ. The approach is online, considering each example in turn and performing two steps: (1) expanding the lexicon and (2) updating the parameters.

Algorithm 1 Loss-driven learning algorithm.

Input: Training set $\{(j_i, \mathcal{C}_i) : i = 1 \dots N\}$ where each example includes the index j_i of a sentence x_i in the conversation $\mathcal{C}_i$. Initial lexicon Λ_0. Number of iterations T. Margin γ. Beam size k for lexicon generation. Loss function $\mathcal{L}(x, j, \mathcal{C})$, as described in Section 4.4.

Definitions: $\textsc{GenLex}(x, \mathcal{C})$ takes as input a sentence and a conversation and returns a set of lexical items as described in Section 4.5. $\textsc{Gen}(x; \Lambda)$ is the set of all possible CCG parses for x given the lexicon Λ. Let $\textsc{GenMax}(x, \Lambda, \theta)$ be the set of max-scoring parses given a sentence x, lexicon Λ and parameters θ. $\text{LF}(y)$ returns the logical form z at the root of the parse tree y. Let $\phi_i(y)$ be shorthand for the feature function $\phi(x_i, y)$ defined in Section 2.4. Define $\textsc{Lex}(y)$ to be the set of lexical entries used in parse y. Finally, let $\text{MIN}\mathcal{L}_\text{i}(\text{Y})$ be $\{y \mid \forall y' \in Y, \mathcal{L}(\text{LF}(y), j_i, \mathcal{C}_i) \leq \mathcal{L}(\text{LF}(y'), j_i, \mathcal{C}_i)\}$, the set of minimal loss parses in Y.

Output: Lexicon Λ and model parameters θ.

$\theta \leftarrow \bar{0}$, $\Lambda \leftarrow \Lambda_0$
for t = 1 … T, i = 1 … N **do**
  » **Step 1: Lexical Induction**
  $\lambda \leftarrow \Lambda \cup \textsc{GenLex}(x_i, \mathcal{C}_i)$
  » Get the max-scoring parses.
  $Y \leftarrow \textsc{GenMax}(x_i, \lambda, \theta)$
  » Select lexical entries from the lowest loss parses and update the lexicon.
  $\lambda_i \leftarrow \bigcup_{y \in \text{MIN}\mathcal{L}_\text{i}(\text{Y})} \{l \mid l \in \textsc{Lex}(y)\}$
  $\Lambda \leftarrow \Lambda \cup \lambda_i$
  » **Step 2: Parameter Update**
  » Distinguish between all minimal loss parses (good) and all other (bad).
  $G_i \leftarrow \text{MIN}\mathcal{L}_\text{i}(\textsc{Gen}(\text{x}_\text{i}, \Lambda, \theta))$
  Let $\mathcal{L}_{min}$ be the minimal loss.
  $B_i \leftarrow \textsc{Gen}(x_i, \Lambda, \theta) - G_i$
  » Define the relative loss function.
  $\Delta_i(y) = \mathcal{L}(y, \mathcal{C}_i) - \mathcal{L}_{min}$
  » Construct sets of margin violating good and bad parses.
  $R_i \leftarrow \{r \mid r \in G_i \ \wedge \exists y' \in B_i \text{ s.t. } \langle \theta, \phi_i(r) - \phi_i(y') \rangle < \gamma \Delta_i(r)\}$
  $E_i \leftarrow \{e \mid e \in B_i \ \wedge \exists y' \in G_i \text{ s.t. } \langle \theta, \phi_i(y') - \phi_i(e) \rangle < \gamma \Delta_i(e)\}$
  » Apply the additive update.
  $\theta \leftarrow \theta + \sum_{r \in R_i} \frac{1}{|R_i|} \phi_i(r) - \sum_{e \in E_i} \frac{1}{|E_i|} \phi_i(e)$
return Λ and θ

Step 1: Lexical Induction We introduce new lexical items by selecting candidates from the function GENLEX, following previous work (Zettlemoyer and Collins, 2005, 2007) as reviewed in Section 2.5. However, we face the new challenge that there is no labeled logical-form meaning z.

Instead, let $Z_{\bar{\mathcal{C}}}$ be set of all logical forms that appear in system utterances in the relevant segment in $\mathcal{C}$. We will now define the conversational lexicon set:

$$\textsc{GenLex}(x, \mathcal{C}) = \bigcup_{z \in \mathcal{Z}_{\bar{\mathcal{C}}}} \textsc{GenLex}(x, z) \ .$$

We use logical forms from system utterances to guess possible CCG categories for analyzing the user utterance. This approach will overgeneralize, when the system talks about things that are unrelated to what the user said, and will also often be incomplete, for example when the system does not repeat parts of the original content. However, it provides a way of guessing lexical items that can be combined with previously learned ones, which can fill in any gaps and help select the best analysis.

In line 4 in Algorithm 1, we use GENLEX to temporarily create a large set of potential categories based on the conversation. In lines 6-9, we select a small subset of these entries to add to the current lexicon Λ: we find the max-scoring parses under the model, re-rank them according to loss, find the lexical items used in the best parses, and add them to Λ. This approach favors lexical items that are used in high-scoring but low-loss analyses, as computed given the current model.

Step 2: Parameter Update Given the loss function $\mathcal{L}(x, i, \mathcal{C})$, we use a variant of a loss-sensitive perceptron to update the parameters (Singh-Miller and Collins, 2007). In lines 12-16, for example i, we compute the relative loss function Δ_i that scales with the loss achieved by the best and worst possible parses under the model. In contrast to previous work, we do not only compute the loss over a fixed n-best list of possible outputs, but instead use the current model score to recompute the options at each update. Then, we create the set of least loss analyses R_i and higher-loss candidates E_i whose models scores are not separated by at least $\gamma\Delta_i$, where γ is a margin scale constant (lines 18-19). The final update (line 21) is additive and increases the parameters for features indicative of analyses with less loss while down weighting those for parses that were not sufficiently separated.

	Lucent	BBN
# Conversations	214	162
Total # of utterances	11,974	12,579
Average utterances per conversation	55.95	77.65
Average tokens per user utterance	3.24	2.39
Total # of training utterances	208	67
Total # of testing utterances	96	67
Average tokens per selected utterance	11.72	9.53

Table 4.1: Data set statistics for Lucent and BBN systems.

Discussion This algorithm uses the conversation to drive learning in two ways: it guides the lexical items that are proposed while also providing the conversational feedback that defines the loss used to update the parameters. The resulting approach is, at every step, using information about how the conversation progressed after a user utterance to reconstruct the meaning of the original statement.

4.6 Data Sets

For evaluation, we used conversation logs from the Lucent and BBN dialog systems in the DARPA Communicator corpus (Walker et al., 2002). We selected these systems since they provide significant opportunities for learning. They asked relatively open ended questions, allowing for more complex user responses, while also using a number of simple remediating strategies to recover from misunderstandings. The original conversational logs included unannotated transcripts of system and user utterances. Inspired by the speech act labeling approach of Walker and Passonneau (2001), we wrote a set of scripts to label the speech acts and logical forms for system statements. This could be done with high accuracy since the original text was generated with templates. These labels represent what the system explicitly said and do not require complex, potentially error-prone annotation of the full state of the original dialog system. The set of speech acts includes confirmations, information requests, selects, offers, instructions, and a miscellaneous category.

The data sets include a total of 376 conversations, divided into training and testing sets. Ta-

ble 4.1 provides details about the training and testing sets, as well as general data set statistics. We developed our system using 4-fold cross validation on the training sets. Although there are approximately 12,000 user utterances in the data sets, the vast majority are simple, short phrases (such as *yes* or *no*) which are not useful for learning a semantic parser. We select user utterances with a small set of heuristics, including a threshold (6 for Lucent, 4 for BBN) on the number of words and requiring that at least one noun phrase is present from our initial lexicon. This approach was manually developed to perform well on the training sets, but is not perfect and does introduce a small amount of noise into the data.

4.7 Experimental Setup

This section describes our experimental setup and comparisons. We follow the setup of Zettlemoyer and Collins (2007) where possible, including feature design, initialization of the semantic parser, and evaluation metrics, as reviewed below.

Features and Parser The features include indicators for lexical item use, properties of the logical form that is being constructed and for parsing operators used to build the tree. The parser attempts to boost recall with a two-pass strategy that allows for word skipping if the initial parse fails.

Initialization and Parameters We use an initial lexicon that includes a list of domain-specific noun phrases, such as city and airport names, and a list of domain-independent categories for closed-class words such as *the* and *and*. We also used a time and number rule-based parser to expand this lexicon for each input sentence with the BIU Number Normalizer.[1] The learning parameters were tuned using the development sets: the margin constant γ is set to 0.5, we use 6 iterations and take the top 30 parses for lexical generation (line 6, Algorithm 1). The parser used for parameter update (lines 12-14, Algorithm 1) has a beam of 250. The parameter vector is initialized to $\bar{0}$.

[1] `http://www.cs.biu.ac.il/~nlp/downloads/`

Exact Match Metric	Lucent			BBN		
	Precision	Recall	F1	Precision	Recall	F1
Without conversational loss	0.35	0.34	0.35	0.66	0.54	0.59
Without domain loss	0.42	0.42	0.42	0.69	0.56	0.61
Our Approach	0.63	0.61	0.62	0.77	0.64	0.69
Supervised method	0.76	0.75	0.75	0.81	0.67	0.73

Table 4.2: Mean exact-match results for cross fold evaluation on the development sets.

Evaluation Metrics For evaluation, we measure performance against gold standard labels. We report both the number of exact matches, fully correct logical forms, and a partial-credit number. We measure partial-credit accuracy by mapping logical forms to attribute-value pairs (for example, the expression $\mathrm{from}(x, \mathrm{LA})$ will be mapped to $\mathrm{from} = \mathrm{LA}$) and report precision and recall on attribute sets. This more lenient measure does not test the overall structure of the logical expression, only its components.

Systems We compare performance with the following systems:

Full Supervision: We measured how a fully supervised approach would perform on our data by hand-labeling the training data and using a 0-1 loss function that tests if the output logical form matches the labeled one. For lexicon generation, the labels were used instead of the conversation.

No Conversation Baseline: We also report results for a no conversation baseline. This baseline system is constructed by making two modifications to the full approach. We remove the conversation loss function and apply the GENLEX templates to every possible logical constant, instead of only those in the conversation. This baseline allows us to measure the importance of having access to the conversations by completely ignoring the context for each sentence.

Ablations: In addition to the baseline above, we also do ablation tests by turning off various individual components of the complete algorithm.

Exact Match Metric	Lucent			BBN		
	Precision	Recall	F1	Precision	Recall	F1
No Conversations Baseline	0	0	0	0.16	0.15	0.15
Our Approach	0.58	0.55	0.56	0.85	0.75	0.79
Supervised method	0.7	0.68	0.69	0.87	0.78	0.82
Partial Credit Metric	**Lucent**			**BBN**		
	Precision	Recall	F1	Precision	Recall	F1
No Conversations Baseline	0.26	0.35	0.29	0.26	0.33	0.29
Our Approach	0.68	0.63	0.65	0.97	0.57	0.72
Supervised method	0.75	0.68	0.72	0.96	0.68	0.79

Table 4.3: Exact- and partial-match results on the test sets.

4.8 Results

Table 4.2 shows exact match results for the development sets, including different system configurations. We report mean results across four folds. To verify their contributions, we include results where we ablate the conversational loss and domain loss functions. Both are essential.

The test results are listed in Table 4.3. The full method significantly outperforms the baseline, indicating that we are making effective use of the conversational feedback, although we do not yet match the fully supervised result. The poor baseline performance is not surprising, given the difficulty of the task and lack of guidance when the conversations are removed. The partial-credit numbers also demonstrate a empirical trend that we observed; in many cases where we do not produce the correct logical form, the output is often close to correct, with only one or two missed flight constraints.

The difference between the two systems is evident. The BBN system presents a simpler approach to the dialog problem by creating a more constrained conversation. This is done by handling one flight at a time, in the case of flight planing, and posing simple and close ended questions to the user. Such an approach encourages the user to make simpler requests, with relatively few constraints in each request. In contrast, the Lucent system presents a less-constrained approach: interactions start with an open ended prompt and the conversations flow in a more natural, less

constrained fashion. BBN's simplified approach makes it easier for learning, giving us superior performance when compared to the Lucent system, despite the smaller training set. This is true for both our approach and supervised learning.

We compared the logical forms recovered by the best conversational model to the labeled ones in the training set. Many of the errors came from cases where the dialog system never fully recovered from confusions in the conversation. For example, the Lucent system almost never understood user utterances that specified flight arrival times. Since it was unable to consistently recover and introduce this constraint, the user would often just recalculate and specify a departure time that would achieve the original goal. This type of failure provides no signal for our learning algorithm, whereas the fully supervised algorithm would use labeled logical forms to resolve the confusion. Interestingly, the test set had more sentences that suffered such failures than the development set, which contributed to the performance gap.

Chapter 5

WEAKLY SUPERVISED LEARNING OF SEMANTIC PARSERS FOR MAPPING INSTRUCTIONS TO ACTIONS

In this chapter, we consider the problem of learning to execute natural language instructions. We focus on representation, inference and learning challenges. We propose to represent imperative sentences as sets of events to be executed using Neo-Davidsonian semantics (Davidson, 1967, 1969; Parsons, 1990), resulting in simpler learned grammars that generalize better during testing. Instructional language is often strongly context-dependent; its meaning varies according to the position of the agent, abilities and observations. To capture the interplay between language and the environment, we jointly reason about the meaning of language and its execution in the world. This allows to effectively account for implicit requests and the constantly-changing state of the world during execution. Finally, to learn from demonstrations of desired system behavior, we present a validation-based variant of the learning algorithm introduced in Chapter 4. When applied to the navigation domain, our approach outperforms the previous state of the art by 60%. This chapter is based on work originally described in Artzi and Zettlemoyer (2013b).

5.1 Introduction

The context in which natural language is used provides a strong signal to reason about its meaning. However, using such a signal to automatically learn to understand unrestricted natural language remains a challenging, unsolved problem.

For example, consider the instructions in Figure 5.1. Correct interpretation requires us to solve many subproblems, such as resolving all referring expressions to specific objects in the environment (including, *the corner* or *the third intersection*), disambiguating word sense based on context (e.g., *the chair* could refer to a chair or sofa), and finding executable action sequences that sat-

move forward twice to the chair
$\lambda a.\text{move}(a) \wedge \text{dir}(a, \text{FORWARD}) \wedge \text{len}(a, 2) \wedge \text{to}(a, \iota(\lambda x.\text{chair}(x)))$
at the corner turn left to face the blue hall
$\lambda a.pre(a, \iota(\lambda x.\text{corner}(x))) \wedge \text{turn}(a) \wedge \text{dir}(a, \text{LEFT}) \wedge$
$\text{post}(a, \text{front}(\text{YOU}, \iota(\lambda x.\text{blue}(x) \wedge \text{hall}(x))))$
move to the chair in the third intersection
$\lambda a.\text{move}(a) \wedge \text{to}(a, \iota(\lambda x.\text{sofa}(x))) \wedge \text{intersect}(\text{order}(\lambda y.\text{junction}(y), \text{frontdist}, 3), x)$

Figure 5.1: A sample navigation instruction set. Each instruction is paired with its lambda-calculus meaning representation.

isfy stated constraints (such as *twice* or *to face the blue hall*). We must also understand implicit requests, for example from the phrase *at the corner*, that describe goals to be achieved without specifying the specific steps. Finally, to do all of this robustly without prohibitive engineering effort, we need *grounded* learning approaches that jointly reason about meaning and context to learn directly from their interplay, with as little human intervention as possible.

Although many of these challenges have been studied separately, this chapter represents, to the best of our knowledge, the first attempt at a comprehensive model that addresses them all. Our approach induces a weighted CCG, including both the parameters of the linear model and a CCG lexicon. To model complex instructional language, we introduce a new semantic modeling approach that can represent a number of key linguistic constructs that are common in spatial and instructional language. To learn from indirect supervision, we define the notion of a validation function, for example that tests the state of the agent after interpreting an instruction. We then show how this function can be used to drive online learning. For that purpose, we adapt the loss-sensitive Perceptron algorithm (Singh-Miller and Collins, 2007; Artzi and Zettlemoyer, 2011) to use a validation function and coarse-to-fine inference for lexical induction.

The joint nature of this approach provides crucial benefits in that it allows situated cues, such as the set of visible objects, to directly influence parsing and learning. It also enables the model to be learned while executing instructions, for example by trying to replicate actions taken by humans. In particular, we show that, given only a small seed lexicon and a task-specific executor, we can

induce high quality models for interpreting complex instructions.

We evaluate the method on a benchmark navigational instructions dataset (MacMahon et al., 2006; Chen and Mooney, 2011). Our joint approach successfully completes 60% more instruction sets relative to the previous state of the art. We also report experiments that vary supervision type, finding that observing the final position of an instruction execution is nearly as informative as observing the entire path. Finally, we present improved results on a new version of the corpus, which we filtered to include only executable instructions paired with correct traces.

5.2 Technical Overview

Task Let $\mathcal{S}$ be the set of possible environment states and $\mathcal{A}$ be the set of possible actions. Given a start state $s \in \mathcal{S}$ and a natural language instruction x, we aim to generate a sequence of actions $\vec{a} = \langle a_1, \ldots, a_n \rangle$, with each $a_i \in \mathcal{A}$, that performs the steps described in x.

For example, in the navigation domain (MacMahon et al., 2006), $\mathcal{S}$ is a set of positions on a map. Each state $s = (x, y, o)$ is a triple, where x and y are integer grid coordinates and $o \in \{0, 90, 180, 270\}$ is an orientation. Figure 5.2 shows an example map with 36 states; the ones we use in our experiments contain an average of 141. The space of possible actions $\mathcal{A}$ is $\{\text{LEFT}, \text{RIGHT}, \text{MOVE}, \text{NULL}\}$. Actions change the state of the world according to a transition function $T : \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{S}$. In our navigation example, moving forward can change the x or y coordinates while turning changes the orientation o.

Model To map instructions to actions, we jointly reason about linguistic meaning and action execution. We use a weighted CCG grammar to rank possible meanings z for each instruction x. Section 5.4 defines how to design such grammars for instructional language. Each logical form z is mapped to a sequence of actions $\vec{a}$ with a deterministic executor, as described in Section 5.5. The final *grounded CCG* model, detailed in Section 5.4.3, jointly constructs and scores z and $\vec{a}$, allowing for robust situated reasoning during semantic interpretation.

Learning We assume access to a training set containing N examples $\{(x_i, s_i, \mathcal{V}_i) : i = 1 \dots N\}$, each containing a natural language sentence x_i, a start state s_i, and a validation function $\mathcal{V}_i$. The validation function $\mathcal{V}_i : A \to \{0, 1\}$ maps an action sequence $\vec{a} \in A$ to 1 if it's correct according to available supervision, or 0 otherwise. This training data contains no direct evidence about the logical form z for each x_i, or the grounded CCG analysis used to construct z. We model all these choices as latent variables. We experiment with two validation functions. The first, $\mathcal{V}^D(\vec{a})$, has access to an observable demonstration of the execution $\vec{a}_i$, a given $\vec{a}$ is valid iff $\vec{a} = \vec{a}_i$. The second, $\mathcal{V}_i^S(\vec{a})$, only encodes the final state s_i' of the execution of x, therefore $\vec{a}$ is valid iff its final state is s_i'. Since numerous logical forms often execute identically, both functions provide highly ambiguous supervision.

Evaluation We evaluate task completion for single instructions on a test set $\{(x_i, s_i, s_i') : i = 1 \dots M\}$, where s_i' is the final state of an oracle agent following the execution of x_i starting at state s_i. We will also report accuracies for correctly interpreting instruction sequences $\vec{a}$, where a single error can cause the entire sequence to fail. Finally, we report accuracy on recovering correct logical forms z_i on a manually annotated subset of the test set.

5.3 Spatial Environment Modeling

We will execute instructions in an environment, see Section 5.2, which has a set of positions. A position is a triple (x, y, o), where x and y are horizontal and vertical coordinates, and $o \in O = \{0, 90, 180, 270\}$ is an orientation. A position also includes properties indicating the object it contains, its floor pattern and its wallpaper. For example, the square at $(4, 3)$ in Figure 5.2 has four positions, one per orientation.

Since instructional language refers to objects and other structures in an environment, we introduce the notion of a position set. For example, in Figure 5.2, the position set $D = \{(5, 3, o) : o \in O\}$ represents a chair, while $B = \{(x, 3, o) : o \in O, x \in [0 \dots 5]\}$ represents the blue floor. Both sets contain all orientations for each (x, y) pair, thereby representing properties of regions. Position sets can have many properties. For example, E, in addition to being a chair, is also an intersection

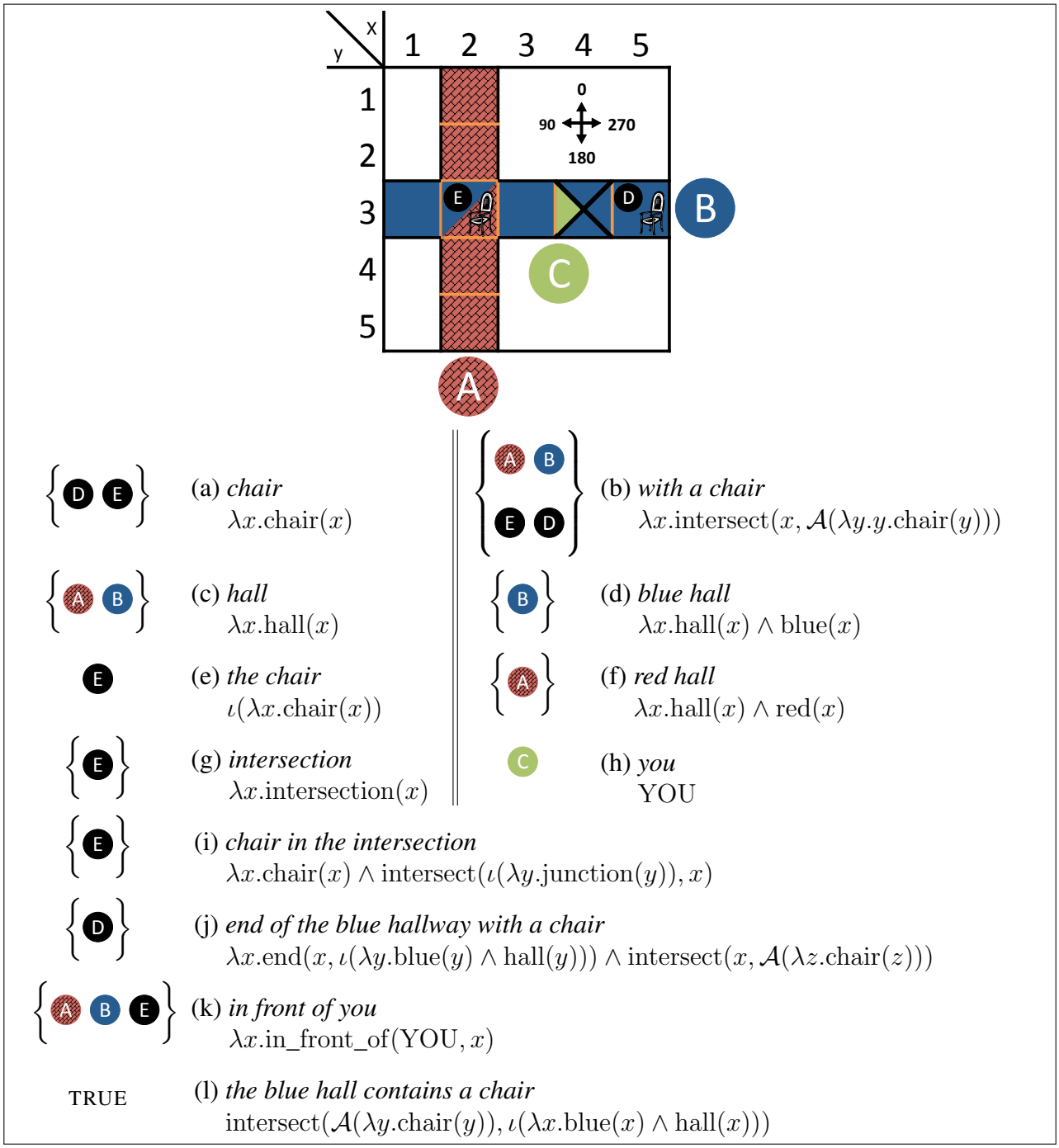


Figure 5.2: Schematic diagram of a map environment and examples of formal representation of spatial phrases.

because it overlaps with the neighboring halls A and B. The set of possible entities includes all position sets and a few additional entries. For example, set $C = \{(4, 3, 90)\}$ in Figure 5.2 represents the agent's position.

5.4 Modeling Instructional Language

We aim to design a semantic representation that is learnable, models grounded phenomena such as spatial relations and object reference, and is executable.

Our semantic representation combines ideas from Carpenter (1997) and Neo-Davidsonian event semantics (Parsons, 1990) in a simply-typed lambda calculus. There are four basic types: (1) entities e that are objects in the world, (2) events ev that specify actions in the world, (3) truth values t, and (4) meta-entities m, such as numbers or directions. We also allow functional types, which are defined by input and output types. For example, $\langle e, t\rangle$ is the type of function from entities to truth values.

5.4.1 Spatial Language Modeling

Nouns and Noun Phrases Noun phrases are paired with e-type constants that name specific entities and nouns are mapped to $\langle e, t\rangle$-type expressions that define a property. For example, the noun *chair* (Figure 5.2a) is paired with the expression $\lambda x.\mathrm{chair}(x)$, which defines the set of objects for which the $\langle e, t\rangle$-typed constant chair returns true. The denotation of this expression is the set $\{D, E\}$ in Figure 5.2 and the denotation of $\lambda x.\mathrm{hall}(x)$ (Figure 5.2c) is $\{A, B\}$. Also, the noun phrase *you* (Figure 5.2h), which names the agent, is represented by the constant YOU with denotation C, the agent's position.

Determiners Noun phrases can also be formed by combining nouns with determiners that pick out specific objects in the world. We consider both definite reference, which names contextually unique objects, and indefinites, which are less constrained.

The definite article is paired with a $\langle\langle e, t\rangle e\rangle$-typed logical expression ι, which, given a set, selects a single object in the world. For example, the phrase *the chair* in Figure 5.2e will be

represented by $\iota(\lambda x.\text{chair}(x))$ which will denote the appropriate chair. However, computing this denotation is challenging when there is perceptual ambiguity, for positions where multiple chairs are visible. We adopt a simple heuristic approach that ranks referents based on a combination of their distance from the agent and whether they are in front of it. For our example, from position C our agent would pick the chair E in front of it as the denotation. The approach differs from previous, non-grounded models that fail to name objects when faced with such ambiguity (e.g., Carpenter, 1997; Heim and Kratzer, 1998).

To model the meaning of indefinite articles, we depart from the Frege-Montague tradition of using existential quantifiers (Lewis, 1970; Montague, 1973; Barwise and Cooper, 1981), and instead introduce a new quantifier $\mathcal{A}$ that, like ι, has type $\langle\langle e,t\rangle,e\rangle$. For example, the phrase *a chair* would be paired with $\mathcal{A}(\lambda x.\text{chair}(x))$ which denotes an arbitrary entry from the set of chairs in the world. Computing the denotation for such expressions in a world will require picking a specific object, without further restrictions. This approach is closely related to Steedman's (2011) generalized Skolem terms.[1]

Meta Entities We use m-typed terms to represent non-physical entities, such as numbers (1, 2, etc.) and directions (LEFT, RIGHT, etc.) whose denotations are fixed. The ability to refer to directions allows us to manipulate position sets. For example, the phrase *your left* is mapped to the logical expression $\text{orient}(\text{YOU}, \text{LEFT})$, which denotes the position set containing the position to the left of the agent.

Prepositions and Adjectives Noun phrases with modifiers, such as adjectives and prepositional phrases are $\langle e,t\rangle$-type expressions that implement set intersection with logical conjunctions. For example in Figure 5.2, the phrase *blue hall* is paired with $\lambda x.\text{hall}(x)\wedge\text{blue}(x)$ with denotation $\{B\}$ and the phrase *chair in the intersection* is paired with $\lambda x.\text{chair}(x)\wedge\text{intersect}(\iota(\lambda y.\text{junction}(y)),x)$ with denotation $\{E\}$. Intuitively, the adjective *blue* introduces the constant blue and *in* adds a

[1] Steedman (2011) uses generalized Skolem terms as a tool for resolving anaphoric pronouns, which we do not model.

$\mathrm{intersect}$. We will describe the details of how these expressions are constructed in Section 5.4.3.

Spatial Relations The semantic representation allows more complex reasoning over position sets and the relations between them. For example, the binary relation $\mathrm{in\_front\_of}$ (Figure 5.2k) tests if the first argument is in front of the second from the point of view of the agent. Additional relations are used to model set intersection, relative direction, relative distance, and relative ordering.

5.4.2 Modeling Instructions

To model actions in the world, we adopt Neo-Davidsonian event semantics (Davidson, 1967; Parsons, 1990), which treats events as ev-type primitive objects. Such an approach allows for a compact lexicon where adverbial modifiers introduce predicates, which are linked by a shared ev-typed variable argument.

Instructional language is characterized by heavy usage of imperatives, which we model as functions from events to truth values.[2] For example, an imperative such as *move* would have the meaning $\lambda a.\mathrm{move}(a)$, which defines a set of events that match the specified constraints. Here, this set would include all events that involve moving actions.

The denotation of ev-type terms is a sequence of n instances of the same action. In this way, an event defines a function $ev : s \rightarrow s'$, where s is the start state and s' the end state. For example, the denotation of $\lambda a.\mathrm{move}(a)$ is the set of move action sequences $\{\langle \mathrm{MOVE}_1, \ldots, \mathrm{MOVE}_n \rangle : n \geq 1\}$. Although performing actions often requires performing additional ones (e.g., the agent might have to $turn$ before being able to $move$), we treat such actions as implicit (Section 5.5.1), rather then modeling them explicitly within the logical form.

Predicates such as move (seen above) and turn are introduced by verbs. Events can also be modified by adverbials, which are intersective, much like prepositional phrases. For example in the imperative, logical form (LF) pair:

---

[2] Imperatives are $\langle ev, t \rangle$-type, much like $\langle e, t \rangle$-type wh-interrogatives. Both define sets, the former includes actions to execute, the later defines answers to a question.

Imperative: *move from the sofa to the chair*

LF: $\lambda a.\text{move}(a) \wedge \text{to}(a, \iota(\lambda x.\text{chair}(x))) \wedge \text{from}(a, \iota(\lambda y.\text{sofa}(y)))$

Each adverbial phrase provides a constraint, and changing their order will not change the LF.

5.4.3 Parsing Instructional Language with CCG

To compose logical expressions from sentences we use CCG, as described in Section 2.2. Figures 5.3 and 5.4 present a sample of lexical entries and how they are combined, as we will describe in this section. The basic syntactic categories we use are N (noun), NP (noun phrase), S (sentence), PP (prepositional phrase), AP (adverbial phrase), ADJ (adjective) and C (a special category for coordinators).

Type Shifting To compactly model syntactic variations, we follow Carpenter (1997), who argues for polymorphic typing. We include the more simple, or lower type, entry in the lexicon and introduce type-shifting rules to reconstruct the other when necessary at parse time. We use four unary type-shifting rules:

$PP : g \rightarrow N\backslash N : \lambda f.\lambda x.f(x) \wedge g(x)$

$ADJ : g \rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$

$AP : g \rightarrow S\backslash S : \lambda f.\lambda a.f(a) \wedge g(a)$

$AP : g \rightarrow S/S : \lambda f.\lambda a.f(a) \wedge g(a)$

The first three are for prepositional, adjectival and adverbial modifications, and the fourth models the fact that adverbials are often topicalized.[3] Figures 5.3 and 5.4 demonstrate using these rules.

Indefinites As discussed in Section 5.4.1, following Steedman (2011), we use a new syntactic analysis for indefinites. Previous approaches would build parses such as

[3]Using type-raising rules can be particularly useful when learning from sparse data. For example, it will no longer be necessary to learn three lexical entries for each adverbial phrase (with syntax AP, $S\backslash S$, and S/S).

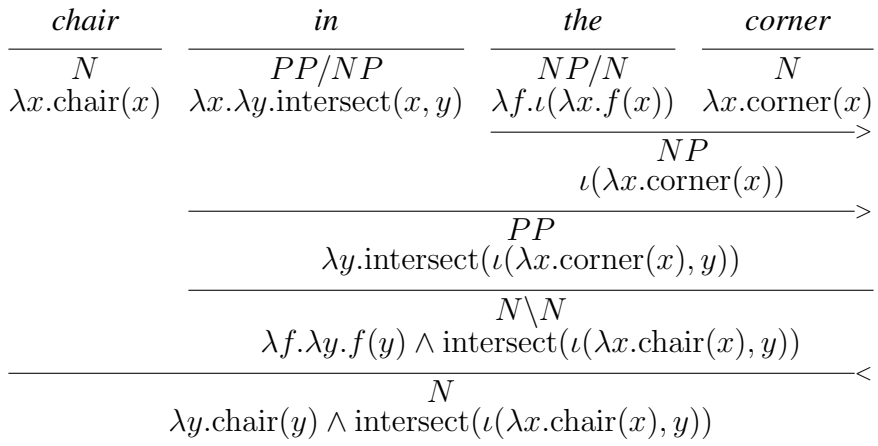
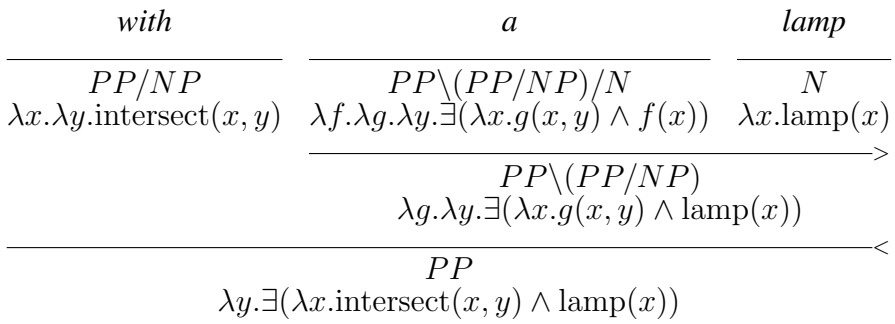


Figure 5.3: A CCG parse with a prepositional phrase.



where *a* has the relatively complex syntactic category $PP\backslash(PP/NP)/N$ and where similar entries would be needed to quantify over different types of verbs (e.g., $S\backslash(S/NP)/N$) and adverbials (e.g., $AP\backslash(AP/NP)/N$). Instead, we include a single lexical entry $a \vdash NP/N : \lambda f.\mathcal{A}(\lambda x.f(x))$ which can be used to construct the correct meaning in all cases.

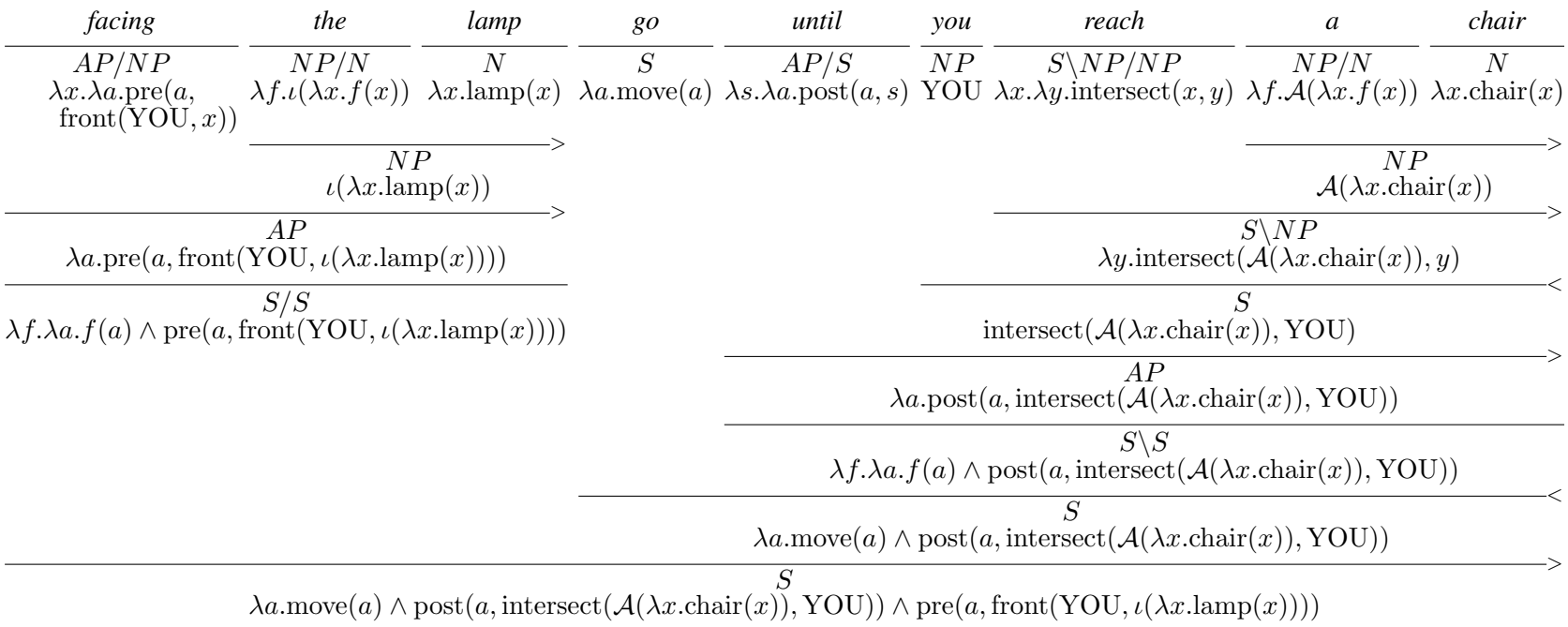


Figure 5.4: A CCG parse showing adverbial phrases and topicalization.

5.5 Joint Parsing and Execution

Our inference includes an execution component and a parser. The parser maps sentences to logical forms, and incorporates the grounded execution model. We first discuss how to execute logical forms, and then describe the joint model for execution and parsing.

5.5.1 Executing Logical Expressions

Dynamic Models In spatial environments, such as the ones in our task, the agent's ability to observe the world depends on its current state. Taking this aspect of spatial environments into account is challenging, but crucial for correct evaluation.

To represent the agent's point of view, for each state $s \in \mathcal{S}$, as defined in Section 5.2, let M_s be the state-dependent logical model. A model M consists of a domain $D_{M,T}$ of objects for each type T and an interpretation function $\mathcal{I}_{M,T} : O_T \rightarrow D_{M,T}$, where O_T is the set of T-type constants. $\mathcal{I}_{M,T}$ maps logical symbols to T-type objects, for example, it will map YOU to the agent's position. We have domains for position sets, actions and so on. Finally, let V_T be the set of variables of type T, and $A_T : V_T \rightarrow \bigcup_{s \in \mathcal{S}} D_{Ms,T}$ be the assignment function, which maps T-typed variables to domain objects.

For each model M_s the domain $D_{M_s,ev}$ is a set of action sequences $\{\langle a_1, ..., a_n \rangle : n \geq 1\}$. Each $\vec{a}$ defines a sequences of states s_i, as defined in Section 5.4.2, and associated models M_{s_i}. The key challenge for execution is that modifiers of the event will need to be evaluated under different models from this sequence. For example, consider the sentence in Figure 5.4. To correctly execute, the pre literal, introduced by the *facing* phrase, it must be evaluated in the model M_{s_0} for the initial state s_0. Similarly, the literal including post requires the final model $M_{s_{n+1}}$. Such state dependent predicates, including pre and post, are called *stateful*. The list of stateful predicates is pre-defined and includes event modifiers, as well the ι quantifier, which is evaluated under M_{s_0}, since definite determiners are assumed to name objects visible from the start position. In general, a logical expression is traversed depth first and the model is updated every time a stateful predicate is reached. For example, the two e-type YOU constants in Figure 5.4 will be evaluated under

different models: the one within the pre literal under the model M_{s_0}, and the one inside the post literal under the updated model $M_{s_{n+1}}$.

Evaluation Given a logical expression z, we can compute the interpretation $\mathcal{I}_{M_{s_0},T}(z)$ by recursively mapping each subexpression to an entry on the appropriate model M.

To reflect the changing state of the agent during evaluation, we define the function UPDATE$(\vec{a}, \mathrm{pred})$. Given an action sequence $\vec{a}$ and a stateful predicate pred, UPDATE returns a model M_s, where s is the state under which the literal containing pred should be interpreted, either the initial state or one visited while executing $\vec{a}$. For example, given the predicate post and the action sequence $\langle a_1, \ldots, a_n \rangle$, UPDATE$(\langle a_1, \ldots, a_n \rangle, \mathrm{post}) = M_{s_{n+1}}$, where s_{n+1} is the state of the agent following action a_n. By convention, the event variable is the first argument in literals that include one.

Given a T-type logical expression z and a starting state s_0, we compute its interpretation $\mathcal{I}_{M_{s_0},T}(z)$ recursively, following these three base cases:

- If z is a λ operator of type $\langle T_1, T_2 \rangle$ with a bound variable v and body b, $\mathcal{I}_{M_s,T}(z)$ is a set of pairs from $D_{T_1} \times D_{T_2}$, where $D_{T_1}, D_{T_2} \in M_s$. For each object $o \in D_{T_1}$, we create a pair (o, i) where i is the interpretation $\mathcal{I}_{M_s,T_2}(b)$ computed under a variable assignment function extended to map $A_{T_2}(v) = o$.

- If z is a literal $c(c_1, \ldots, c_n)$ with n arguments where c has type P and each c_i has type P_i, $\mathcal{I}_{M_s,T}(z)$ is computed by first interpreting the predicate c to the function $f = \mathcal{I}_{M_s,T}(z)$. In most cases, $\mathcal{I}_{M_s,T}(z) = f(\mathcal{I}_{M_s,P_1}(c_1), \ldots, \mathcal{I}_{M_s,P_n}(c_n))$. However, if c is a stateful predicate, such as pre or post, we instead first retrieve the appropriate new model, which reflects the updated state, $M_{s'} = \text{UPDATE}(\mathcal{I}_{M_s,P_1}(c_1), c)$, where c_1 is the event argument and $\mathcal{I}_{M_s,P_1}(c_1)$ is its interpretation. Then, the final results is $\mathcal{I}_{M_s,T}(z) = f(\mathcal{I}_{M_{s'},P_1}(c_1), \ldots, \mathcal{I}_{M_{s'},P_n}(c_n))$.

- If z is a T-type constant or variable, $\mathcal{I}_{M_s,T}(z)$.

The worst case complexity of the process is exponential in the number of bound variables. Although in practice we observed tractable evaluation in the majority of development cases we

considered, a more comprehensive and tractable evaluation procedure is an issue that we leave for future work.

Implicit Actions Instructional language rarely specifies every action required for execution. For a detailed discussion in the navigation domain, see MacMahon (2007). For example, the sentence in Figure 5.4 can be said even if the agent is not facing a blue hallway, with the clear implicit request that it should turn to face such a hallway before moving. To allow our agent to perform implicit actions, we extend the domain of ev-type variables by allowing the agent to prefix up to k_I action sequences before each explicit event. For example, in the agent's position in Figure 5.2 (set C), the set of possible events includes $\langle \text{MOVE}^I, \text{MOVE}^I, \text{RIGHT}^I, \text{MOVE} \rangle$, which contains two implicit sequences (marked by I).

Resolving Action Ambiguity Logical forms often fail to determine a unique action sequences, due to instruction ambiguity. For example, consider the instruction *go forward* and the agent state as specified in Figure 5.2 (set C). The instruction, which maps to $\lambda a.\text{move}(a) \wedge \text{forward}(a)$, evaluates to the set containing $\langle \text{MOVE} \rangle$, $\langle \text{MOVE}, \text{MOVE} \rangle$ and $\langle \text{MOVE}, \text{MOVE}, \text{MOVE} \rangle$, as well as five other sequences that have implicit prefixes followed by explicit MOVE actions. To resolve such ambiguity, we prefer shorter actions without implicit actions. In the example above, we will select $\langle \text{MOVE} \rangle$, which includes a single action and no implicit actions.

5.5.2 Joint Inference

We incorporate the execution procedure described above with a linear weighted CCG parser, as described in Section 2.2, to create a joint model of parsing and execution. Specifically, we execute logical forms in the current state and observe the result of their execution. For example, the word *chair* can be used to refer to different types of objects, including chairs, sofas, and barstools, in the maps domains. Our CCG grammar would include a lexical item for each meaning, but execution might fail depending on the presence of objects in the world, influencing the final parse output. Similarly, allowing implicit actions provides robustness when resolving these and other

ambiguities. For example, an instruction with the precondition phrase *from the chair* might require additional actions to reach the position with the named object.

To allow such joint reasoning we define an derivation d to include a parse tree d^y and trace $d^{\vec{a}}$, and define our feature function to be $\phi(x, s_i, d)$, where $sent$ is an instruction and s is the start state. This approach allows joint dependencies: the state of the world influences how the agent interprets words, phrases and even complete sentences, while language understanding determines actions.

Finally, to execute sequences of instructions, we execute each starting from the end state of the previous one, using a beam of size k_s.

5.6 Learning

Algorithm 2 presents the complete learning algorithm. Our approach is online, considering each example in turn and performing two steps: expanding the lexicon and updating parameters. The algorithm assumes access to a training set $\{(x_i, s_i, \mathcal{V}_i) : i = 1 \ldots N\}$, where each example includes an instruction x_i, starting state s_i and a validation function $\mathcal{V}_i$, as defined in Section 5.2. In addition the algorithm takes a seed lexicon Λ_0. The output is a joint model, that includes a lexicon Λ and parameters θ.

Coarse Lexical Generation Given an instruction x, state s and a validation function $\mathcal{V}$, we use the function GENLEX$(x, s, \mathcal{V}; \Lambda, \theta)$ to generate potential lexical entries, where Λ is the current lexicon and θ is a parameter vector. In GENLEX we use coarse logical constants, as described below, to efficiently prune the set of potential lexical entries. This set is then pruned further using more precise inference in line 5.

To compute GENLEX, we initially generate a large set of lexical entries and then prune most of them. The full set is generated by taking the cross product of a set of templates, computed by factoring out all templates in the seed lexicon Λ_0, and all logical constants. For example, if Λ_0 has a lexical item with the category $AP/NP : \lambda x.\lambda a.\mathrm{to}(a, x)$ we would create entries $w \vdash AP/NP : \lambda x.\lambda a.p(a, x)$ for every phrase w in x and all constants p with the same type as to.[4]

[4]Generalizing previous work (Kwiatkowski et al., 2011), we allow templates that abstract a subset of the constants. For example, the seed entry facing $\vdash AP/NP : \lambda x.\lambda a.\mathrm{pre}(a, \mathrm{front}(\mathrm{YOU}, x))$ would create 7 templates.

Algorithm 2 Validation-driven situated learning algorithm.

Input: Training set $\{(x_i, s_i, \mathcal{V}_i) : i = 1 \dots n\}$ where x_i is a sentence, s_i is a state and $\mathcal{V}_i$ is a validation function, as described in Section 5.2, initial lexicon Λ_0, number of iterations T, margin γ and beam size k for lexicon generation.

Definitions: Let a derivation d include a parse tree d^y and a trace $d^{\vec{a}}$. $\textsc{Gen}(x, s; \Lambda, \theta)$ is the set of all possible derivations for the instruction x and state s, given the lexicon Λ and parameters θ. $\textsc{Lex}(y)$ is the set of lexical entries used in the parse tree y. $\textsc{GenMax}(x, s, \mathcal{V}, \lambda_G; \Lambda, \theta)$ is the set of max-scoring derivations d for x and s, given Λ and θ, s.t. $\mathcal{V}(d^{\vec{a}}) = 1$ and $\textsc{Lex}(d^y) \cap \lambda_G \leq 1$, as described in Section 5.6. Let $\phi_i(e)$ be shorthand for the feature function $\phi(x_i, s_i, d)$ defined in Section 5.5.2. Define $\Delta_i(d, d') = |\phi_i(d) - \phi_i(d')|_1$. $\textsc{GenLex}(x, s, \mathcal{V}; \lambda, \theta)$ takes as input an instruction x, state s, validation function $\mathcal{V}$, lexicon λ and model parameters θ, and returns a set of lexical entries, as defined in Section 5.6. Finally, for a set of derivations D let $\text{MAX}\mathcal{V}_\text{i}(\text{D}; \theta)$ be $\{d | \forall d' \in D, \langle \theta, \phi_i(d') \rangle \leq \langle \theta, \phi_i(d) \rangle \wedge \mathcal{V}_i(d^{\vec{a}}) = 1\}$, the set of highest scoring valid executions.

Output: Lexicon Λ and model parameters θ.

$\Lambda \leftarrow \Lambda_0$
for t = 1 ... T, i = 1 ... n **do**
  $\lambda_G \leftarrow \textsc{GenLex}(x_i, s_i, \mathcal{V}_i; \Lambda, \theta)$, $\lambda \leftarrow \Lambda \cup \lambda_G$
  » Get max-scoring valid derivations with at most one new lexical entry.
  $D \leftarrow \textsc{GenMax}(x_i, s_i, \mathcal{V}_i, \lambda_G; \lambda, \theta)$
  » Select lexical entries from the highest scoring valid derivations.
  $\lambda_i \leftarrow \bigcup_{e \in \text{MAX}\mathcal{V}_\text{i}(\text{D};\theta)} \textsc{Lex}(d^y)$
  » Update lexicon.
  $\Lambda \leftarrow \Lambda \cup \lambda_i$
  » Get valid and invalid derivation for parameter update.
  $G_i \leftarrow \text{MAX}\mathcal{V}_\text{i}(\textsc{Gen}(\text{x}_\text{i}, \text{s}_\text{i}; \Lambda); \theta)$
  $B_i \leftarrow \{d | d \in \textsc{Gen}(x_i, s_i; \Lambda) \wedge \mathcal{V}_i(d^{\vec{a}}) \neq 1\}$
  » Construct sets of margin violating good and bad parses.
  $R_i \leftarrow \{g | g \in G_i \ \wedge \exists b \in B_i \ s.t. \ \langle \theta, \phi_i(g) - \phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$
  $E_i \leftarrow \{b | b \in B_i \ \wedge \exists g \in G_i \ s.t. \ \langle \theta, \phi_i(g) - \phi_i(b) \rangle < \gamma \Delta_i(g, b)\}$
  » Apply the additive update.
  $\theta \leftarrow \theta + \frac{1}{|R_i|} \sum_{r \in R_i} \phi_i(r) - \frac{1}{|E_i|} \sum_{e \in E_i} \phi_i(e)$
return Λ and θ

On the development set, this approach often generated nearly 100k entries per sentence. To ease the cost of parsing at this scale, we developed a coarse-to-fine two-pass parsing approach that limits the number of new entries considered. The algorithm first parses with coarse lexical entries that abstract the identities of the logical constants in their logical forms, thereby greatly reducing

the search space. It then uses the highest scoring coarse parses to constrain the lexical entries for a final, fine parse.

Formally, we construct the coarse lexicon λ_a by replacing all constants with a common ancestor type α with a single newly created, α-typed temporary constant. We then parse to create a set of trees A, such that each $y \in A$

1. is a parse for sentence x, given the world state s with the combined lexicon $\Lambda \cup \lambda_a$,

2. scored higher than d^y by at least a margin of δ_L, where d^y is the tree of d, the highest scoring derivation of x, at position s under the current model, s.t. $\mathcal{V}(d^{\vec{a}}) = 1$,

3. contains at most one entry from λ_a.

Finally, from each entry $l \in \{l \mid l \in \lambda_a \wedge \exists y \in A.l \in \textsc{Lex}(y)\}$, we create multiple lexical entries by replacing all temporary constants with all possible appropriately typed constants from the original set. GENLEX returns all these lexical entries, which will be used to form our final fine-level analysis.

Lexical Induction To expand our model's lexicon, we use GENLEX to generate candidate lexical entries and then further refine this set by parsing with the current model. In line 3, we use GENLEX to create a temporary set of potential lexical entries λ_G. In lines 5-7, we select a small subset of these lexical entries to add to the current lexicon Λ: we find the max-scoring valid executions under the model, which use at most one entry from λ_G, and add their lexical entries to the current lexicon.

Parameter Update We use a variant of a loss-driven perceptron (Singh-Miller and Collins, 2007; Artzi and Zettlemoyer, 2011) for parameter updates. However, instead of taking advantage of a loss function we use a validation signal. In lines 11-12 we collect the highest scoring valid parses and all invalid parses. Then, in lines 14-15 we construct the set R_i of valid analyses and E_i of invalid ones, such that their model scores are not separated by a margin δ that is scaled by the number of wrong features (Taskar et al., 2003). Finally, in line 17 we apply the update.

Discussion The algorithm uses the validation signal to drive both lexical induction and parameter updates. Unlike previous work (Zettlemoyer and Collins, 2005, 2007; Artzi and Zettlemoyer, 2011), we have no access to a set of logical constants, either through the labeled logical form or the weak supervision signal, to guide the GENLEX procedure. Therefore, to avoid over-generating lexical entries, thereby making parsing and learning intractable, we leverage typing for coarse parsing to prune the generated set. By allowing a single new entry per derivation, we create a conservative, cascading effect, whereas a lexical entry that is introduced opens the way for many other sentence to be parsed and introduce new lexical entries. Furthermore, grounded features improve derivation selection, thereby generating higher quality lexical entries.

5.7 Experimental Setup

Data For evaluation, we use the navigation task from MacMahon et al. (2006), which includes three environments and the SAIL corpus of instructions and follower traces. Chen and Mooney (2011) segmented the data, aligned traces to instructions, and merged traces created by different subjects. The corpus includes raw sentences, without any form of linguistic annotation. The original collection process (MacMahon et al., 2006) created many uninterpretable instructions and incorrect traces. To focus on the learning and interpretation tasks, we also created a new dataset that includes only accurate instructions labeled with a single, correct execution trace. From this *oracle* corpus, we randomly sampled 164 instruction sequences (816 sentences) for evaluation, leaving 337 (1863 sentences) for training. This simple effort will allow us to measure the effects of noise on the learning approach and provides a resource for building more accurate algorithms. Table 5.1 compares the two sets.

Features and Parser Following Zettlemoyer and Collins (2005), we use a CKY parser with a beam of k. To boost recall during test, we adopt a two-pass strategy, which allows for word skipping if the initial parse fails. We use features that indicate usage of lexical entries, templates, lexemes and type-raising rules, as described in Section 5.4.3, and repetitions in logical coordinations. Finally, during joint parsing, we consider only parses executable at s_i as complete.

	Oracle	SAIL
Number of instruction sequences	501	706
Number of instruction sequences with implicit actions	431	
Total number of sentences	2679	3233
Mean sentences per sequence	5.35	4.61
Mean tokens per sentence	7.5	7.94
Vocabulary size	373	522

Table 5.1: Corpora statistics (lower-cased data).

Seed Lexicon To construct our seed lexicon we labeled 12 instruction sequences with 141 lexical entries. The sequences were randomly selected from the training set, so as to include two sequences for each participant in the original experiment. Figures 5.3 and 5.4 make use of sample entries from our seed lexicon.

Initialization and Parameters We set the weight of each template indicator feature to the number of times it is used in the seed lexicon and each repetition feature to -10. Learning parameters were tuned using cross-validation on the training set: the margin δ is set to 1, the GENLEX margin δ_L is set to 2, we use 6 iterations (8 for experiments on SAIL) and take the 250 top parses during lexical generation (line 5, Algorithm 2). For parameter update (lines 11-12, Algorithm 2) we use a parser with a beam of 100. GENLEX generates lexical entries for token sequences up to length 4. k_s, the instruction sequence execution beam, is set to 10. Finally, k_I is set to 2, allowing up to two implicit action sequences per explicit one.

Evaluation Metrics To evaluate single instructions x, we compare the agent's end state to a labeled state s', as described in Section 5.2. We use a similar method to evaluate the execution of instruction sequences $\vec{x}$, but disregard the orientation, since end goals in MacMahon et al. (2006) are defined without orientation. When evaluating logical forms we measure exact match accuracy.

	Single sentence	Sequence
Final state validation		
Complete system	81.98 (2.33)	59.32 (6.66)
No implicit actions	77.7 (3.7)	38.46 (1.12)
No joint execution	73.27 (3.98)	31.51 (6.66)
Trace validation		
Complete system	82.74 (2.53)	58.95 (6.88)
No implicit actions	77.64 (3.46)	38.34 (6.23)
No joint execution	72.85 (4.73)	30.89 (6.08)

Table 5.2: Cross-validation development accuracy and standard deviation on the oracle corpus.

	Single Sentence	Sequence
Chen and Mooney (2011)	54.4	16.18
Chen (2012)	57.28	19.18
+ additional data	57.62	20.64
Kim and Mooney (2012)	57.22	20.17
Trace validation	**65.28** (5.09)	**31.93** (3.26)
Final state validation	**64.25** (5.12)	**30.9** (2.16)

Table 5.3: Cross-validation accuracy and standard deviation for the SAIL corpus.

Validation	Single Sentence	Sequence	LF
Final state	77.6 (1.14)	54.63 (3.5)	44 (6.12)
Trace	78.63 (0.84)	58.05 (3.12)	51.05 (1.14)

Table 5.4: Oracle corpus test accuracy and standard deviation results.

5.8 Results

We repeated each experiment five times, shuffling the training set between runs. For the development cross-validation runs, we also shuffled the folds. As our learning approach is online, this allows us to account for performance variations arising from training set ordering. We report mean accuracy and standard deviation across all runs (and all folds).

Table 5.2 shows accuracy for 5-fold cross-validation on the oracle training data. We first varied the validation signal by providing the complete action sequence or the final state only, as described in Section 5.2. Although the final state signal is weaker, the results are similar. The relatively large difference between single sentence and sequence performance is due to (1) cascading errors in the more difficult task of sequential execution, and (2) corpus repetitions, where simple sentences are common (e.g., *turn left*). Next, we disabled the system's ability to introduce implicit actions, which was especially harmful to the full sequence performance. Finally, ablating the joint execution decreases performance, showing the benefit of the joint model.

Table 5.3 lists cross validation results on the SAIL corpus. To compare to previous work (Chen and Mooney, 2011), we report cross-validation results over the three maps. The approach was able to correctly execute 60% more sequences then the previous state of the art (Kim and Mooney, 2012). We also outperform the results of Chen (2012), which used 30% more training data.[5] Using the weaker validation signal creates a marginal decrease in performance. However, we still outperform all previous work, despite using weaker supervision. Interestingly, these increases were achieved with a relatively simple executor, while previous work used MARCO (MacMahon et al., 2006), which supports sophisticated recovery strategies.

Finally, we evaluate our approach on the held out test set for the oracle corpus (Table 5.4). In contrast to experiments on the Chen and Mooney (2011) corpus, we use a held out set for evaluation. Due to this discrepancy, all development was done on the training set only. The increase in accuracy over learning with the original corpus demonstrates the significant impact of noise on our performance. In addition to execution results, we also report exact match logical form (LF)

[5]This additional training data wasn't publicly available.

accuracy results. For this purpose, we annotated 18 instruction sequences (105 sentences) with logical forms. The gap between execution and LF accuracy can be attributed to the complexity of the linguistic representation and redundancy in instructions. These results provide a new baseline for studying learning from cleaner supervision.

Chapter 6

LEARNING COMPACT LEXICONS FOR CCG SEMANTIC PARSING

This chapter further studies the CCG lexicon induction problem. With the goal of inducting compact lexicons, we design a learning algorithm that uses global corpus statistics to make lexicon learning decisions. In contrast to the online algorithm presented in previous chapters, we choose to process the data in batch and observe the entire data when modifying the lexicon. We then propose two techniques: *voting* to select the best entries to add to the lexicon and *pruning* to remove entries. Experimental results on the navigation domain (Chapter 5) demonstrate new state-of-the-art performance with significantly smaller lexicons.

6.1 Introduction

In this chapter we present learning techniques to explicitly control the size of the CCG lexicon, and show that this results in improved task performance and more compact models. In most approaches for inducing CCGs for semantic parsing, lexicon learning and parameter estimation are performed jointly in an online algorithm, as introduced by Zettlemoyer and Collins (2007). To induce the lexicon, words extracted from the training data are paired with CCG categories one sample at a time (for an overview of CCG, see Section 2.2). Joint approaches have the potential advantage that only entries participating in successful parses are added to the lexicon. However, new entries are added greedily and these decisions are never revisited at later stages. In practice, this often results in a large and noisy lexicon.

Figure 6.1 lists a sample of CCG lexical entries learned for the word *chair* with a greedy joint algorithm (Chapter 5. In the studied navigation domain, the word *chair* is often used to refer to chairs and sofas, as captured by the first two entries. However, the system also learns several spurious meanings: the third shows an erroneous usage of *chair* as an adverbial phrase describing

chair $\vdash$ $\boldsymbol{N : \lambda x.\mathbf{chair}(x)}$
chair $\vdash$ $\boldsymbol{N : \lambda x.\mathbf{sofa}(x)}$
chair $\vdash AP : \lambda a.\text{len}(a, 3)$
chair $\vdash NP : \mathcal{A}(\lambda x.\text{corner}(x))$
chair $\vdash ADJ : \lambda x.\text{hall}(x)$

Figure 6.1: Lexical entries for the word *chair* as learned with no corpus-level statistics. Our approach is able to correctly learn only the top two bolded entries.

action length, while the fourth treats it as a noun phrase and the fifth as an adjective. In contrast, we now describe an approach that is able to correctly learn only the top two lexical entries.

We present a *batch* algorithm focused on controlling the size of the lexicon when learning CCG semantic parsers (Section 6.2). Because we make updates only after processing the entire training set, we can take corpus-wide statistics into account before each lexicon update. To explicitly control the size of the lexicon, we adopt two complementary strategies: *voting* and *pruning*. First, we consider the lexical evidence each sample provides as a vote towards potential entries. We describe two voting strategies for deciding which entries to add to the model lexicon (Section 6.3). Second, even though we use voting to only conservatively add new lexicon entries, we also prune existing entries if they are no longer necessary for parsing the training data. These steps are incorporated into the learning framework, allowing us to apply stricter criteria for lexicon expansion while maintaining a single learning algorithm.

We evaluate this approach on the robot navigation semantic parsing task, which we presented in Chapter 5. Our experimental results show that we outperform previous state of the art on executing sequences of instructions, while learning significantly more compact lexicons (Section 6.5 and Table 6.3).

6.2 Learning

Learning a CCG semantic parser requires inducing the entries of the lexicon Λ and estimating parsing parameters θ. We describe a batch learning algorithm (Algorithm 3), which explicitly

attempts to induce a compact lexicon, while fully explaining the training data. At training time, we assume access to a set of N examples $\mathcal{D} = \{(x_i, s_i, \mathcal{V}_i) : i = 1 \dots N\}$, where x_i is an instruction, s_i is the state where the instruction is issued and $\mathcal{V}_i$ is a validation function (Section 5.2). Since there may be many valid derivations d (i.e., $\mathcal{V}_i(d^{\vec{a}}) = 1$), we marginalize over all valid derivations when maximizing the following regularized log-likelihood:

$$\mathscr{L}(\theta, \Lambda, D) = \sum_{(x_i, s_i, \mathcal{V}_i) \in D} \sum_{d \in \mathcal{D}(\mathcal{V}_i)} p(d|x_i, s_i; \theta, \Lambda) - \frac{\gamma}{2}\|\theta\|_2^2 \ , \tag{6.1}$$

where $\mathcal{D}(\mathcal{V}_i)$ is the set of derivations where for each derivation d, $\mathcal{V}_i(d^{\vec{a}}) = 1$, and the hyper-parameter γ is a regularization constant. Due to the large number of potential combinations,[1] it is impractical to consider the complete set of lexical entries, where all strings (single words and n-grams) are associated with all possible CCG categories. Therefore, similar to prior work, we gradually expand the lexicon during learning. As a result, the parameter space changes throughout training whenever the lexicon is modified. The learning problem involves jointly finding the best set of parameters and lexicon entries. In the remainder of this section, we describe how we optimize Equation 6.1, while explicitly controlling the lexicon size.

6.2.1 Optimization Algorithm

We present a learning algorithm to optimize the data log-likelihood, where both lexicon learning and parameter updates are performed in *batch*, i.e., after observing all the training corpus. The batch formulation enables us to use information from the entire training set when updating the model lexicon. Algorithm 3 presents the outline of our optimization procedure. It takes as input a training dataset D, number of iterations T, seed lexicon Λ_0, learning rate μ and regularization constant γ.

Learning starts with initializing the model lexicon Λ using Λ_0 (line 1). In lines 2-14, we run T iterations; in each, we make two passes over the corpus, first to generate lexical entries, and second

[1] For the navigation task, given the set of CCG category templates (see Section 2.3) and parameters used there would be between 7.5-10.2M lexical entries to consider, depending on the corpus used (Section 6.4).

Algorithm 3 Batch algorithm for maximizing $\mathscr{L}(\theta, \Lambda, D)$. See Section 6.2.1 for details.

Input: Training dataset $D = \{(x_i, s_i, \mathcal{V}_i) : i = 1 \dots N\}$, number of learning iterations T, seed lexicon Λ_0, a regularization constant γ, and a learning rate μ.

Definitions: Let VOTE be a voting function, as described in Section 6.3. GENENTRIES (Algorithm 4) and COMPUTEUPDATE (Algorithm 5) are described in and Section 6.2.

Output: Lexicon Λ and model parameters θ

1: $\Lambda \leftarrow \Lambda_0$
2: **for** $t = 1$ to T **do**
3: » Generate lexical entries for all samples.
4: **for** $i = 1$ to N **do**
5: $\lambda_i \leftarrow \textsc{GenEntries}(x_i, s_i, \mathcal{V}_i, \theta, \Lambda)$
6: » Add corpus-wide voted entries to model lexicon.
7: $\Lambda \leftarrow \Lambda \cup \textsc{Vote}(\Lambda, \{\lambda_1, \dots, \lambda_N\})$
8: » Compute gradient and entries to prune.
9: **for** $i = 1$ to N **do**
10: $\langle \lambda_i^-, \Delta_i \rangle \leftarrow \textsc{ComputeUpdate}(x_i, s_i, \mathcal{V}_i, \theta, \Lambda)$
11: » Prune lexicon.
12: $\Lambda \leftarrow \Lambda \setminus \bigcap_{i=1}^{N} \lambda_i^-$
13: » Update model parameters.
14: $\theta \leftarrow \theta + \mu \sum_{i=1}^{N} \Delta_i - \gamma\theta$
15: **return** Λ and θ

to compute gradient updates and lexical entries to prune. To generate lexical entries (lines 4-5) we use the subroutine GENENTRIES to independently generate entries for each sample, as described in Section 6.2.2. Given the entries for each sample, we vote on which to add to the model lexicon. The subroutine VOTE (line 7) chooses a subset of the proposed entries using a particular voting strategy (see Section 6.3). Given the updated lexicon, we process the corpus a second time (lines 9-10). The subroutine COMPUTEUPDATE, as described in Section 6.2.3, computes the gradient update for each sample $(x_i, s_i, \mathcal{V}_i)$, and also generates the set of lexical entries not included in the max-scoring valid derivations for the sample, which are candidates for pruning. We prune from the model lexicon all lexical entries not used in any valid derivation (line 12). During this pruning step, we ensure that no entries from Λ_0 are removed from Λ. Finally, the gradient updates are accumulated to update the model parameters (line 14).

Algorithm 4 GENENTRIES: Algorithm to generate lexical entries from one training sample. See Section 6.2.2 for details.

Input: Sentence x, state s, validation function $\mathcal{V}$, model parameters θ and lexicon Λ.

Definitions: Let $\textsc{GenMax}(x, s, \mathcal{V}; \Lambda, \theta)$ be the set of viterbi derivations from sentence x and state s that are valid according to $\mathcal{V}$ given the lexicon Λ and parameters θ. $\textsc{Lex}(d)$ is the set of lexical entries used in the derivation d. GENLEX is defined in Section 5.6.

Output: Sample-specific lexical entries λ.

1: » Augment lexicon with sample-specific entries.
2: $\Lambda^+ \leftarrow \Lambda \cup \textsc{GenLex}(x, s, \mathcal{V}, \Lambda, \theta)$
3: » Get max-scoring valid derivations.
4: $\mathcal{D}^+ \leftarrow \textsc{GenMax}(x, s, \mathcal{V}; \Lambda^+, \theta)$
5: » Extract lexical entries from max-scoring derivations.
6: $\lambda \leftarrow \bigcup_{d \in \mathcal{D}^+} \textsc{Lex}(d^y)$
7: **return** λ

6.2.2 Lexical Entries Generation

For each training sample $(x, s, \mathcal{V})$, the subroutine GENENTRIES, as described in Algorithm 4, generates a set of potential entries. The subroutine uses the function GENLEX, as described in Section 5.6. Briefly, GENLEX uses the sentence and validation function to generate new lexemes, which are then paired with a set of templates factored from Λ_0 to generate new lexical entries.

Since GENLEX over-generates entries, we need to determine the set of entries that participate in max-scoring parses that lead to valid derivations. We therefore create a sentence-specific lexicon Λ_+ by taking the union of the GENLEX-generated entries for the current sentence and the model lexicon (line 2). We define $\textsc{GenMax}(x, s, \mathcal{V}; \Lambda_+, \theta)$ to be the set of all max-scoring parses according to the parameters θ that are in $\textsc{Gen}(x, s; \Lambda_+)$ and are valid according to $\mathcal{V}$ (line 4). In line 6 we use the function $\textsc{Lex}(d^y)$, which returns the lexical entries used in the parse tree d^y, to compute the set of all lexical entries used in these parses. This final set contains all newly generated entries for this datapoint and is returned to the optimization algorithm.

Algorithm 5 COMPUTEUPDATE: Algorithm to compute the gradient and the set of lexical entries to prune for one datapoint. See Section 6.2.3 for details.

Input: Sentence x, state s, validation function $\mathcal{V}$, model parameters θ and lexicon Λ.

Definitions: Let $\phi(x, s, d)$ be a l-dimensional feature vector, given a sentence x, state s and derivation d. GENMAX and LEX are defined in Algorithm 4.

Output: $\langle \lambda^-, \Delta \rangle$, lexical entries to prune for $\langle x, s, \mathcal{V} \rangle$ and gradient.

1: » Get max-scoring valid derivations given Λ and θ.
2: $\mathcal{D}^+ \leftarrow \textsc{GenMax}(x, s, \mathcal{V}; \Lambda, \theta)$
3: » Create the set of entries to prune.
4: $\lambda^- \leftarrow \Lambda \setminus \bigcup_{d \in \mathcal{D}^+} \textsc{Lex}(d^y)$
5: » Compute gradient.
6: $\Delta \leftarrow E_{p(d|\mathcal{V}(d)=1,x,s;\Lambda,\theta)}[\phi(x, s, d)] - E_{p(d|x,s;\Lambda,\theta)}[\phi(x, s, d)]$
7: **return** $\langle \lambda^-, \Delta \rangle$

6.2.3 Pruning and Gradient Computation

Algorithm 5 describes the subroutine COMPUTEUPDATE that, given a training sample $(x, s, \mathcal{V})$, the current model lexicon Λ and model parameters θ, returns the gradient update and the set of lexical entries to prune for the sampel. First, similar to GENENTRIES we compute the set of valid max-scoring derivations using GENMAX (line 2). This time, however, we do not use a sentence-specific lexicon, but instead use the model lexicon that has been previously expanded with all voted entries. As a result, the set of max-scoring parses producing the valid derivation may be different compared to GENENTRIES. $\textsc{Lex}(d^y)$ is then used to extract the lexical entries from these derivations, and the set difference (λ^-) between the model lexicon and these entries is set to be pruned (line 4). Finally, the partial derivative for the sample is computed using the difference of two expected feature vectors, according to two distributions (line 6): (a) derivations conditioned on the validation function $\mathcal{V}$, the sentence x, state s and the model, and (b) all derivations not conditioned on the validation function $\mathcal{V}$. The derivatives are approximate due to the use of beam search, as described in Section 2.4.

	Round 1	Round 2	Round 3	Round 4
x_1	$\langle$*chair*, {chair}$\rangle$ 1/3 $\langle$*chair*, {hatrack}$\rangle$ 1/3 $\langle$*chair*, {turn, direction}$\rangle$ 1/3	$\langle$*chair*, {chair}$\rangle$ 1/2 $\langle$*chair*, {hatrack}$\rangle$ 1/2	$\langle$*chair*, {chair}$\rangle$ 1	$\langle$*chair*, {chair}$\rangle$ 1
x_2	$\langle$*chair*, {chair}$\rangle$ 1/2 $\langle$*chair*, {hatrack}$\rangle$ 1/2	$\langle$*chair*, {chair}$\rangle$ 1/2 $\langle$*chair*, {hatrack}$\rangle$ 1/2	$\langle$*chair*, {chair}$\rangle$ 1	$\langle$*chair*, {chair}$\rangle$ 1
x_3	$\langle$*chair*, {chair}$\rangle$ 1/2 $\langle$*chair*, {easel}$\rangle$ 1/2	$\langle$*chair*, {chair}$\rangle$ 1/2 $\langle$*chair*, {easel}$\rangle$ 1/2	$\langle$*chair*, {chair}$\rangle$ 1/2 $\langle$*chair*, {easel}$\rangle$ 1/2	$\langle$*chair*, {chair}$\rangle$ 1
x_4	$\langle$*chair*, {easel}$\rangle$ 1	$\langle$*chair*, {easel}$\rangle$ 1	$\langle$*chair*, {easel}$\rangle$ 1	$\langle$*chair*, {easel}$\rangle$ 1
Votes	$\langle$*chair*, {chair}$\rangle$ 1 1/3 $\langle$*chair*, {easel}$\rangle$ 1 1/2 $\langle$*chair*, {hatrack}$\rangle$ 5/6 $\langle$*chair*, {turn, direction}$\rangle$ 1/3	$\langle$*chair*, {chair}$\rangle$ 1 1/2 $\langle$*chair*, {easel}$\rangle$ 1 1/2 $\langle$*chair*, {hatrack}$\rangle$ 1	$\langle$*chair*, {chair}$\rangle$ 2 1/2 $\langle$*chair*, {easel}$\rangle$ 1 1/2	**$\langle$*chair*, {chair}$\rangle$ 3** $\langle$*chair*, {easel}$\rangle$ 1
Discard	$\langle$*chair*, {turn, direction}$\rangle$	$\langle$*chair*, {hatrack}$\rangle$	$\langle$*chair*, {easel}$\rangle$	

Figure 6.2: Four rounds of CONSENSUSVOTE for the string *chair* for four training samples. For each sample, we specify the set of lexemes generated in the Round 1 column, and update this set after each round. At the end, the highest voted new lexeme according to the final votes is returned. In this example, MAXVOTE and CONSENSUSVOTE lead to different outcomes. MAXVOTE, based on the initial sets only, will select $\langle$*chair*, {easel}$\rangle$.

6.3 Global Voting for Lexicon Learning

Our goal is to learn compact and accurate CCG lexicons. To this end, we globally reason about adding new entries to the lexicon by *voting* (VOTE, Algorithm 3, line 7), and remove entries by *pruning* the ones no longer required for explaining the training data (Algorithm 3, line 12). In voting, each sample can be considered as attempting to influence the learning algorithm to update the model lexicon with the entries required to parse it. In this section we describe two alternative voting strategies. Both strategies ensure that new entries are only added when they have wide support in the training data, but count this support in different ways. For reproducibility, we also provide step-by-step pseudocode for both methods in Appendix A.

Since we only have access to validation functions and treat derivations as latent, we consider as correct all derivations that are valid. Frequently, however, incorrect derivations spuriously lead to valid traces. Lexical entries extracted from such spurious derivations generalize poorly. The goal of voting is to eliminate such entries.

Voting is formulated on the factored lexicon representation, where each lexical entry is factored into a lexeme and a template, as described in Section 2.3. Each lexeme is a pair containing a natural language string and a set of logical constants.[2] A lexeme is combined with a template to create a lexical entry. In our lexicon learning approach only new lexemes are generated, while the set of templates is fixed; hence, our voting strategies reason over lexemes and only create complete lexicon entries at the end. Decisions are made for each string independently of all other strings, but considering all occurrences of that string in the training data.

In lines 4-5 of Algorithm 3 GENENTRIES is used to propose new lexical entries for each training sample. For each sample, a set λ_i, which includes all lexical entries participating in valid derivations, is generated. In these sets, the same string can appear in multiple lexemes. To normalize its influence, each sample is given a vote of 1.0 for each string, which is distributed uniformly among all lexemes containing the same string.

For example, a specific λ_i may consist of the following three lexemes: $\langle$*chair*, $\{\text{chair}\}\rangle$, $\langle$*chair*, $\{\text{hatrack}\}\rangle$, $\langle$*face*, $\{\text{post}, \text{front}, \text{you}\}\rangle$. In this set, the phrase *chair* has two possible meanings, which will therefore each receive a vote of 0.5, while the third lexeme will be given a vote of 1.0. Such ambiguity is common and occurs when the available supervision is insufficient to discriminate between different parses, for example, if they lead to identical executions.

Each of the two following strategies reasons over these votes to globally select the best lexemes. To avoid polluting the model lexicon, both strategies adopt a conservative approach and only select at most one lexeme for each string in each training iteration.

6.3.1 Strategy 1: MAXVOTE

The first strategy for selecting voted lexical entries is straightforward. For each string it simply aggregates all votes and selects the new lexeme with the most votes. A lexeme is considered new if it is not already in the model lexicon. If no such single lexeme exists (e.g., no new entries were used in valid derivations or in the case of a tie) no lexeme is selected in this iteration.

---

[2]Recall, for example, that in one lexeme the string *walk* may be paired with the set of constants $\{\text{move}, \text{direction}\}$.

A potential limitation of MAXVOTE is that the votes for all rejected lexemes are lost. However, it is often reasonable to re-allocate these votes to other lexemes. For example, consider the sets of lexemes for the word *chair* in the Round 1 column of Figure 6.2. Using MAXVOTE on these sets will select the lexeme $\langle$*chair*, $\{\text{easel}\}\rangle$, rather than the correct lexeme $\langle$*chair*, $\{\text{chair}\}\rangle$. This occurs when the samples supporting the correct lexeme distribute their votes over many spurious lexemes.

6.3.2 Strategy 2: CONSENSUSVOTE

Our second strategy CONSENSUSVOTE aims to capture the votes that are lost in MAXVOTE. Instead of discarding votes that do not go to the maximum scoring lexeme, voting is done in several rounds. In each round the lowest scoring lexeme is discarded and votes are re-assigned uniformly to the remaining lexemes. This procedure is continued until convergence. Finally, given the sets of lexemes in the last round, the votes are computed and the new lexeme with most votes is selected.

Figure 6.2 shows a complete voting process for four training datapoints. In each round, votes are aggregated over the four sets of lexemes, and the lexeme with the fewest votes is discarded. For each set of lexemes, the discarded lexeme is removed, unless it will lead to an empty set.[3] In the example, while $\langle$*chair*, $\{\text{easel}\}\rangle$ is discarded in Round 3, it remains in the set of x_4. The process converges in the fourth round, when there are no more lexemes to discard. The final sets include two entries: $\langle$*chair*, $\{\text{chair}\}\rangle$ and $\langle$*chair*, $\{\text{easel}\}\rangle$. By avoiding wasting votes on lexemes that have no chance of being selected, the more widely supported lexeme $\langle$*chair*, $\{\text{chair}\}\rangle$ receives the most votes, in contrast to Round 1, where $\langle$*chair*, $\{\text{easel}\}\rangle$ was the highest voted one.

6.4 Experimental Setup

To isolate the effect of our lexicon learning techniques we closely follow the experimental setup of Chapter 5. This includes evaluation metrics, the same beam-search CKY parser, two-pass parsing for testing, beam search for executing sequences of instructions and the same seed lexicon, weight

[3]This restriction is meant to ensure that discarding lexemes will not change the set of sentences that can be parsed. In addition, it means that the total amount of votes given to a string is invariant between rounds. Allowing for empty sets will change the sum of votes, and therefore decrease the number of samples contributing to the decision.

initialization and features. We also use the same data and train-test split, including both SAIL and ORACLE (Section 5.7). Except the optimization parameters specified below, we use the same parameter settings. We repeated each experiment five times and report mean precision, recall,[4] harmonic mean (F1) and lexicon size. For held-out test results we also report standard deviation. For the baseline online experiments we shuffled the training data between runs.

Systems We report two baselines. Our batch baseline uses the same regularized algorithm, but updates the lexicon by adding all entries without voting and skips pruning. Additionally, we added post-hoc pruning to Algorithm 2 by discarding all learned entries that are not participating in max-scoring valid derivations at the end of training. For ablation, we study the influence of the two voting strategies and pruning, while keeping the same regularization setting. Finally, we compare our approach to previous published results on both corpora.

Optimization Parameters We optimized the learning parameters using cross validation on the training data to maximize recall of complete sequence execution and minimize lexicon size. We use 10 training iterations and the learning rate $\mu = 0.1$. For SAIL we set the regularization parameter $\gamma = 1.0$ and for ORACLE $\gamma = 0.5$.

Full Sequence Inference To execute sequences of instructions we use the beam search procedure presented in Section 5.5.2 with an identical beam size of 10. The beam stores states, and is initialized with the starting state. Instructions are executed in order, each is attempted from all states currently in the beam, the beam is then updated and pruned to keep the 10-best states. At the end, the best scoring state in the beam is returned.

6.5 Results

Table 6.1 shows ablation results for 5-fold cross-validation on the ORACLE training data. We evaluate against the online learning algorithm (Algorithm 2), an extension of it to include post-

[4]Recall is identical to accuracy as reported in prior work.

ORACLE corpus cross-validation	Single sentence			Sequence			Lexicon size
	P	R	F1	P	R	F1	
Artzi and Zettlemoyer (2013b)	84.59	82.74	83.65	68.35	58.95	63.26	5383
w/ post-hoc pruning	84.32	82.89	83.60	66.83	61.23	63.88	3104
Batch baseline	85.14	81.91	83.52	72.64	60.13	65.76	6323
w/ MAXVOTE	84.04	82.25	83.14	72.79	64.86	68.55	2588
w/ CONSENSUSVOTE	84.51	82.23	83.36	72.99	63.45	67.84	2446
w/ pruning	**85.58**	**83.51**	**84.53**	75.15	65.97	70.19	2791
w/ MAXVOTE + pruning	84.50	82.89	83.69	72.91	**66.40**	69.47	2186
w/ CONSENSUSVOTE + pruning	85.22	83.00	84.10	**75.65**	66.15	**70.55**	**2101**

Table 6.1: Ablation study using cross-validation on the ORACLE corpus training data. We report mean precision (P), recall (R) and harmonic mean (F1) of execution accuracy on single sentences and sequences of instructions and mean lexicon sizes. Bold numbers represent the best performing method on a given metric.

hoc pruning and a batch baseline. Our best sequence execution development result is obtained with CONSENSUSVOTE and pruning. The results provide a few insights. First, simply switching to batch learning provides mixed results: precision increases, but recall drops and the learned lexicon is larger. Second, adding pruning results in a much smaller lexicon, and, especially in batch learning, boosts performance. Adding voting further reduces the lexicon size and provides additional gains for sequence execution. Finally, while MAXVOTE and CONSENSUSVOTE give comparable performance on their own, CONSENSUSVOTE results in more precise and compact models when combined with pruning.

Table 6.2 lists our test results. We significantly outperform previous state of the art on both corpora when evaluating sequence accuracy. In both scenarios our lexicon is 60-70% smaller. In contrast to the development results, single sentence performance decreases slightly compared to Artzi and Zettlemoyer (2013b). The discrepancy between single sentence and sequence results might be due to the beam search performed when executing sequences of instructions. Models with more compact lexicons generate fewer logical forms for each sentence: we see a decrease of roughly 40% in model size compared to (Artzi and Zettlemoyer, 2013b). This is especially helpful during sequence execution, where we use a beam size of 10, resulting in better sequences of executions. In general, this shows the potential benefit of using more compact models in scenarios

	Final results	Single sentence			Sequence			Lexicon size
		P	R	F1	P	R	F1	
SAIL	Chen and Mooney (2011)		54.40			16.18		
	Chen (2012)		57.28			19.18		
	+ additional data		57.62			20.64		
	Kim and Mooney (2012)		57.22			20.17		
	Kim and Mooney (2013)		62.81			26.57		
	Artzi and Zettlemoyer (2013b)	**67.60**	**65.28**	**66.42**	38.06	31.93	34.72	10051
	Our Approach	66.67	64.36	65.49	**41.30**	**35.44**	**38.14**	**2873**
ORACLE	Artzi and Zettlemoyer (2013b)	**81.17** (0.68)	**78.63** (0.84)	**79.88** (0.76)	68.07 (2.72)	58.05 (3.12)	62.65 (2.91)	6213 (217)
	Our Approach	79.86 (0.50)	77.87 (0.41)	78.85 (0.45)	**76.05** (1.79)	**68.53** (1.76)	**72.10** (1.77)	**2365** (57)

Table 6.2: Our final results compared to previous work on the SAIL and ORACLE corpora. We report mean precision (P), recall (R), harmonic mean (F1) and lexicon size results and standard deviation between runs (in parenthesis) when appropriate. *Our Approach* stands for batch learning with a consensus voting and pruning. Bold numbers represent the best performing method on a given metric.

that incorporate reasoning about parsing uncertainty.

To illustrate the types of errors avoided with voting and pruning, Table 6.3 describes common error classes and shows example lexical entries for batch trained models with CONSENSUSVOTE and pruning and without. Quantitatively, the mean number of entries per phrase on development folds decreases from 16.77 for online training to 8.11.

Finally, the total computational cost of our approach is roughly equivalent to online approaches. In both approaches, each pass over the data makes the same number of inference calls, and in practice, Artzi and Zettlemoyer (2013b) used 6-8 iterations for online learning while batch learning used 10. A benefit of the batch method is its insensitivity to data ordering, as expressed by the lower standard deviation between randomized runs in Table 6.2.[5]

[5]Results still vary slightly due to multi-threading.

Phrase	# lexical entries		Example categories
	Batch baseline	With voting and pruning	
The algorithm often treats common bigrams as multiword phrases, and later learns the more general separate entries. Without pruning the initial entries remain in the lexicon and compete with the correct ones during inference.			
octagon carpet	45	0	~~$N : \lambda x.\text{wall}(x)$~~ ~~$N : \lambda x.\text{hall}(x)$~~ ~~$N : \lambda x.\text{honeycomb}(x)$~~
carpet	51	5	$N : \lambda x.\text{hall}(x)$ ~~$N/N : \lambda f.\lambda x.x == \text{argmin}(f, \lambda y.\text{dist}(y))$~~
octagon	21	5	$N : \lambda x.\text{honeycomb}(x)$ ~~$N : \lambda x.\text{cement}(x)$~~ $ADJ : \lambda x.\text{honeycomb}(x)$
We commonly see in the lexicon a long tail of erroneous entries, which compete with correctly learned ones. With voting and pruning we are often able to avoid such noisy entries. However, some noise still exists, e.g., the entry for "intersection".			
intersection	45	7	$N : \lambda x.\text{intersection}(x)$ ~~$S\backslash N : \lambda f.\text{intersect}(you, (f))$~~ $AP : \lambda a.len(a, 1)$ ~~$N/NP : \lambda x.\lambda y.\text{intersect}(y, x)$~~
twice	46	2	$AP : \lambda a.\text{len}(a, 2)$ ~~$AP : \lambda a.\text{pass}(a, \mathcal{A}(\lambda x.\text{empty}(x)))$~~ ~~$AP : \lambda a.\text{pass}(a, \mathcal{A}(\lambda x.\text{hall}(x)))$~~
stone	31	5	$ADJ : \lambda x.\text{stone}(x)$ ~~$ADJ : \lambda x.\text{brick}(x)$~~ ~~$ADJ : \lambda x.\text{honeycomb}(x)$~~ ~~$NP/N : \lambda f.\mathcal{A}(f)$~~
Not all concepts mentioned in the corpus are relevant to the task and some of these are not semantically modeled. However, the baseline learner doesn't make this distinction and induces many erroneous entries. With voting the model better handles such cases, either by pairing such words with semantically empty entries or learning no entries for them. During inference the system can then easily skip such words.			
now	28	0	~~$AP : \lambda a.\text{len}(a, 3)$~~ ~~$AP : \lambda a.\text{direction}(a, \text{forward})$~~
only	38	0	~~$N/NP : \lambda x.\lambda y.\text{intersect}(y, x)$~~ ~~$N/NP : \lambda x.\lambda y.\text{front}(y, x)$~~
here	31	8	~~$NP : \text{you}$~~ $S/S : \lambda x.x$ ~~$S\backslash N : \lambda f.\text{intersect}(\text{you}, \mathcal{A}(f))$~~
Without pruning the learner often over-splits multiword phrases and has no way to reverse such decisions.			
coat	25	0	~~$N : \lambda x.\text{intersection}(x)$~~ ~~$ADJ : \lambda x.\text{hatrack}(x)$~~
rack	45	0	~~$N : \lambda x.\text{hatrack}(x)$~~ ~~$N : \lambda x.\text{furniture}(x)$~~
coat rack	55	5	$N : \lambda x.\text{hatrack}(x)$ ~~$N : \lambda x.\text{wall}(x)$~~ ~~$N : \lambda x.\text{furniture}(x)$~~
Voting helps to avoid learning entries for rare words when the learning signal is highly ambiguous.			
orange	20	0	~~$N : \lambda x.\text{cement}(x)$~~ ~~$N : \lambda x.\text{grass}(x)$~~
pics of towers	26	0	~~$N\lambda x.\text{intersection}(x)$~~ ~~$N : \lambda x.\text{hall}(x)$~~

Table 6.3: Example entries from a learned ORACLE corpus lexicon using batch learning. For each phrase we report the number of lexical entries without voting (CONSENSUSVOTE) and pruning and with, and provide a few examples. Struck entries were successfully avoided when using voting and pruning.

Chapter 7

BROAD-COVERAGE CCG SEMANTIC PARSING WITH AMR

This chapter considers the problem of learning a broad-coverage semantic parser to map any English sentence to a logical form representation. We use the recently released AMR Bank for training, where samples are newswire sentences paired with their graph-based meaning representation. Learning the lexicon relies on a two-pass induction algorithm. Each sentence is first processed bottom-up with template-based lexical induction (Section 2.5). In addition, we propose to learn new templates using a new recursive variation of the splitting procedure introduced by Kwiatkowski et al. (2010). To accurately reason about non-compositional dependencies, for example as common in pronoun co-reference, we propose a two-stage model to first reason about meaning compositionally and then resolve distant dependencies. The two stages are scored jointly. Experiments on the AMR Bank demonstrate new state-of-the-art performance. This chapter is based on work in submission.

7.1 Introduction

Semantic parsers map sentences to formal representations of their meaning (Zelle and Mooney, 1996a; Zettlemoyer and Collins, 2005; Liang et al., 2011). Existing learning algorithms have primarily focused on building actionable meaning representations which can, for example, directly query a database (Liang et al., 2011; Kwiatkowski et al., 2013) or instruct a robotic agent (Chen, 2012; Artzi and Zettlemoyer, 2013b). However, due to their end-to-end nature, such models must be relearned for each new target application and have only been used to parse restricted styles of text, such as questions and imperatives.

Recently, AMR (Banarescu et al., 2013) was proposed as a general-purpose meaning representation language for broad-coverage text, and work is ongoing to study its use for variety of

applications such as machine translation (Jones et al., 2012) and summarization (Liu et al., 2015). The AMR meaning bank provides a large new corpus that, for the first time, enables us to study the problem of grammar induction for broad-coverage semantic parsing. However, it also presents significant challenges for existing algorithms, including much longer sentences, more complex syntactic phenomena and increased use of non-compositional semantics, such as within-sentence coreference. In this chapter, we introduce a new, scalable CCG induction approach that solves these challenges with a learned joint model of both compositional and non-compositional semantics, and achieves state-of-the-art performance on AMR Bank parsing.

We map sentences to AMR structures in a two-stage process (Section 7.3). First, we use CCG to construct lambda-calculus representations of the compositional aspects of AMR. CCG is designed to capture a wide range of linguistic phenomena, such as coordination and long-distance dependencies, and has been used extensively for semantic parsing. To use CCG for AMR parsing we define a simple encoding for AMRs in lambda calculus, for example, as seen with the logical form z and AMR a in Figure 7.1 for the sentence *Pyongyang officials denied their involvement*. However, using CCG to construct such logical forms requires a new mechanism for non-compositional reasoning, for example to model the long-range anaphoric dependency introduced by *their*.

To represent such dependencies while maintaining a relatively compact grammar, we follow Steedman's (2011) use of generalized Skolem terms, a mechanism to allow global distant references in lambda calculus (Section 2.6). We then allow the CCG derivation to mark when non-compositional reasoning is required with underspecified placeholders. For example, Figure 7.1 shows an underspecified logical form u that would be constructed by the grammar with the bolded placeholder ID indicating an unresolved anaphoric reference. These placeholders are resolved by a factor graph model that is defined over the output logical form and models which part of it they refer to, for example to find the referent for a pronoun.

Following most work in semantic parsing, we consider two learning challenges: grammar induction, which assigns meaning representations to words and phrases, and parameter estimation, where we learn a model for combining these pieces to analyze full sentences. We introduce a new CCG grammar induction algorithm which incorporates ideas from previous algorithms (Zettle-

moyer and Collins, 2005; Kwiatkowski et al., 2010) in a way that scales to the longer sentences and more varied syntactic constructions observed in newswire text. During lexical generation (Section 7.4.1), the algorithm first attempts to use a set of templates to hypothesize new lexical entries. It then attempts to combine bottom-up parsing with top-down recursive splitting to select the best entries and learn new templates for complex syntactic and semantic phenomena, which are re-used in later sentences to hypothesize new entries.

Finally, while previous algorithms (e.g., Zettlemoyer and Collins, 2005) have assumed the existence of a grammar that can parse nearly every sentence to update its parameters, this does not hold for AMR Bank. Due to sentence complexity and search errors during parsing, our state-of-the-art model cannot produce fully correct logical forms for a significant portion of the training data. To learn from as much of the data as possible and accelerate learning, we adopt an early update strategy that is able to generate effective updates from partially correct analyses (Section 7.4.2).

We evaluate performance on the publicly available AMR corpus (Banarescu et al., 2013). We show that our modeling and learning contributions are crucial for grammar induction at this scale and achieve new state-of-the-art results for AMR parsing (Section 7.6).[1] We also present a number of ablation studies, including, for the first time, removing surface-form alignment heuristics, which demonstrate the need for future work, especially to generalize to other languages.

7.2 Technical Overview

Task Let $\mathcal{X}$ be the set of all possible sentences and A the set of all AMR structures. Given a sentence $x \in \mathcal{X}$, we aim to generate an AMR $a \in A$. We use a small set of rules to define a simple, deterministic, and invertible conversion process between AMRs and lambda-calculus logical forms; roughly speaking, each AMR variable gets its own lambda term, which is scoped as low as possible, and each AMR role becomes a binary predicate applied to these variables. Figure 7.1 shows an example, and the full details are provided in the supplementary materials. Therefore, henceforth we discuss the task of mapping a sentence $x \in \mathcal{X}$ to a logical form $z \in \mathcal{Z}$,

[1] Our models, lexicons and code will be publicly released.

x: *Pyongyang officials denied their involvement.*

a:
$(d/\text{deny-01}$
:ARG0 $(p/\text{person}$
:ARG0-of $(h/\text{have-org-role-91}$
:ARG1 $(c/\text{city}$
: name $(n/\text{name}$:op1"Pyongyang"))
:ARG2$(o/\text{official})))$
:ARG1 $(i/\text{involve-01}$:ARG1 $p))$

u:
$\mathcal{A}_1(\lambda d.\text{deny-01}(d) \wedge$
$\text{ARG0}(d, \mathcal{A}_2(\lambda p.\text{person}(p) \wedge$
$\textbf{REL-of}(p, \mathcal{A}_3(\lambda h.\text{have-org-role-91}(h) \wedge$
$\text{ARG1}(h, \mathcal{A}_4(\lambda c.\text{city}(c) \wedge$
$\text{name}(c, \mathcal{A}_5(\lambda n.\text{name}(n) \wedge$
$\text{op1}(n, \text{PYONGYANG}))))) \wedge$
$\textbf{REL}(h, \mathcal{A}_6(\lambda o.\text{official}(o)))))) \wedge$
$\text{ARG1}(d, \mathcal{A}_7(\lambda i.\text{involve-01}(i) \wedge$
$\text{ARG1}(i, \mathcal{R}(\textbf{ID})))))$

z:
$\mathcal{A}_1(\lambda d.\text{deny-01}(d) \wedge$
$\text{ARG0}(d, \mathcal{A}_2(\lambda p.\text{person}(p) \wedge$
$\text{ARG0-of}(p, \mathcal{A}_3(\lambda h.\text{have-org-role-91}(h) \wedge$
$\text{ARG1}(h, \mathcal{A}_4(\lambda c.\text{city}(c) \wedge$
$\text{name}(c, \mathcal{A}_5(\lambda n.\text{name}(n) \wedge$
$\text{op1}(n, \text{PYONGYANG}))))) \wedge$
$\text{ARG2}(h, \mathcal{A}_6(\lambda o.\text{official}(o)))))) \wedge$
$\text{ARG1}(d, \mathcal{A}_7(\lambda i.\text{involve-01}(i) \wedge$
$\text{ARG1}(i, \mathcal{R}(2)))))$

Figure 7.1: A sentence (x) paired with its AMR (a), underspecified logical form (u) and fully specified logical form (z) representations.

where $\mathcal{Z}$ is the set of all logical forms. For example, in Figure 7.1 we would map the sentence x to the logical form z. We evaluate recovering the correct AMR using SMATCH (Cai and Knight, 2013) (Section 7.6).

Model Given a sentence x and lexicon Λ, we generate the set of possible derivations $\textsc{Gen}(x; \Lambda)$ using a two-stage process (Section 7.3). First, we use a weighted CCG to map x to an *underspecified logical form* u (Section 7.3.1), a logical form with placeholder constants for unresolved elements. For example, in the underspecified logical form u in Figure 7.1, the constants REL-of, REL and ID are placeholders. We then resolve these placeholders by defining a factor graph to find their optimal mapping and generate the final logical form z. In the figure, REL-of is mapped to ARG0-of, REL to ARG2 and ID to 2.

Learning We assume access to a training set of N examples $\{(x_i, z_i) : i = 1 \dots N\}$, each containing a sentence x_i and a logical form z_i. Our goal is to learn a CCG, which constitutes learning the lexicon and estimating the parameters of both the grammar and the factor graph. We define a learning procedure (Section 7.4) that alternates between expanding the lexicon and updating the parameters. Learning new lexical entries relies on a two-pass process that combines learning the meaning of words and new syntactic structures.

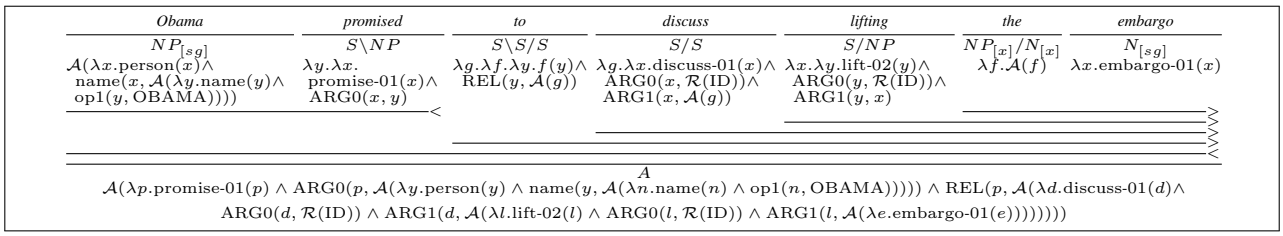
7.3 Mapping Sentences to Logical Form

Given a sentence x and lexicon Λ, the function $\textsc{Gen}(x, \Lambda) \subset \mathcal{D}$ defines the set of possible derivations, where $\mathcal{D}$ is the set of all derivations. Each derivation d is a tuple $\langle y, \mathcal{M} \rangle$, where y is a CCG parse tree and $\mathcal{M}$ is a mapping of constants from u, the *underspecified logical form* at the root of y, to their fully specified form.

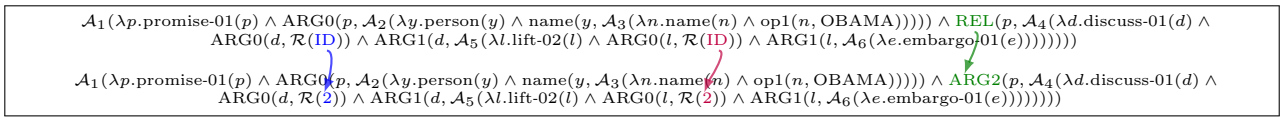
7.3.1 Underspecified Logical Forms

An underspecified logical form represents multiple logical forms via a mapping function that maps its constants to sets of constants and Skolem IDs, as defined in Section 2.6. For example, consider the underspecified logical form at the top of Figure 7.2b. If, for example, REL can be mapped to employed_by or ARG2, then the sub-expression $\text{REL}(h, \mathcal{A}_4(\lambda o.\text{discuss-01}(o)))$ represents $\text{employed\_by}(h, \mathcal{A}_4(\lambda o.\text{discuss-01}(o)))$ or $\text{ARG2}(h, \mathcal{A}_4(\lambda o.\text{discuss-01}(o)))$. In our exper-

(a) **CCG parse y:** Map the sentence x to an underspecified logical form (Section 7.3.1), which contains placeholders for unresolved decisions: ID for reference identifiers and the predicate REL for an unresolved relation.



(b) **Constant mapping $\mathcal{M}$:** All Skolem terms are assigned unique IDs and each constant in u, the logical form at the root of y, is mapped to a Skolem ID or a logical constant. Only mappings that modify constants are illustrated.



(c) **Final logical form:** A fully specified representation that can be deterministically converted to an AMR.

$$\mathcal{A}_1(\lambda p.\text{promise-01}(p) \wedge \text{ARG0}(p, \mathcal{A}_2(\lambda y.\text{person}(y) \wedge \text{name}(y, \mathcal{A}_3(\lambda n.\text{name}(n) \wedge \text{op1}(n, \text{OBAMA}))))) \wedge \text{ARG2}(p, \mathcal{A}_4(\lambda d.\text{discuss-01}(d) \wedge \text{ARG0}(d, \mathcal{R}(2)) \wedge \text{ARG1}(d, \mathcal{A}_5(\lambda l.\text{lift-02}(l) \wedge \text{ARG0}(l, \mathcal{R}(2)) \wedge \text{ARG1}(l, \mathcal{A}_6(\lambda e.\text{embargo-01}(e))))))))$$

Figure 7.2: A complete derivation for the sentence *Obama promised to discuss lifting the embargo*.

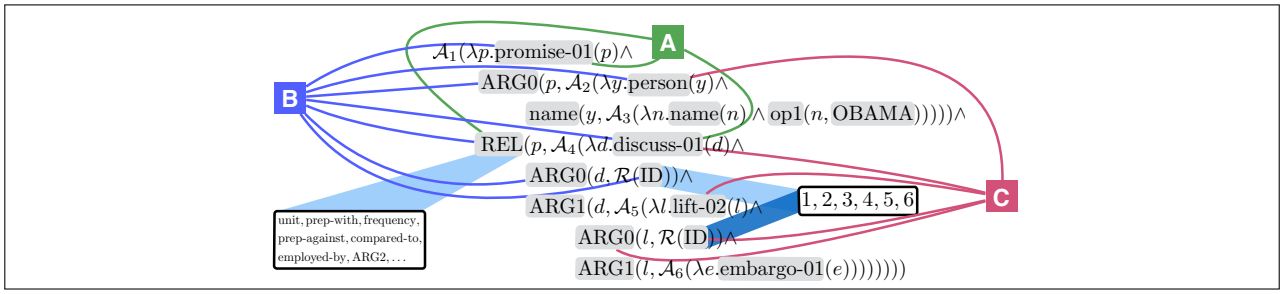


Figure 7.3: A visualization of the factor graph constructed for the derivation in Figure 7.2. Only a subset of the factors are included and the set of possible assignments is only specified for placeholders (for REL only a subset is listed).

iments, REL can be assigned one of 65 constants derived from the roles in AMR,[2] and, in the example, each of the ID placeholders can be assigned one of the 6 Skolem IDs.

Formally, let $\mathcal{C}$ be the set of all constants and $\mathcal{I}(u)$ the set of all Skolem IDs in the logical form u. Let $\mathcal{S}_u : \mathcal{C} \rightarrow 2^{\mathcal{C} \cup \mathcal{I}(u)}$ be a specification function. We call a constant c a *placeholder* if $|\mathcal{S}_u(c)| > 1$. Given an underspecified logical form u, applying $\mathcal{S}_u$ to all constants u contains generates a set of fully specified logical forms.

7.3.2 Derivations

The first part of a derivation $d = \langle y, \mathcal{M} \rangle$ is a CCG parse tree y with an underspecified logical form u at its root. For example, Figure 7.2a shows such a CCG parse tree, where the logical form contains the placeholders REL and ID. ID represents a distant reference, which is not captured by the CCG parse, and REL abstracts over the multiple meanings the preposition *to* may have.

The second part of the derivation is a function $\mathcal{M} : \textsc{Consts}(u') \rightarrow \mathcal{C} \cup \mathcal{I}(u')$, where u' is generated by assigning a unique ID to each Skolem term in u and $\textsc{Consts}(u')$ is the set of all occurrences of constants in u'. For example, in the figure, $\textsc{Consts}(u')$ contains, among others, three different occurrences of $\mathrm{ARG0}$ and two occurrences of ID. $\mathcal{M}$ then maps REL to $\mathrm{ARG2}$ and both occurrences of ID to the Skolem ID 2. The set of potential assignments for each occurrence of constant c is $\mathcal{S}_{u'}(c)$ and $\mathcal{M}$ is a disambiguation of $\mathcal{S}_{u'}$, which returns a single element for each constant rather than a set. The result of applying $\mathcal{M}$ to all constants in u' is the final logical form z (Figure 7.2c).

Decomposing the derivations provides two advantages. First, we are able to defer decisions from the CCG parse to the factor graph, for example by not resolving the exact meaning of the preposition *to*, thereby considering fewer hypotheses during parsing and simplifying the computation. Second, we can represent distant references while avoiding the complex parse trees that would have been required to represent these dependencies with scoped variables instead of Skolem IDs, which are globally scoped.

---

[2]The complete specification function we use is detailed in Appendix C.

7.3.3 Model

Given a sentence x, we use a weighted log-linear CCG to rank the space of possible parses under the grammar Λ. At the root of each CCG derivation is the underspecified logical form u, which is assigned unique Skolem IDs to generate u'.

For each u' we build a factor graph $G_{u'} = \langle V, F, E \rangle$, where $V = \textsc{Consts}(u')$ is the set of variables, F is the set of factors and E is the set of edges. Each edge is of the form (v, f) where $v \in V$ and $f \in F$. Figure 7.3 shows the factor graph used in generating the derivation in Figure 7.2, including all the variables and a subset of the factors. For each variable $v_c \in V$ such that $c \in \textsc{Consts}(u')$ the set of possible assignments is determined by $\mathcal{S}_{u'}(c)$. In the figure, the set of assignments to the ID placeholders is $\{1, 2, 3, 4, 5, 6\}$, the set of Skolem IDs in u'. REL may be assigned any of a large set of relations $\{\text{unit}, \text{prep-with}, \text{frequency}, \dots\}$.[2]

To generate the factors F and edges E we use the function $\Phi(V')$ that maps a set of variables $V' \subseteq V$ to a factor f and a set of edges, each one of the form (v, f), where $v \in V'$. Factors express various features (Section 7.5), such as selectional preferences and control structures. In the figure, the factor A captures the selectional preference for the assignment to the relation REL between promise-01 and discuss-01. Factor C captures a similar reference, this time to resolve the second ID placeholder to the Skolem ID 2, which has type person. Finally, factor B captures a control structure, as created between promise-01 and discuss-01, for resolving the first ID placeholder to the Skolem ID 2. Since the assignment of many of the variables is fixed, i.e., they are fully specified constants, in practice our factor graph representation simply conditions on them.

Derivations are scored using a log-linear model that includes both CCG parse features and those defined by the factor graph. Let $\mathcal{D}(z)$ be the subset of derivations with the final logical form z and $\theta \in \mathbb{R}^l$ be a l-dimensional parameter vector. We define the probability of the logical form z:

$$p(z|x; \theta, \Lambda) = \sum_{d \in \mathcal{D}(z)} p(d|x; \theta, \Lambda) \quad ,$$

and the probability of a derivation d is defined:

$$p(d|x;\theta,\Lambda) = \frac{e^{\theta\cdot\phi(x,d)}}{\sum_{d'\in \textsc{Gen}(x;\Lambda)} e^{\theta\cdot\phi(x,d')}} \quad , \tag{7.1}$$

where $\phi(x,d) \in \mathbb{R}^l$ is a feature vector (Section 7.5).

7.3.4 Inference

To compute the set of derivations $\textsc{Gen}(x;\Lambda)$ we define a two-stage process. We first run the CCG parser (Section 7.3.3) to generate underspecified logical forms. Following previous work (Zettlemoyer and Collins, 2005), we use CKY parsing to enumerate the top-K underspecified logical forms. We dynamically generate lexical entries for numbers and dates using regular expression patterns and for named-entities using a recognizer. For every underspecified logical form u, we construct a factor graph and use beam search to find the top-L configurations of the graph.[3]

7.4 Learning

Learning the two-stage model requires inducing the entries of the CCG lexicon Λ and estimating the parameters θ, which score both stages of the derivation. We assume access to a training set of N examples $D = \{(x_i, z_i), i = 1 \ldots N\}$, each containing a sentence x_i and a logical form z_i. This data does not include information about the lexical entries and CCG parsing operations required to construct the correct derivations. We consider all these decisions as latent. We first describe the main learning algorithm (Algorithm 6) and then its lexical learning subroutine (Algorithm 7) and gradient computation procedure.

The main learning algorithm (Algorithm 6) starts by initializing the lexicon (line 1) and then processes the data T times (line 2), each time alternating between batch expansion of the lexicon and a sequence of mini-batch parameter updates. An iteration starts with a batch pass to expand the lexicon. The subroutine GENENTRIES, described in Section 7.4.1 and Algorithm 7, is called

[3]Experiments with loopy belief propagation showed it to be slower and less effective for our task.

Algorithm 6 The main learning algorithm.

Input: Training set $D = \{(x_i, z_i) : i = 1 \dots N\}$, number of iterations T, mini-batch size M, seed lexicon Λ_0 and learning rate μ.

Definitions: $\textsc{Sub}(D, i, j)$ is the set of the next j samples from D starting at i. $\textsc{GenMax}(x, z, \theta, \Lambda)$ is the set of viterbi derivations from x with the final result z given parameters θ and lexicon Λ. $\textsc{Lex}(d)$ is the set of lexical entries used in the derivation d. $\textsc{ComputeGrad}(x, z, \theta, \Lambda)$ computes the gradient for sentence x and logical form z, given the parameters θ and lexicon Λ, and it described in Section 7.4.2. $\textsc{AdaGrad}(\Delta)$ applies a per-feature learning rate to the gradient Δ (Duchi et al., 2011).

Output: Lexicon Λ and model parameters θ.

1: $\Lambda \leftarrow \Lambda_0$
2: **for** $t = 1$ to T **do**
3: » Generate entries and update the lexicon.
4: **for** $i = 1$ to N **do**
5: $\lambda_{\text{new}} \leftarrow \lambda_{\text{new}} \cup \textsc{GenEntries}(x_i, z_i, \theta, \Lambda)$
6: $\Lambda \leftarrow \Lambda \cup \lambda_{\text{new}}$
7: » Compute and apply mini-batch gradient updates.
8: **for** $i = 1$ to $\lceil \frac{N}{M} \rceil$ **do**
9: $\Delta \leftarrow \vec{0}$
10: **for** (x, z) in $\textsc{Sub}(D, i, M)$ **do**
11: » Compute and aggregate the gradient.
12: $\Delta \leftarrow \Delta + \textsc{ComputeGrad}(x, z, \theta, \Lambda)$
13: $\theta \leftarrow \theta + \mu \textsc{AdaGrad}(\Delta)$
14: » Get all correct viterbi derivations.
15: $V \leftarrow \bigcup_{(x,z) \in D} \textsc{GenMax}(x, z, \theta, \Lambda)$
16: » Retain only entries from derivations in V.
17: $\Lambda \leftarrow \bigcup_{d \in V} \textsc{Lex}(d)$
18: **return** Λ and θ

to generate a set of new entries for each sample (line 5).

Next, we update the parameters θ with mini-batch updates. Given a mini-batch size of M, we use the procedure $\textsc{Sub}(D, i, M)$ to get the i-th segment of the data D of size M. We process this segment (line 10) to accumulate the mini-batch gradient Δ by calling the procedure $\textsc{ComputeGrad}(x, z, \theta, \Lambda)$ (line 12), which computes the gradient for x and z given θ and Λ, as described in Section 7.4.2. We use AdaGrad (Duchi et al., 2011) parameter updates (line 13).

Each iteration concludes with a pass over the data to remove all lexical entries that do not

Algorithm 7 GENENTRIES: Procedure to generate lexical entries from one training sample. See Section 7.4.1 for details.

Input: Sample (x, z), model parameters θ and lexicon Λ.
Definitions: $\textsc{GenLex}(x, z, \Lambda)$ and
$\textsc{RecSplit}(z, z, \theta, \Lambda)$ are defined in Section 7.4.1.
Output: Set of lexical entries λ.

```
1: » Augment lexicon with sample-specific entries.
2: Λ+ ← Λ ∪ GENLEX(x, z, Λ)
3: » Get max-scoring correct derivations.
4: D+ ← GENMAX(x, z, Λ+, θ)
5: if |D+| > 0 then
6:     » Return entries from max-scoring derivations.
7:     return ⋃_{d∈D+} LEX(d)
8: else
9:     » Top-down splitting to generate new entries.
10:    return RECSPLIT(x, z, θ, Λ+)
```

appear in max-scoring correct derivations, to correct for overgeneration (lines 15-17).

7.4.1 Lexicon Expansion: GENENTRIES

Given a sentence x, a logical form z, parameters θ and a lexicon Λ, $\textsc{GenEntries}(x, z, \theta, \Lambda)$ (Algorithm 7) computes a set of lexical entries, such that there exists at least one derivation d using these entries from x to z. We first use $\textsc{GenLex}(x, z, \Lambda)$ to generate a large set of potential lexical entries by generating lexemes[4] and pairing them with all templates in Λ. We then use a two-pass process to select the entries to return. The set of generated lexemes is a union of: (a) the set G_{gen} that includes all pairings of subsets of constants from z with spans in x up to length k_{gen} and (b) the set that is constructed by identifying named-entity constants[5] in z and their corresponding mentions in the text via case-insensitive matching to create new lexemes with potentially any other

[4]Lexeme and templates are used to generate lexical entries. For example, consider the lexical entry $\textit{lifting} \vdash S/NP : \lambda x.\lambda y.\text{lift-02}(y) \wedge \text{ARG0}(y, \mathcal{R}(ID)) \wedge \text{ARG1}(y, x)$, which can be generated by pairing the lexeme $\langle \textit{lifting}, \{\text{lift-02}\}\rangle$ with the template $\lambda v_1.[S/NP : \lambda x.\lambda y.v_1(a) \wedge \text{ARG0}(y, \mathcal{R}(\text{ID})) \wedge \text{ARG1}(y, x)]$. Our lexicon includes lexemes and templates and our algorithm learns both. See the Section 2.3 for more details.

[5]Named-entity constants are created from name instances when converting from AMR to lambda calculus (Appendix B).

constant (for lexemes with multiple constants). Λ is augmented with the generated set of lexical entries to create Λ_+ (line 2).

First Pass Given the augmented lexicon Λ_+, we compute the set $\mathcal{D}_+ = \textsc{GenMax}(x, z, \theta, \Lambda_+)$ (line 4). Similarly to Section 5.6, we constrain the set of derivations to include only those that use at most one lexeme from G_{gen}. If generating new lexemes is sufficient to derive z from x, $\mathcal{D}_+$ will contain these derivations and we return their lexical entries to be added to the lexicon Λ (lines 5-7). Otherwise, we proceed to do a second pass, where we try to generate new lexical templates to parse the sentence.

Second Pass: RecSplit In this pass we try to generate max-scoring derivations in a top-down process. Starting from u, the underspecified form of z, we search for CCG parsing steps that will connect to existing partial derivations in the CKY chart to create a complete parse tree. Since the space of possible operations is extremely large, we use automatically generated CCGBank (Hockenmaier and Steedman, 2007) categories to prune the categories we consider, as described below.

The second pass is executed by calling $\textsc{RecSplit}(x, z, \theta, \Lambda_+)$, which returns a set of lexical entries to add to the model (line 10). We recursively apply the splitting operation introduced by Kwiatkowski et al. (2010). Given a CCG category, splitting outputs all possible category pairs that could have originally generated it. For example, given the category

$$S \vdash \lambda x.\text{lift-02}(x) \wedge \text{ARG0}(x, \mathcal{R}(ID)) \wedge \text{ARG1}(x, \mathcal{A}(\lambda y.\text{embargo-01}(y))) \ ,$$

one of the possible splits will include the categories

$$S/NP \vdash \lambda y.\lambda x.\text{lift-02}(x) \wedge \text{ARG0}(x, \mathcal{R}(ID)) \wedge \text{ARG1}(x, y) \ \text{ and}$$
$$NP \vdash \mathcal{A}(\lambda y.\text{embargo-01}(y)) \ ,$$

which would combine with the operation $>$ (forward application). Kwiatkowski et al. (2010) present the full details. The process starts from u, the underspecified form of z, and recursively

applies the splitting operation while ensuring that: (1) there is at most one entry from G_{gen} or one entry where both the template and lexemes are new in the derivation, (2) each parsing step must have at least one child that may be constructed from an existing partial derivation, and (3) for each new parsing step, the syntax of a newly generated child must match the syntax of an automatically generated CCGBank category for the same span. To search the space of derivations we populate a CKY chart with all partial derivations and then do a top-down beam search, where in each step we split categories for smaller spans.

7.4.2 Gradient Computation: COMPUTEGRAD

Given a sentence x, its labeled logical form z, parameters θ and lexicon Λ, we compute the gradient for the sample (x, z) with the procedure COMPUTEGRAD(x, z, θ, Λ). Let $\mathcal{D}^*(z) =$ GENMAX(x, z, θ, Λ), the set of max-scoring correct derivations. The *hard* gradient update is:

$$\frac{1}{|\mathcal{D}^*(z)|} \sum_{d \in \mathcal{D}^*(z)} \phi(x_i, d) - E_{p(d,|x_i;\theta,\Lambda)}[\phi(x_i, d)] \quad , \tag{7.2}$$

where $\phi(x, d) \in \mathbb{R}^l$ is a l-dimensional feature vector (Section 7.3.3) and the positive portion of the gradient, rather than using expected features, averages over all max-scoring correct derivations.

Early updates To generate an effective update when no correct derivation is observed, we follow Collins and Roark (2004) and do an early update if $\mathcal{D}^*(z)$ is empty or if GEN$(x; \Lambda)$, the set of derivations for x, does not contain a derivation with the correct final logical form z.

Given the partial derivations, our gradient computation is identical to Equation 7.2. However, in contrast to Collins and Roark (2004) our data does not include gold derivations. Therefore, we attempt to identify max-scoring partial derivations that may lead to the correct derivation. We extract sub-expressions from u,[6] the underspecified form generated from the annotated z, and search the CKY chart for the top-scoring non-overlapping spans that contain categories with these logical forms. We use the partial derivations leading to these cells to compute the gradient.

[6]We extract all sub-expressions of type e, $\langle e, t\rangle$, $\langle e, t\rangle$, $\langle e, t\rangle\rangle$ or $\langle e, \langle e, t\rangle\rangle$ from u.

The benefit of early updates is two-fold. First, as expected, it leads to higher quality updates that are focused on the errors the model makes. Second, given the complexity of the data, it allows us to have updates for many examples that would be otherwise ignored. In our experiments, we observe this behavior with nearly 40% of the training set.

7.5 Experimental Setup

Data, Tools and Metric For evaluation, we use AMR Bank release 1.0 (LDC2014T12). We use the proxy report portion, which includes newswire articles from the English Gigaword corpus, and follow the official split for training (6603 sentences), development (826 sentences) and evaluation (823 sentences). We use EasyCCG (Lewis and Steedman, 2014) trained with the re-banked CCGBank (Hockenmaier and Steedman, 2007; Honnibal et al., 2010) for getting CCGBank spans, the Illinois Named Entity Tagger (Ratinov and Roth, 2009) for NER and Stanford CoreNLP (Manning et al., 2014) for tokenization. We report the SMATCH metric (Cai and Knight, 2013) for evaluating logical forms converted back to AMRs.

Features During CCG parsing, we use features that indicate the presence of unary type shifting, crossing composition and dynamically generated lexical entries, lexemes and templates. We also use indicators for co-occurrence of part-of-speech tags and syntactic attributes, repetitions in logical conjunctions and attachment of entities (via an AMR relation). In the factor graph, we use features for control structures, parent-relation-child selectional preferences and for mapping a relation to its final form. We lexicalize all relation names with the words that introduced them for computing attachment and factor graph features.

Initialization and Parameters We create the seed lexicon by annotating 50 random sentences with their lexical entries and adding entries for pronouns. We initialize the seed lexeme features and any entry generated by named-entity matching (Section 7.4.1) to 10. Following Kwiatkowski et al. (2011), we use GIZA++ scores to initialize lexeme features for all other entries. We also add to the seed lexicon all possible lexemes for all alignments generated by JAMR (Flanigan et al.,

2014) for the training data. We set the number of iterations $T = 10$ and select the best model based on development results. We set the max number of tokens for lexical generation $k_{\text{gen}} = 2$, learning rate $\mu = 0.1$, CCG parsing beam $K = 50$, factor graph beam $L = 100$, mini batch size $M = 40$ and use a beam of 100 when computing GENMAX. We initialize unary type-shifting and crossing-composition features to -3 and set a weight of 3 to features that pair singular and plural part-of-speech tags with singular and plural syntactic attributes (Section 2.2). All other weights are initialized to 0.

Two-pass Inference During testing, we perform two passes of inference for every sentence. First, we run our inference procedure (Section 7.3.4). If no derivations are generated, we run inference again, allowing the parser to skip words at a fixed cost and use the entries for *related* words if a word is unknown. We find related words in the lexicon using case, plurality and inflection string transformations. Finally, if necessary, we heuristically transform the logical forms at the root of the CCG parse trees to valid AMR logical forms. We set the cost of logical form transformation and word skipping to 10 and the cost of using related entries to 5.

7.6 Results

Table 7.1 shows SMATCH test results. We compare our approach to the latest, *fixed* version of JAMR (Flanigan et al., 2014) available online,[7] the only system to report test results on the official LDC release. Our approach outperforms JAMR by 3 SMATCH F1 points, with a significant gain in recall. Given consensus inter-annotator agreement of 83 SMATCH F1 (Flanigan et al., 2014), this improvement narrows the gap between automated methods and human performance by 15%. Although not strictly comparable, Table 7.1 also includes results on the pre-release AMR Bank corpus, including the published JAMR results, their most recent fixed results and the results of Wang et al. (2015).

Table 7.2 shows SMATCH scores for the developments set, with ablations. We also provide example outputs from our system for sentences from the development set. Figures 7.4 and 7.5 show

[7]JAMR is available at `http://bit.ly/1Jc3IyS`.

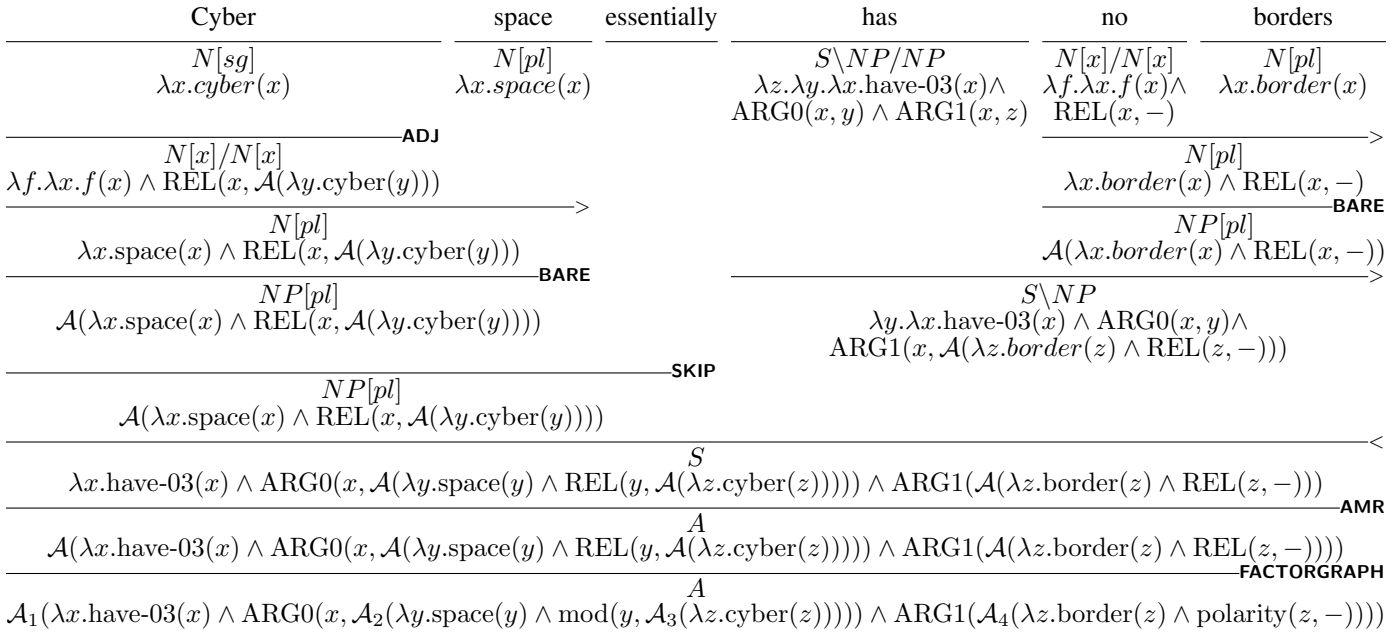
	P	R	F1
JAMR (fixed)	67.8	59.2	63.2
Our approach	66.8	65.7	66.3
Pre-release corpus results			
JAMR (Flanigan et al., 2014)	52.0	66.0	58.0
JAMR (fixed)	66.8	58.3	62.3
Wang et al. (2015)	64.0	62.0	63.0

Table 7.1: Test SMATCH results.

	P	R	F1
Full system	67.2	65.1	66.1
w/o underspecified constants	66.9	64.2	65.5
Lexical learning ablations			
w/o splitting	65.0	65.0	65.0
w/o G_{gen}	62.6	62.7	62.6
w/o surface-form similarity	55.9	38.5	45.6

Table 7.2: Development SMATCH results.

the max-scoring derivations and AMRs from our learned model, along with gold-label AMRs. We first remove underspecifying constants, which leaves the factor graph to resolve only references. While the expressivity of the model remains the same, more decisions are considered during parsing, modestly impacting performance.



(a) Derivation of the final logical form.

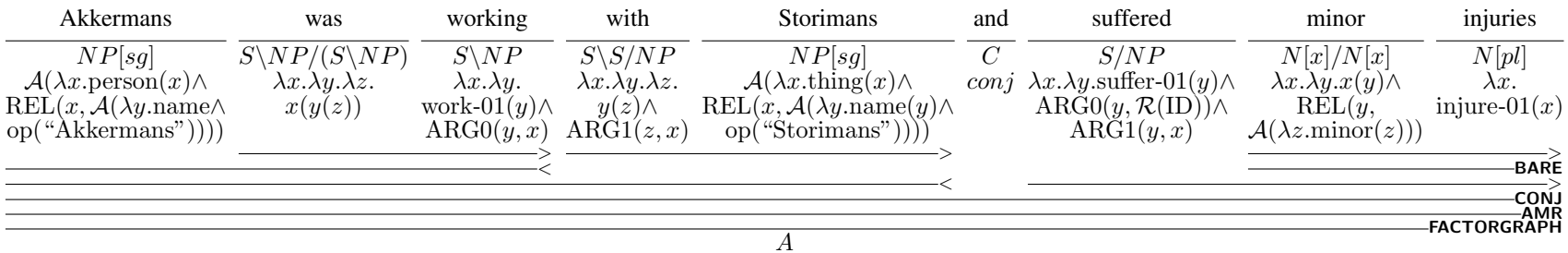
```
(h/have-03
  :ARG0 (s/space
    :mod (c/cyber))
  :ARG1 (b/border
    :polarity −)
```

(b) Predicted AMR.

```
(h/have-03
  :ARG0 (s/space
    :mod (c/cyber))
  :ARG1 (b/border)
  :polarity −
  :manner (e/essential))
```

(c) Gold-label AMR.

Figure 7.4: Results from parsing the sentence: *Cyber space essentially has no borders.* The system failed to recover the manner of have-03, and it incorrectly applied the negation to border rather than to have-03. The SMATCH F1 for this example is 0.80.



(a) Derivation of the final logical form. The intermediate categories are omitted for brevity.

```
(a/and
  :op1 (w/work
    :ARG0 (p/person
      :name (n/name :op1 "Akkermans")
    :ARG1 (t/thing
      :name (n2/name :op1 "Storimans")
  :op2 (i/suffer-01
    :ARG0 p
    :ARG1 (i/injury-01
      :polarity(m/minor)))
```

(b) Predicted AMR from our system.

```
(a/and
  :op1 (w/work
    :ARG0 (p/person
      :name (n/name :op1 "Akkermans")
    :ARG3 (p2/person
      :name (n2/name :op1 "Storimans")
  :op2 (i/injure-01
    :ARG1 p
    :degree (m/minor)))
```

(c) Gold-label AMR.

Figure 7.5: Results from parsing the sentence: *Akkermans was working with Storimans and suffered minor injuries*. The system correctly analyzed the coordination, but it incorrectly introduced a suffer-01 predicate and inferred that *Storimans* fills the role of a job rather than a coworker. The SMATCH F1 for this example is 0.70.

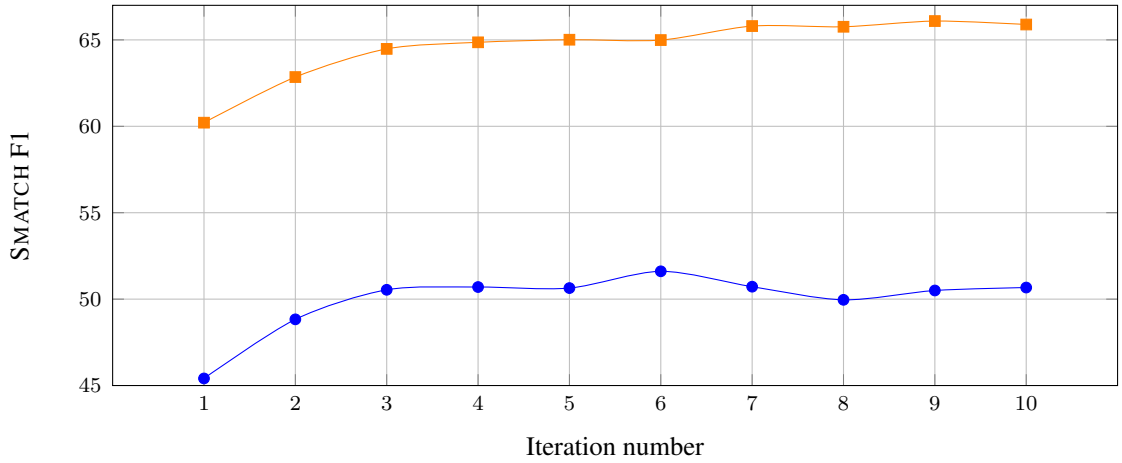


Figure 7.6: Development SMATCH F1 without early updates (●) and with early updates (■).

We also study the different methods for lexical generation. Skipping the second recursive splitting pass in GENENTRIES creates an interesting tradeoff. As we are unable to learn templates without splitting, we induce a significantly smaller lexicon (500K vs. 1.6M entries). Although we are unable to recover many syntactic constructions, our search problem is in general much simpler. We therefore see a relatively mild drop in overall performance (1.1 F1). Removing G_{gen} during lexical generation (Section 7.4.1) creates a more significant drop in performance (3.4 F1), demonstrating how considering all possible lexemes allows the system to recover entries that are not covered by heuristic alignments. We are also able for the first time to report AMR parsing results without any surface-form similarity features. This includes removing both JAMR alignments and named-entity matching lexical generation. The significant drop in performance (20 points F1) demonstrates the need for further study of alignment algorithms that do not depend on surface-form similarity, which is critical for learning AMR parsers for other languages.

Finally, Figure 7.6 plots development SMATCH F1 with and without early updates. As expected, early updates increase the learning rate significantly and have a large impact on overall performance. Without early updates we are unable to learn from almost half of the data, and performance drops by nearly 15 points.

Chapter 8

CONCLUSION

We presented algorithms for learning to map sentences to logical form meaning representations. We focused on CCG induction techniques that (1) require no linguistic annotation, (2) consider situated signals and (3) are able to model non-compositional semantics.

In Chapter 4, we presented a loss-driven learning approach that induces the lexicon and parameters of a CCG parser for mapping sentences to logical forms. The loss was defined over the conversational context, without requiring annotation of user utterance meaning. The overall approach assumes that, in aggregate, conversations contain sufficient signal (remediations such as clarification, etc.) to learn effectively. In our evaluation, we satisfied this requirement by using logs from automated systems that deployed reasonably effective recovery strategies.

In Chapter 5, we showed how to do grounded learning of a CCG semantic parser that includes a joint model of meaning and context for executing natural language instructions. The joint nature allows situated cues to directly influence parsing and also enables algorithms that learn while executing instructions. This style of algorithm, especially when using the weaker end state validation, is closely related to reinforcement learning approaches (Branavan et al., 2009, 2010). However, we differ on optimization and objective function, where we aim for minimal loss. We expect many RL techniques to be useful to scale to more complex environments, including sampling actions and using an exploration strategy. We also designed a semantic representation to closely match the linguistic structure of instructional language, combining ideas from multiple semantic theories, including, for example, Neo-Davidsonian events (Parsons, 1990). This approach allowed us to learn a compact and executable grammar that generalized well.

Chapter 6 considered the problem of learning compact CCG lexicons, and presented voting and pruning methods based on corpus-level statistics for lexicon learning. We incorporated these

techniques into a batch modification of our online learning approach for joint lexicon induction and parameter estimation. Our evaluation demonstrates that both voting and pruning contribute towards learning a compact lexicon and illustrates the effect of lexicon quality on system performance.

In Chapter 7, we described an approach for broad-coverage CCG induction for semantic parsing, including a joint representation of compositional and non-compositional semantics, a new grammar induction technique and a procedure to generate early updates when we are unable to recover a correct derivation. We used AMR as the target representation and presented new state-of-the-art AMR parsing results.

Finally, the research described in this thesis was accompanied by the development and released of the University of Washington Semantic Parsing Framework (SPF; Artzi and Zettlemoyer, 2013a), which has been used for research beyond the scope of this thesis.

8.1 Future Work

There are multiple potential directions for future work, including integration with dialog systems, extensions to other languages and linguistic phenomena and designing faster parsing algorithms.

Autonomous and continuous language learning in interactive systems For natural language interfaces to be common, their deployment must require minimal engineering effort. Learning models for such interfaces must then rely on easily gathered non-expert data. Fortunately, as we showed in Chapter 4, interactions provide ample opportunities for learning. However, continuous improvement through interactions remains an open problem. One possibility is to bootstrap language understanding from system failures and subsequent recovery efforts. In each iteration of the system, it will interact with users to collect conversational logs, which will be used to learn a better grammar. To initialize the process, various methods can be used, including for example, using a small amount of supervised data for learning. Such an iterative process will allow natural language systems to improve through interaction with users.

Supporting more languages Language coverage is also paramount to supporting a wide variety of systems and user populations. For example, Hebrew, Arabic and Hungarian display a much richer morphology than English, which means that the word form is built from a base root augmented with additional information, such as possessiveness, gender and case. To successfully support such languages, we would need to develop more expressive theories of the lexicon, the grammar component which stores word meaning. Our use of templates in GENLEX is also a barrier for applying our approaches to other languages, as each language will require to re-design the templates from scratch. While the splitting technique we discuss in Chapter 7 is able to learn new templates, it requires annotated logical forms. Learning without manually designed templates and no linguistic supervision remains an open challenge. Solving it will allow a wider application of our techniques to other languages and domains.

Real time parsing of speech to formal meaning representation To live to its full potential, natural language interfaces must robustly support speech input. Therefore, another direction to make our approach more applicable to such interaction is to design algorithms that reason over speech and accurately recover formal representations of spoken language. One important direction is to develop fast incremental parsing algorithms, which will be able to interpret speech in real time. Since spoken language is often ungrammatical, it is important to design techniques that are robust to disfluencies and are able to relax grammar constraints when needed. As modern speech recognizers rely on shallow language models to prune candidates during recognition, we can also study replacing these shallow models with more expressive semantic parsers. This will potentially allow recognizers to consider high-level semantic information and decrease recognition error rate.

Appendix A

PSEUDOCODE FOR VOTING PROCEDURES

In this appendix we provide the step-by-step pseudo-code for the voting strategies described in Section 6.3. We first define in Section A.1 the function AGGREGATEVOTES to aggregate votes over lexemes (for review of lexemes and factored lexicons, see Section 2.3). We then describe MAXVOTE in Section A.1.1 and CONSENSUSVOTE in Section A.1.2. While voting is defined over lexemes, as described in Section 6.3, the output of each voting strategy algorithm is a set of lexical entries to add to the lexicon, as expected for the procedure VOTE in line 7 of Algorithm 3.

A.1 Vote Aggregation for Lexemes

Given sets of lexemes, votes are aggregated independently for each of the phrases included in the lexemes. Let $L_1, \ldots, L_N$ be a collection of N sets of lexemes, and let $L_\cup = \bigcup_{i=1}^{N} L_i$ be their union. Let $S(.)$ be a function that returns the phrase(s) for a single lexeme (set of lexemes). Therefore, $S(L_\cup)$ denotes the set of all phrases for the N lexeme sets. Given the above definitions, AGGREGATEVOTES$(L_1, \ldots, L_N)$ computes a voting dictionary V with an entry V_s for each phrase $s \in S(L_\cup)$. Each entry V_s is a function $L_\cup \to \mathbb{R}$, which given a lexeme $\ell \in L_\cup$ returns:

$$V_s(\ell) = \sum_{i=1}^{N} \frac{\mathbb{1}\{\ell \in L^{(i)}\}}{\sum_{\ell' \in L_\cup} \mathbb{1}\{\ell' \in L^{(i)} \wedge S(\ell') = s\}} \quad .$$

Where $\mathbb{1}\{.\}$ is an indicator function. $V_s(\ell)$ is the vote assigned to the lexeme ℓ for the phrase s. Each L_i may contribute a total vote of 1.0 for each phrase, which is distributed uniformly among all lexemes containing this phrase in L_i. See Section 6.3 for example votes computation.

Algorithm 8 MAXVOTE: Voting strategy to select the highest voted lexical entries. See SectionA.1.1 for details.

Input: Global lexicon Λ and sample-specific sets of lexical entries $\{\lambda_1, \ldots, \lambda_N\}$.
Output: Voted set of lexicon entries Λ_{max}.

1: » Get the set of new lexemes from the N generated sets.
2: $\lambda_\cup \leftarrow \bigcup_{i=1}^{N} \lambda_i$, $L_{new} \leftarrow (\lambda_\cup) \backslash (\Lambda)$
3: » Compute voting dictionary (Section A.1).
4: $V \leftarrow$ AGGREGATEVOTES$((\lambda_1), \ldots, (\lambda_N))$
5: » For each phrase $s \in S(L_{new})$, add to L_{max} the new lexeme with the most votes.
6: $L_{max} \leftarrow \emptyset$
7: **for** each phrase $s \in S(L_{new})$ **do**
8: $\quad L_{max} \leftarrow L_{max} \cup \{\arg\max_{\ell \in L_{new}} V_s(\ell)\}$
9: » Select all the lexical entries with max voted lexemes.
10: $\Lambda_{max} \leftarrow \{e : e \in \lambda_\cup$ and $(e) \in L_{max}\}$
11: **return** Λ_{max}

A.1.1 Strategy 1: MAXVOTE

Algorithm 9 provides the pseudocode for the first voting strategy, as described in Section 6.3.1. We define the function $(.)$ to return the lexeme(s) for a lexical entry (set of lexical entries). We first create the set of all new lexemes L_{new} (line 2), which includes all generated lexemes that don't appear in the model lexicon. Next, the voting dictionary V is computed for the N lexeme sets using AGGREGATEVOTES (line 4). In lines 7-8, for each phrase $s \in S(L_{new})$, we find the new lexeme with most votes. In case of a tie, no lexeme is selected. Finally, in lines 10-11, the lexical entries corresponding to the selected lexemes are returned.

A.1.2 Strategy 2: CONSENSUSVOTE

Algorithm 9 provides the pseudocode for the second voting strategy, as described in Section 6.3.2. CONSENSUSVOTE iteratively discards lexemes and re-distributes their votes between the remaining ones (lines 2-15). This process continues until convergence, i.e., no more lexemes are discarded. To discard lexemes, first we iterate over all sample-specific lexeme sets L_i and the phrases s they contain (lines 4-8). For each s and L_i, we create the set L that contains all lexemes in L_i that are not in the discard set L_-. If L still contains lexemes with the pharse s, we replace L_i with it (lines 7-8). This condition is intended to ensure that discarding lexemes will not change the

Algorithm 9 CONSENSUSVOTE: Voting strategy to select lexical entries preferred by most datapoints through multiple rounds of voting. See Section A.1.2 for details.

Input: Global lexicon Λ and sample-specific sets of lexical entries $\{\lambda_1, \ldots, \lambda_N\}$.
Output: Voted set of lexicon entries Λ_{con}.

1: $L_i \leftarrow (\lambda_i), L_- \leftarrow \emptyset$
2: **repeat**
3: » For each i, get the set of lexemes that have not been discarded.
4: **for** $i = 1$ to $N, \forall s \in S(L_i)$ **do**
5: $L \leftarrow \{\ell : \ell \in L^{(i)}$ and either $S(\ell) \neq s$ or $\ell \notin L_-)\}$
6: » Replace original set, only if s is still covered.
7: **if** $s \in S(L)$ **then**
8: $L^{(i)} \leftarrow L$
9: » Compute voting dictionary (Section A.1).
10: $V \leftarrow$ AGGREGATEVOTES$(L_1, \ldots, L_N)$
11: $L_\cup \leftarrow \bigcup_{i=1}^{N} L_i$
12: » Update the set of discarded lexemes L_- with minimally voted lexemes for each phrase.
13: **for** each phrase $s \in S(L_\cup)$ **do**
14: $L_- \leftarrow L_- \cup \{\ell : \ell \in L_\cup$ and $V_s(\ell) = \min_{\ell' \in L_\cup} V_s(\ell')\}$
15: **until** L_- does not change
16: » Update lexicon entry sets per datapoint.
17: **for** $i = 1$ to N **do**
18: $\lambda_i \leftarrow \{e : e \in \lambda_i$ and $(e) \in L_i\}$
19: $\Lambda_{con} \leftarrow$ MAXVOTE$(\Lambda, \{\lambda_1, \ldots, \lambda_N\})$
20: **return** Λ_{con}

set of sentences that can be parsed. After this update, each L_i covers the same set of phrase, but possibly contains fewer lexemes. Next, we update the set of discarded lexemes L_-. First, we use AGGREGATEVOTE to compute the updated voting dictionary (line 10) and collect all lexemes into a single set (line 11). To update L_- we iterate over all phrases (lines 13-14), and for each pharse s add the lexemes with the lowest number of votes to L_-. This process (lines 2-15) repeats until no more entries are discarded. In lines 17-18, the sample-specific lexical entry sets are reassigned based on the remaining lexemes in each L_i, and finally, we call MAXVOTE to get the max-voted lexical entries containing the undiscarded lexemes.

Appendix B

AMR TO LAMBDA CALCULUS CONVERSION

We define a deterministic and invertible conversion process between AMR and lambda-calculus representations. Table B.1 describes one direction (AMR to lambda calculus) of the recursive CONVERT process. In practice, we implemented both conversion directions and are able to accurately convert in both directions. We distinguish among name instance types and all other instance types. For name we concatenate the name tokens and create a conjunction for all other relations. For all other relations, we simply create conjunctions between binary literals and recursively convert the related instance. For variables re-used for reference, we replace them with a reference and the appropriate Skolem ID. Finally, for all other symbols, we create e-typed constants.

Our simple and deterministic conversion process makes no distinction between references that are the result of syntactic control and other types of references. For example, the structures for *Pyongyang officials denied their involvement* and *Pyongyang officials denied involvement* will be identical, although their syntactic derivation is different. Therefore, our model, inference procedure and learned grammar do not make this linguistic distinction. While a more linguistically motivated approach to control structures is likely to increase the accuracy of resolving such dependencies, it will require treating the distinction between such cases and other references as latent variables, further complicating the learning process. We, therefore, consider this a promising direction for future work.

Description	AMR A	Lambda calculus: CONVERT(A)	Update IDMAP
Name instances	$(n$/name :op1 TOKEN1 :op2 TOKEN2 ... :opm TOKENm :relation1 $A1$:relation2 $A2$ ... :relationl $Al)$	$\mathcal{A}_k(\lambda n.\text{name}(n) \wedge$ $\text{op}(t, TOKEN1\_TOKEN2\_$ $\ldots\_TOKENm) \wedge$ $\text{relation1}(t, \text{CONVERT}(A1)) \wedge$ $\text{relation2}(t, \text{CONVERT}(A2)) \wedge$ $\cdots \wedge$ $\text{relationl}(t, \text{CONVERT}(l))$	IDMAP$[n] = k$
All other instances	$(t$/instance_type :relation1 $A1$:relation2 $A2$ ... :relationl $Al)$	$\mathcal{A}_k(\lambda t.\text{instance\_type}(t) \wedge$ $\text{relation1}(t, \text{CONVERT}(A1)) \wedge$ $\text{relation2}(t, \text{CONVERT}(A2)) \wedge$ $\cdots \wedge$ $\text{relationl}(t, \text{CONVERT}(Al))$	IDMAP$[t] = k$
Re-using a variable for reference	v	$\mathcal{R}(\text{IDMAP}[v])$	
Other AMR symbols	symbol	symbol	

Table B.1: AMR to lambda calculus conversion cases.

Appendix C

AMR SPECIFICATION FUNCTION $\mathcal{S}$

Table C.1 describes our specification function $\mathcal{S}$. For all cases not listed in the table, the function returns the singleton set with the input constant.

Input constant c	$\mathcal{S}_z(c)$	Notes
ID	$\mathcal{I}(z)$	$\mathcal{I}(z)$ is the set of Skolem IDs in z.
REL	unit, prep-with, frequency, prep-against, compared-to, employed-by, location, prep-amid, prep-by, part, medium, ord, subevent, purpose, mode, poss, prep-except, example, prep-toward, prep-on, extent, prep-into, prep-to, prep-per, mod, name, age, prep-without, prep-under, degree, value, scale, time, cause, prep-within, prep-between, manner, prep-in-addition-to, domain, path, beneficiary, duration, prep-in, source, condition, prep-among, prep-from, destination, accompanier, topic, direction, concession, prep-considering, quant, polarity, li, range, prep-at, instrument, prep-as, ARG3, ARG2, ARG5, ARG4, prep-for	Active relations
REL-of	example-of, manner-of, duration-of, quant-of, time-of, consist-of, prep-of, extent-of, location-of, condition-of, cause-of, name-of, frequency-of, part-of, prep-instead-of, source-of, poss-of, subevent-of, instrument-of, purpose-of, destination-of, topic-of, degree-of, concession-of, domain-of, medium-of, subset-of, path-of, prep-on-behalf-of, ARG3-of, ARG4-of, ARG2-of, ARG1-of, ARG0-of	Passive relations

Table C.1: The specification function $\mathcal{S}$ used in our experiments.

Bibliography

Allen, J., Manshadi, M., Dzikovska, M., and Swift, M. (2007). Deep linguistic processing for spoken dialogue systems. In *Proceedings of the Workshop on Deep Linguistic Processing*.

Andreas, J., Vlachos, A., and Clark, S. (2013). Semantic parsing as machine translation. In *Proceedings of the annual meeting on Association for Computational Linguistics*.

Androutsopoulos, I., Ritchie, G. D., and Thanisch, P. (1995). Natural language interfaces to databases–an introduction. *Natural language engineering*, 1.

Artzi, Y. and Zettlemoyer, L. S. (2011). Bootstrapping semantic parsers from conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Artzi, Y. and Zettlemoyer, L. S. (2013a). UW SPF: The University of Washington Semantic Parsing Framework.

Artzi, Y. and Zettlemoyer, L. S. (2013b). Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1(1):49–62.

Baldridge, J. (2002). *Lexically specified derivational control in combinatory categorial grammar*. PhD thesis.

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. In *Proceedings of the Linguistic Annotation Workshop*.

Barwise, J. and Cooper, R. (1981). Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4(2):159–219.

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic parsing on freebase from question-answer pairs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Berant, J. and Liang, P. (2014). Semantic parsing via paraphrasing. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Bisk, Y., Christodoulopoulos, C., and Hockenmaier, J. (2015). Labeled grammar induction with minimal supervision. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Bisk, Y. and Hockenmaier, J. (2013). An hdp model for inducing combinatory categorial grammars. *Transactions of the Association for Computational Linguistics*, 1.

Bisk, Y. and Hockenmaier, J. (2015). Probing the linguistic strengths and limitations of unsupervised grammar induction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD international conference on Management of data*.

Boole, G. (1854). *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities*. Dover Publications.

Bos, J. (2008). Wide-coverage semantic analysis with boxer. In *Proceedings of the Conference on Semantics in Text Processing*.

Bozsahin, C. (1998). Deriving the predicate-argument structure for a free word order language. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and International Conference on Computational Linguistics*. Association for Computational Linguistics.

Branavan, S., Chen, H., Zettlemoyer, L. S., and Barzilay, R. (2009). Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*.

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

Bugmann, G., Klein, E., Lauria, S., and Kyriacou, T. (2004). Corpus-based robotics: A route instruction example. In *Proceedings of Intelligent Autonomous Systems*.

Cai, Q. and Yates, A. (2013a). Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Cai, Q. and Yates, A. (2013b). Semantic parsing freebase: Towards open-domain semantic parsing. In *Proceedings of the Joint Conference on Lexical and Computational Semantics*.

Cai, S. and Knight, K. (2013). Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the Conference of the Association of Computational Linguistics*.

Carbonell, J. G. and Hayes, P. J. (1983). Recovery strategies for parsing extragrammatical language. *Computational Linguistics*, 9.

Carpenter, B. (1997). *Type-Logical Semantics*. The MIT Press.

Chen, D. (2012). Fast online lexicon learning for grounded language acquisition. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Chen, D. L. and Mooney, R. J. (2011). Learning to interpret natural language navigation instructions from observations. In *Proceedings of the National Conference on Artificial Intelligence*.

Choi, E., Kwiatkowski, T., and Zettlemoyer, L. S. (2015). Scalable semantic parsing with partial ontologies. In *Proceedings of the Association of Computational Linguistics*.

Church, A. (1932). A set of postulates for the foundation of logic. *The Annals of Mathematics*, 33:346–366.

Church, A. (1940). A formulation of the simple theory of types. *The journal of symbolic logic*, 5:56–68.

Clark, S. and Curran, J. R. (2007a). Perceptron training for a wide-coverage lexicalized-grammar parser. In *Proceedings of the Workshop on Deep Linguistic Processing*.

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

Clarke, J., Goldwasser, D., Chang, M., and Roth, D. (2010). Driving semantic parsing from the world's response. In *Proceedings of the Conference on Computational Natural Language Learning*.

Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*.

Dahl, D. A., Bates, M., Brown, M., Fisher, W., Hunicke-Smith, K., Pallett, D., Pao, C., Rudnicky, A., and Shriberg, E. (1994). Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Proceedings of the workshop on Human Language Technology*.

Davidson, D. (1967). The logical form of action sentences. *Essays on actions and events*, pages 105–148.

Davidson, D. (1969). The individuation of events. In *Essays in honor of Carl G. Hempel*, pages 216–234. Springer.

Di Eugenio, B. and White, M. (1992). On the interpretation of natural language instructions. In *Proceedings of the Conference of the Association of Computational Linguistics*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, pages 2121–2159.

Dzifcak, J., Scheutz, M., Baral, C., and Schermerhorn, P. (2009). What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Eisner, J. (1996). Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the annual meeting on Association for Computational Linguistics*.

Epstein, M., Papineni, K., Roukos, S., Ward, T., and Della Pietra, S. (1996). Statistical natural language understanding using hidden clumpings. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*.

Flanigan, J., Thomson, S., Carbonell, J., Dyer, C., and Smith, N. A. (2014). A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of the Conference of the Association of Computational Linguistics*.

Garrette, D., Dyer, C., Baldridge, J., and Smith, N. A. (2015). Weakly-supervised grammar-informed bayesian CCG parser learning. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Ge, R. and Mooney, R. (2006). Discriminative reranking for semantic parsing. In *Porceedings of the Association for Computational Linguistics*.

Ge, R. and Mooney, R. J. (2005). A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Conference on Computational Natural Language Learning*.

Gildea, D. and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational linguistics*, 28.

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

Goldwasser, D. and Roth, D. (2011). Learning from natural instructions. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

He, Y. and Young, S. (2005). Semantic processing using the hidden vector state model. *Computer Speech and Language*, 19(1):85–106.

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

Heim, I. and Kratzer, A. (1998). *Semantics in Generative Grammar*. Blackwell Oxford.

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

Hockenmaier, J. and Bisk, Y. (2010). Normal-form parsing for combinatory categorial grammars with generalized composition and type-raising. In *Proceedings of the International Conference on Computational Linguistics*.

Hockenmaier, J. and Steedman, M. (2007). CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, pages 355–396.

Hoffman, B. (1995). The computational analysis of the syntax and interpretation of "free" word order in turkish. *IRCS Technical Reports Series*, page 130.

Hoffmann, R., Zhang, C., Ling, X., Zettlemoyer, L. S., and Weld, D. S. (2011). Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the Conference of the Association of Computational Linguistics*.

Honnibal, M., Curran, J. R., and Bos, J. (2010). Rebanking CCGBank for Improved NP Interpretation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Jones, B., Andreas, J., Bauer, D., Hermann, K. M., and Knight, K. (2012). Semantics-based machine translation with hyperedge replacement grammars. In *Proceedings of the International Conference on Computational Linguistics*.

Joshi, A. K., Shanker, K. V., and Weir, D. (1990). The convergence of mildly context-sensitive grammar formalisms. Technical report, Department of Computer & Information Science, University of Pennsylvania.

Kamp, H., Genabith, J., and Reyle, U. (2011). Discourse representation theory. In *Handbook of Philosophical Logic*, volume 15 of *Handbook of Philosophical Logic*. Springer.

Kamp, H. and Reyle, U. (1993). *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*. Springer.

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

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

Kim, J. and Mooney, R. J. (2012). Unsupervised pcfg induction for grounded language learning with highly ambiguous supervision. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Kim, J. and Mooney, R. J. (2013). Adapting discriminative reranking to grounded language learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., et al. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Kollar, T., Tellex, S., Roy, D., and Roy, N. (2010). Toward understanding natural language directions. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*.

Krishnamurthy, J. and Kollar, T. (2013). Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association for Computational Linguistics*, 1(1):193–206.

Krishnamurthy, J. and Mitchell, T. (2012). Weakly supervised training of semantic parsers. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Krishnamurthy, J. and Mitchell, T. M. (2014). Joint syntactic and semantic parsing with combinatory categorial grammar. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Krishnamurthy, J. and Mitchell, T. M. (2015). Learning a compositional semantics for freebase with an open predicate vocabulary. *Transactions of the Association for Computational Linguistics*, 3.

Kuhlmann, M. and Satta, G. (2014). A new parsing algorithm for combinatory categorial grammar. *Transactions of the Association for Computational Linguistics*, 2.

Kushman, N. and Barzilay, R. (2013). Using semantic unification to generate regular expressions from natural language. In *Proceedings of the Human Language Technology Conference of the North American Association for Computational Linguistics*.

Kwiatkowski, T., Choi, E., Artzi, Y., and Zettlemoyer, L. S. (2013). Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Kwiatkowski, T., Goldwater, S., Zettlemoyer, L. S., and Steedman, M. (2012). A probabilistic model of syntactic and semantic acquisition from child-directed utterances and their meanings. *Proceedings of the Conference of the European Chapter of the Association of Computational Linguistics*.

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

Kwiatkowski, T., Zettlemoyer, L. S., Goldwater, S., and Steedman, M. (2011). Lexical Generalization in CCG Grammar Induction for Semantic Parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Levin, E., Narayanan, S., Pieraccini, R., Biatov, K., Bocchieri, E., Di Fabbrizio, G., Eckert, W., Lee, S., Pokrovsky, A., Rahim, M. G., Ruscitti, P., and Walker, M. (2000). The at&t-darpa communicator mixed-initiative spoken dialog system. In *Proceedings of the International Conference on Spoken Language Processing*.

Lewis, D. (1970). General semantics. *Synthese*, 22(1):18–67.

Lewis, M. and Steedman, M. (2014). A* CCG parsing with a supertag-factored model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

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

Litman, D., Moore, J., Dzikovska, M., and Farrow, E. (2009). Using Natural Language Processing to Analyze Tutorial Dialogue Corpora Across Domains Modalities. In *Proceedings of the Conference on Artificial Intelligence in Education*.

Liu, F., Flanigan, J., Thomson, S., Sadeh, N., and Smith, N. A. (2015). Toward abstractive summarization using semantic representations. In *Proceedings of the North American Association for Computational Linguistics*.

Lu, W., Ng, H., Lee, W., 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*.

MacMahon, M. (2007). *Following Natural Language Route Instructions*. PhD thesis, University of Texas at Austin.

MacMahon, M., Stankiewics, B., and Kuipers, B. (2006). Walk the talk: Connecting language, knowledge, action in route instructions. In *Proceedings of the National Conference on Artificial Intelligence*.

Maienborn, C., Von Heusinger, K., and Portner, P. (2011). *Semantics: An international handbook of natural language and meaning*. Walter de Gruyter.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19.

Matuszek, C., FitzGerald, N., Zettlemoyer, L. S., Bo, L., and Fox, D. (2012). A joint model of language and perception for grounded attribute learning. In *Proceedings of the International Conference on Machine Learning*.

Matuszek, C., Fox, D., and Koscher, K. (2010). Following directions using statistical machine translation. In *Proceedings of the international conference on Human-robot interaction*.

Miller, S., Bobrow, R., Ingria, R., and Schwartz, R. (1994). Hidden understanding models of natural language. In *Proceedings of the Conference of the Association of Computational Linguistics*.

Miller, S., Stallard, D., Bobrow, R., and Schwartz, R. (1996). A fully statistical approach to natural language interfaces. In *Proceedings of the Conference of the Association of Computational Linguistics*.

Montague, R. (1970a). English as a Formal Language. In Visentini, B., editor, *Linguaggi Nella Societ*
'{a} e Nella Tecnica, pages 188–221. Edizioni di Communita.

Montague, R. (1970b). Universal grammar. *Theoria*, 36.

Montague, R. (1973). The proper treatment of quantification in ordinary english. *Approaches to natural language*, 49:221–242.

Ng, V. (2010). Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of the annual meeting of the association for computational linguistics*.

Och, F. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational linguistics*, pages 19–51.

Papineni, K. A., Roukos, S., and Ward, T. R. (1997). Feature-based language understanding. In *Proceedings of the European Conference on Speech Communication and Technology*.

Parsons, T. (1990). *Events in the Semantics of English*. The MIT Press.

Pasupat, P. and Liang, P. (2015). Compositional semantic parsing on semi-structured tables. In *Proceedings of the annual meeting on Association for Computational Linguistics*.

Poon, H. (2013). Grounded unsupervised semantic parsing. In *Proceedings of the annual meeting on Association for Computational Linguistics*.

Popescu, A.-M., Armanasu, A., Etzioni, O., Ko, D., and Yates, A. (2004). Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the international conference on Computational Linguistics*.

Pourdamghani, N., Gao, Y., Hermjakob, U., and Knight, K. (2014). Aligning english strings with abstract meaning representation graphs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Ramaswamy, G. and Kleindienst, J. (2000). Hierarchical feature-based translation for scalable natural language understanding. In *Proceedings of the International Conference on Spoken Language Processing*.

Ratinov, L. and Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the Conference on Computational Natural Language Learning*.

Riedel, S., Yao, L., McCallum, A., and Marlin, B. M. (2013). Relation extraction with matrix factorization and universal schemas.

Singh-Miller, N. and Collins, M. (2007). Trigger-based language modeling using a loss-sensitive perceptron algorithm. In *IEEE International Conference on Acoustics, Speech and Signal Processing*.

Steedman, M. (1996). *Surface Structure and Interpretation*. The MIT Press.

Steedman, M. (2000). *The Syntactic Process*. The MIT Press.

Steedman, M. (2011). *Taking Scope*. The MIT Press.

Steedman, M. and Baldridge, J. (2003). Combinatory categorial grammar. *Non-Transformational Syntax*, pages 181–224.

Tang, L. R. and Mooney, R. J. (2000). Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Tang, L. R. and Mooney, R. J. (2001). Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the European Conference on Machine Learning*.

Taskar, B., Guestrin, C., and Koller, D. (2003). Max-Margin Markov Networks. In *Proceedings of the Conference on Neural Information Processing Systems*.

Tellex, S., Kollar, T., Dickerson, S., Walter, M., Banerjee, A., Teller, S., and Roy, N. (2011). Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the National Conference on Artificial Intelligence*.

Thompson, C. and Mooney, R. J. (2003). Acquiring word-meaning mappings for natural language interfaces. *Journal of Artificial Intelligence Research*, 18(1):1–44.

Tur, G., Hakkani-Tur, D., and Heck, L. (2010). What is left to be understood in atis? In *Proceedings of the Spoken Language Technology Workshop*.

Vijay-Shanker, K. and Weir, D. J. (1990). Polynomial time parsing of combinatory categorial grammars. In *Proceedings of the annual meeting on Association for Computational Linguistics*.

Vogel, A. and Jurafsky, D. (2010). Learning to follow navigational directions. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Walker, M. and Passonneau, R. (2001). DATE: a dialogue act tagging scheme for evaluation of spoken dialogue systems. In *Proceedings of the First International Conference on Human Language Technology Research*.

Walker, M., Rudnicky, A., Prasad, R., Aberdeen, J., Bratt, E., Garofolo, J., Hastie, H., Le, A., Pellom, B., Potamianos, A., et al. (2002). Darpa communicator: Cross-system results for the 2001 evaluation. In *Proceedings of the International Conference on Spoken Language Processing*.

Wang, A., Kwiatkowski, T., and Zettlemoyer, L. S. (2014). Morpho-syntactic lexical generalization for CCG semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Wang, C., Xue, N., Pradhan, S., and Pradhan, S. (2015). A transition-based algorithm for amr parsing. In *Proceedings of the North American Association for Computational Linguistics*.

Ward, W. and Issar, S. (1994). Recent improvements in the cmu spoken language understanding system. In *Proceedings of the workshop on Human Language Technology*.

Webber, B., Badler, N., Di Eugenio, B., Geib, C., Levison, L., and Moore, M. (1995). Instructions, intentions and expectations. *Artificial Intelligence*, 73(1):253–269.

Wei, Y., Brunskill, E., Kollar, T., and Roy, N. (2009). Where to go: Interpreting natural directions using global inference. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Weir, D. (1988). *Characterizing mildly context-sensitive grammar formalisms*. PhD thesis.

Winograd, T. (1972). Understanding natural language. *Cognitive Psychology*, 3(1):1–191.

Wong, Y. and Mooney, R. J. (2006). Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Association for Computational Linguistics*.

Wong, Y. and Mooney, R. J. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Woods, W. A. (1968). Procedural semantics for a question-answering machine. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*.

Yatskar, M., Volkova, S., Celikyilmaz, A., Dolan, B., and Zettlemoyer, L. S. (2013). Learning to relate literal and sentimental descriptions of visual properties. In *Proceedings of the North American Association for Computational Linguistics*.

Young, S., Gasic, M., Keizer, S., Mairesse, F., Schatzmann, J., Thomson, B., and Yu, K. (2010). The hidden information state model: A practical framework for POMDP-based spoken dialogue management. *Computer Speech & Language*, 24(2):150–174.

Zelle, J. and Mooney, R. J. (1996a). Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*.

Zelle, J. M. (1995). *Using inductive logic programming to automate the construction of natural language parsers*. PhD thesis, University of Texas, Austin.

Zelle, J. M. and Mooney, R. J. (1993). Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*.

Zelle, J. M. and Mooney, R. J. (1994). Inducing deterministic prolog parsers from treebanks: A machine learning approach. In *Proceedings of the National Conference on Artificial Intelligence*.

Zelle, J. M. and Mooney, R. J. (1996b). Comparative results on using inductive logic programming for corpus-based parser construction. In *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*, pages 355–369. Springer.

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 *Proceedings 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 Association for Computational Linguistics and International Joint Conference on Natural Language Processing*.