Syntax-Mediated Semantic Parsing

Siva Reddy

Doctor of Philosophy

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

2017

Abstract

Querying a database to retrieve an answer, telling a robot to perform an action, or teaching a computer to play a game are tasks requiring communication with machines in a language interpretable by them. *Semantic parsing* is the task of converting human language to a machine interpretable language. While human languages are sequential in nature with latent structures, machine interpretable languages are formal with explicit structures. The computational linguistics community have created several treebanks to understand the formal syntactic structures of human languages. In this thesis, we use these to obtain formal meaning representations of languages, and learn computational models to convert these meaning representations to the target machine representation. Our goal is to evaluate if existing treebank syntactic representations are useful for semantic parsing.

Existing semantic parsing methods mainly learn domain-specific grammars which can parse human languages to machine representation directly. We deviate from this trend and make use of general-purpose syntactic grammar to help in semantic parsing. We use two syntactic representations: Combinatory Categorial Grammar (CCG) and dependency syntax. CCG has a well established theory on deriving meaning representations from its syntactic derivations. But there are no CCG treebanks for many languages since these are difficult to annotate. In contrast, dependencies are easy to annotate and have many treebanks. However, dependencies do not have a well established theory for deriving meaning representations. In this thesis, we propose novel theories for deriving meaning representations from dependencies.

Our evaluation task is question answering on a knowledge base. Given a question, our goal is to answer it on the knowledge base by converting the question to an executable query. We use Freebase, the knowledge source behind Google's search engine, as our knowledge base. Freebase contains millions of real world facts represented in a graphical format. Inspired from the Freebase structure, we formulate semantic parsing as a graph matching problem, i.e., given a natural language sentence, we convert it into a graph structure from the meaning representation obtained from syntax, and find the subgraph of Freebase that best matches the natural language graph.

Our experiments on Free917, WebQuestions and GraphQuestions semantic parsing datasets conclude that general-purpose syntax is more useful for semantic parsing than induced task-specific syntax and syntax-agnostic representations.

Lay Summary

Querying a database to retrieve an answer, telling a robot to perform an action, or teaching a computer to play a game are tasks requiring communication with machines in a language interpretable by them. *Semantic parsing* is the task of converting human language to machine interpretable language. Machine interpretable languages have a formal meaning representation. Whereas human language is sequential with hidden structures. The field of computational linguistics have created several treebanks to understand the syntactic structure of human language. In this thesis, we formulate formal ways to extract meaning representations from syntactic structure of language and learn computational models to convert these meaning representations to the target machine representation. In doing so, we will study whether existing treebank syntactic representations are useful for real world tasks.

Acknowledgements

Where should I start? More than a degree, this PhD has been a life-changing experience. I am grateful to many people during this journey and acknowledging everybody will be a book on itself which I will save for future. Until then ...

Supervisors I have two wonderful supervisors: Mirella Lapata, my primary, and Mark Steedman, my secondary. The day I started my PhD, Mirella gave me 30 papers on different topics. She told me to find the paper that fires a spark and that will decide my PhD topic. Now you know how I landed on semantic parsing. This spark is still alive. Meetings with Mirella are a lot of fun. If you never had one, I strongly recommend one. As a supervisor, she knew which research direction is worth pursuing, protected my time, kept me focused, met every week (even when she is traveling), gave detailed feedback on the drafts, and helped in writing beautifully. I wish one day I will be like her. As a person, she genuinely cared about my personal life and helped in whatever way she can. I will miss these meetings but I hope to continue working with her. Once I chose semantic parsing, Mark was the obvious person to go and find more about the topic. He was very welcoming and treated me just like his student spending a lot of time explaining CCG, and encouraging me to think big. Eventually he became my second supervisor. Even after all his success, he is modest and kind-hearted. Although I started as a novice, he treated me as a matured researcher and truly believed in me. This feeling is super-powerful. He is never tired of writing books. If I ever write a book, it is only because of the confidence he gave me. I will miss those intricate discussions on CCG, scope and weekly group meetings.

Google I was very fortunate to land an internship in Google Research Parsing team in New York, one of the most influential groups of NLP. Dipanjan Das discovered me in a Google workshop, and asked me to apply for an internship. If anyone from this team asks you for an internship, do yourself a favour by saying yes. This is where I met Oscar Täckström as my Google host, Michael (Mike) Collins as a co-intern (believe me), Tom Kwiatkowski as a new hire, Slav Petrov the manager you wish to have, and of course Fernando Pereira the godfather of Google's NLP. My first few weeks were tough with Oscar's code-reviewing comments longer than my code. Eventually I did push some code into Google codebase. Mike played a major role in coming up with a lambda calculus for dependencies. All of us including my supervisors kept refining this. It took almost an year to get it to a working form for English, and one more year

to generalize it to multiple languages. In the meantime, Google also awarded me a PhD fellowship allowing me to focus on the work. I am indebted to the Google parsing team, the PhD fellowship organization, and to the Google for giving back to the society. Oscar, thank you so much for being a close mentor and a friend, and Dipanjan for giving me this chance. I also had nice social life during internship, thanks to Greg Coppola, Karl Stratos, Nicholas FitzGerald, Yin-Wen Chang.

My Examiners I would like to thank Adam Lopez and Luke Zettlemoyer for their valuable time in assessing the thesis, and the critical feedback that helped improving this thesis. The amount of time Adam invests in students (not only his) is commendable. We had several discussions on the papers in this thesis when they are evolving, and his feedback on presentations for conferences and jobs were super useful. Luke's work on CCG semantic parsing has been highly influential in shaping up this thesis. Thank you for your extreme kindness when I visited UW. I will not forget it.

My Collaborators It was a great fun collaborating with several people on different projects beyond my thesis topic: Bharat Ambati, Yonatan Bisk, John Blitzer, Jianpeng Cheng, Shay Cohen, Rajarshi Das, Li Dong, Federico Fancellu, Julia Hockenmaier, Adam Lopez, Andrew McCallum, Maria Nădejde, Shashi Narayan, Srikanth Ronanki, Bonnie Webber, Kun Xu, Manzil Zaheer. I learned a lot from them, and most importantly I got to know them personally. These collaborations taught me research can be fun and need not be a one person show.

NLP Community It is on top of Jayant Krishnamurthy's code I built my first semantic parser. This thesis greatly benefited from Jonathan Berant and Percy Liang's WebQuestions and Sempre, Yoav Artzi and Luke Zettlemoyer's UW SPF, Mike Lewis's EasyCCG, the Stanford CoreNLP and the Universal Dependencies treebanks. I owe a lot to them. And to those anonymous reviewers (and sometimes bad ones) for assessing and improving my work. I was lucky to give talks at the Stanford NLP, UW NLP, UMass Amherst, Cambridge LTL, UvA ILLC, TU Darmstadt, MSR Redmond, AI2 and Nuance AI. Special thanks to these people who made me feel welcoming during the visits: Chris Manning, Percy, Chris Potts, Sebastian Schuster, Panupong Pasupat, Sida Wang, Luke, Noah Smith, Mike, Ioannis Konstas, Sameer Singh, Emily Bender, Andrew McCallum, Rajarshi Das, Nicholas Monath, Anna Korhonen, Diana McCarthy, Ivan Titov, Tejaswini Deoskar, Stella Frank, Desmond Elliot, Diego Marcheggiani, Michael

Schlichtkrull, Iryna Gurevych, Avinesh PVS, Daniil Sorokin, Scott Yih, Chris Quirk, Hoifung Poon, Kristina Toutanova, Lucy Vanderwende, Margaret Mitchell, Michel Galley, Matthew Richardson, Oren Etzioni, Jayant Krishnamurthy, Matt Gardner, Valeria de Paiva and Edward Stabler. One of these visits paved way to my postdoc with the world's leading expert of NLP, Chris Manning, and I can't wait to submit this thesis.

Edinburgh Group The ProbModels (now EdinburghNLP) meetings has taught me how "not" to present my work. There is an internal saying in the group that if you survive presenting here, you can present in sleep-mode anywhere else. I will fondly remember Mark's weekly group meetings and its members: Aciel, Andrew, Bharat, Christos, Greg, John, Mark (mini), Mike, Nathan, Omri, Teju. I will miss my officemates with whom I shared most of my day time, ups and downs, ranting about politics, global warming, saving animals, board games, numerous parties and of course beers to fuel up: Carina, Lea, Leimin, William, Hadi, Howard, Sam, Marco, Kathrin, and hall-of-fame members Akash, Janie, Stef, Philip, Uchenna. I will remember the good times with Ali, Annie, Cat, Chen, Clara, Daniel, Dave, Diego, Dominikus, Eva, Frank, Helena, Herman, Javad, Jen, Manick, Michael, Mihai, Miltos, Nate, Pippa, Ramya, Rico, Sharon, Silvia, Victor and Xingxing. I am bad at admin work, but Avril, Diana, Julie, Nicola, and Seona made it look so easy and at times stretching the possibilities, thank you for your patience. A special thanks to Caroline for working out the logistics of the Google fellowship and for the enhanced stipend. And to Margaret for cheering up every morning.

Family Out of some absurd luck, my PhD timeline overlapped with Bharat (my classmate since undergrad), Speech Siva and Praveen (our hometowns are only hours apart), and Srikanth (my undergrad junior). It was one happy family. Later additions to this are Rupa (Praveen's wife) and Spandana (my best half). These people made Edinburgh feel like home. There was no lack of entertainment, no topic we left undiscussed, and no dish we experimented. I will greatly miss them, luckily Bharat and I still live in the same city, and Spandana will eventually have no option :). I had a similar family when I lived in York with Matt, Sonia and Suraj, and we found many occasions to live that life whenever possible. Other people I consider as family in UK are Adam Kilgarriff and Diana McCarthy who cared about my well-being. For the first time, I felt the pain of losing a loved one when Adam passed away.

I have been an unconventional son to my parents (= amma, nanna, chinni, babai, mama, ammama, tata, mamayya, attaya). They have seen the worst of life, from rags to

middle class, and for them the point of education is to get a nice job. When everyone of my age group are getting real jobs, building houses and buying cars, I was studying and studying, and if I end up in an academic career, I don't see an end to it. Although it is painful for them to accept this reality (and sometimes to myself), they still encourage me to do what makes me happy, even at the cost of their happiness. My sisters and brother-in-laws took care of many of my duties towards my parents, supporting me during the good and bad times. Quite opposite to my parents is Spandana, the love of my life, who keeps up with my craziness but with reality checks once in a while. She stood beside me firmly even at times when we were challenged by the adverse stereotypical definitions of Indian marriages of what's possible and not. If it is not for her love, I cannot imagine having a happy ending to this PhD. And I dedicate this thesis to her.

Declaration

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

(*Siva Reddy*)

Table of Contents

Chapter 1

Introduction

Language is a unique ability of humans, one central to cognition and communication. The ease with which we generate language and the sophistication it offers us to express intense emotions to intricate philosophical arguments makes natural language an attractive medium to communicate with computing machinery. But computers do not understand human languages. They only understand programming languages, i.e., mathematical description of computation. This makes interaction with them difficult for a large fraction of population. Semantic parsing aims to address this problem by converting natural language to machine interpretable language. The practical implications of semantic parsing means devices that can interact with humans in natural language. These include robots, personal assistants, intelligent search engines, home appliances and intelligent cars.

Although humans are capable of writing machine interpretable language, these are hard to learn. They change with the change in the target application domain. Even for domain experts, machine terminology is difficult to remember requiring domain manuals. It would be convenient and less stressful to interact in natural language than in machine language. Computing machines also come in various shapes and sizes, e.g., laptops, tablets, smartphones and cars. Some of these keyboards are inconvenient to type. Relying on natural language allows one to dictate in natural speech via a speech recognition module. With the recent advances in speech recognition, natural language interaction with machines will likely be the most preferred interface.

Semantic parsing underlies several successful applications. Semantic parsing helped IBM Watson defeat humans in Jeopardy; Siri is a semantic parsing based personal assistant which serves million of Apple customers; Google answers a fair share of questions on knowledge graph using semantic parsing; and Tesla cars are enabled with natural

language interfaces. These applications are currently limited in the expressiveness of natural language but are a good starting point for more research in this area.

Since semantic parsing deals with many aspects of language, our goal is to evaluate theories of syntax and semantics using semantic parsing as a testbed. Specifically, we evaluate methods for converting CCG syntax (Steedman, 1996) and dependency syntax (Tesnière, 1959) to a formal semantic language. This formal language is in turn converted to machine interpretable language using a new semantic parsing framework.

1.1 History of Semantic Parsing

The earliest known semantic parsing system is BASEBALL (Green Jr et al., 1961), which answered questions such as *"On how many days in July did eight teams play?"* against a database containing American League statistics. Questions were converted to executable specification lists by matching the words and constituent phrases to database entities and predicates using a dictionary. The early success of semantic parsing can be attributed to two systems: LUNAR (Woods et al., 1972) and SHRDLU (Winograd, 1972). LUNAR answered natural language questions about the compositions of moon rocks and soil collected in the Apollo mission. The application was designed to make the database accessible to geologists across the US without having to learn the target programming language. Questions such as *"What samples contains Silicon?"* can be posed without the user being aware Silicon is represented as SIO2 in the database. In a public demonstration, the system answered 78% of the questions correctly, an impressive feat in 1970's when computers were slow, expensive, and inconvenient to work with. This system had almost all the components which current systems have: a syntactic parser to provide grammatical structure of the sentence, a rule-based semantic interpreter to convert syntactic fragments to target meaning representation, and an executor to execute the meanings on the database. Another impressive fact is that system was designed to handle quantifiers, something at which even modern semantic parsing systems fail.

SHRDLU is an intelligent agent that lived in a blocks world designed by Winograd (1972) during his PhD to understand block world instructions such as *"place the blue block on the block adjacent to the triangle"*. It was considered a landmark for successful demonstration of natural human-machine interaction. Until SHRDLU, the syntax of language was seen as an explanation for organizing strings without necessarily having to deal with semantics. SHRDLU's philosophy was to put the meaning of the sentences

first, and treat syntax an an organization of strings to convey this meaning. Sentence structure was treated as the result of series of choices made in order to generate this meaning. Another crucial philosophical difference from other systems was that semantics could not be defined on its own without reference to a domain. When we hear a sentence, we make full sense of it based on our knowledge about the topic under consideration. So for programs to understand language, they should consider world knowledge by only allowing those hypotheses that the world under consideration allows, e.g., in *"I rode down the street in a car"* a syntactic analysis that attaches car to the street should not be considered since the semantic realization does not allow the street to be contained in a car. A truly remarkable thing about SHRDLU was that it remembered past context and also interacted with the users when there was ambiguity asking for clarification. Its semantic processing component was a rule based system which mapped syntactic fragments to semantic fragments, and considered only those analyses that could be executed.

Around the same time, Montague (1973) proposed the marriage between lambda calculus and syntax to derive rich general purpose logical forms. This paved the way to methods which transform general purpose logical forms to domain-specific logical forms (Copestake & Jones, 1990), and also gave rise to many theories which derive logical forms from syntactic structures (Dalrymple et al., 1995; Steedman, 1996; Copestake et al., 2001; Gardent & Kallmeyer, 2003).

Statistical Semantic Parsing Although SHRDLU was a successful demonstration of semantic parsing, it soon became apparent that hand-coding rules is not scalable. Additionally, with the change in domains, these rules changed. Projects inspired from SHRDLU such as Cyc (Lenat et al., 1985) aiming to manually write down common sense knowledge such as *"dogs are pets, and pets are animals"* became increasingly complex as the number of rules increased. With the advent of statistical methods for NLP in 1990, many rule based methods became statistical. In this paradigm, data aligned with target representations is used to train machine learning models that learn the rules automatically. These methods also deal with ambiguity, ranking each possible interpretation.

Zelle & Mooney (1996) were the first to apply statistical learning methods to semantic parsing. They created a dataset containing questions paired with logical forms in a language called Geoquery. These queries can be executed on a database containing facts about US geography to retrieve answers. An example logical form is shown in

Question	What is the capital of the state with the largest population?
Logical Form	answer(C, (capital(S,C), largest(P, (state(S), population(S,P)))))

Figure 1.1: An example question paired with its logical form.

Figure 1.1. Their system, CHILL, uses inductive logic programming to learn a semantic parser in two phases. First, the training instances are used to learn an overly-general Prolog program for shift-reduce parsing to generate logical forms. Clauses in this program take the current states of the stack and buffer as input, and produce new states. The state configurations and transition actions represent the logical form. The initial parser produces many spurious analyses for a given sentence. In the second phase, the initial program is then specialized using positive and negative derivations from the previous step. The goal of this step is to generalize and impose additional constraints such that only positive examples can be generated by the initial program. A variety of follow-up systems capitalized on this work (Tang & Mooney, 2001; Thompson & Mooney, 2003; Kate et al., 2005; Ge & Mooney, 2005).

The next breakthrough came with Zettlemoyer & Collins (2005). At the time, most methods mainly used context-free grammar with projectivity assumptions, limiting the expressivity of meaning representations. Zettlemoyer & Collins used a probabilistic CCG grammar which is mildly context sensitive and can also deal with long-range dependencies and non-projective meaning representations. Due to the transparent syntax-semantic interface of CCG, they were able to learn both syntax and semantics simultaneously (See Chapter 3 for more details on CCG syntax-semantics interface). This was the first time where Montague (1973)'s paradigm of syntax being transparent to semantics was realized. Later on other grammar formalisms were applied to semantic parsing such as dependencies (Liang et al., 2011), and regular tree grammars (Jones et al., 2012).

Scaling Semantic Parsers Although semantic parsing methods became more robust with statistical methods, these methods assume access to sentences annotated with logical forms. Such data is expensive and difficult to annotate. Moreover knowledge bases became larger and larger with billions of facts (Suchanek et al., 2007; Bollacker et al., 2008). Creating supervised data for these domains became impossible.

So the next wave of innovations in semantic parsing were in scaling semantic parsing methods with alternative forms of supervision. Initial small scale attempts

Question	what are the major religions in Russia?
Answer	{Russian Orthodox Church, Islam}

Figure 1.2: An example question paired with its answer.

include learning from the world's response by providing linguistic input, e.g., a question and the desired output e.g., the answer to a question (Clarke et al., 2010; Liang et al., 2011). The first major boost came with Berant et al. (2013) who created a large semantic parsing dataset for Freebase (Bollacker et al., 2008), a large knowledge base. The main idea was that humans are good at answering questions rather than writing logical forms since answering a question does not require expertise in the the target application's language. Berant et al. collected answers to questions by showing mechanical turkers the corresponding Freebase pages in which the answer can be found. Figure 1.2 shows one such example.

Methods which rely on even weaker forms of supervisions such as domains paired with natural language descriptions (Artzi & Zettlemoyer, 2011; Krishnamurthy & Mitchell, 2012; Reddy et al., 2014; Choi et al., 2015; Bisk et al., 2016a) or paraphrases (Fader et al., 2013; Berant & Liang, 2014; Narayan et al., 2016) also came into existence.

Current research on semantic parsing focuses on designing novel ways to collect data (Wang et al., 2015b; Su et al., 2016), neural methods which make few assumptions about the structure of language (Dong & Lapata, 2016; Jia & Liang, 2016; Kočiský et al., 2016; Neelakantan et al., 2016), and alternate forms of knowledge sources as text, web tables, images (Krishnamurthy & Kollar, 2013; Pasupat & Liang, 2015; Andreas et al., 2016; Krishnamurthy & Mitchell, 2015; Xu et al., 2016; Gardner & Krishnamurthy, 2017).

1.2 Dimensions of Semantic Parsing

There are several design decisions that are important for engineering semantic parsing systems. We call these dimensions and explain how systems vary in order to understand their strengths and weaknesses.

Grounded World A primary decision is to choose the world in which the language has to be grounded, i.e., a domain in which the target language can be executed to take

an action (e.g. answer a question). The domain is often a knowledge base used for question answering (Banko et al., 2007; Suchanek et al., 2007; Bollacker et al., 2008; Carlson et al., 2010; Lehmann et al., 2015). If you are interacting with a robot, it would have its own programming language (MacMahon et al., 2006; Chen & Mooney, 2008; Matuszek et al., 2013; Bisk et al., 2016b). In both these scenarios, the target language is symbolic. In the first scenario, the world is static, i.e., the KBs will not change during execution time, but in the latter scenario, the world gets updated with each action taken, and correspondingly the target language becomes more complicated. For example, it contains dynamic variables that would have different values in different time intervals even when present in the same query (Artzi & Zettlemoyer, 2013).

In a few cases, the target language can be a continuous function. For example, if the goal is to answer questions based on an image, the target language could be attention distribution on the image, where attention indicates the region of interest (Andreas et al., 2016). Recently there has also been interest in non-executable but goal oriented semantic parsing (Krishnamurthy & Mitchell, 2015; Gardner & Krishnamurthy, 2017). These methods treat natural language as a distribution of grounded predicates and ungrounded (words in natural language) predicates.

Training data As mentioned above, the training data could consist of sentences paired with target meaning representations (Zelle & Mooney, 1996), or sentences paired with question-answer pairs (Clarke et al., 2010; Berant et al., 2013; Bisk et al., 2016a), or sentences paired with system behavior (Branavan et al., 2009; Chen & Mooney, 2011; Goldwasser & Roth, 2011) or just sentences from the domain (Goldwasser et al., 2011; Krishnamurthy & Mitchell, 2012; Poon, 2013; Reddy et al., 2014).

Lexical Mismatch Words in natural language often look different in surface form to predicates in the grounded world. Consider, the sentence *"what is Charles Darwin?"*, the intention here is to know the profession of Charles Darwin which is represented as *people.profession* in Freebase. None of the words in the original sentence has match this predicate. Methods dealing with this problem use a lexicon which maps words to knowledge base predicates either by handcoding them or learning them, (Zettlemoyer & Collins, 2005; Kwiatkowksi et al., 2010; Liang et al., 2011; Yao & Van Durme, 2014; Artzi et al., 2014; Krishnamurthy, 2016) or exploit lexical similarity between words in natural language with words in KB predicates (Kwiatkowski et al., 2013; Bast & Haussmann, 2015; Dong & Lapata, 2016).

Structural Mismatch On the surface, natural language is sequential in nature. Whereas target representations generally have tree or graph structures. Replacing words in the source with target language symbols does not often lead to the target representation. To cope with this problem, methods rely on the *principle of compositionality* (Pelletier, 1994). This principle says that the meaning of a phrase can be derived by combining the meaning of the words in the phrase. Usually each word is associated with a meaning in terms of the target language symbol and its interaction within the context. These word meanings are combined in a hierarchical fashion to form the meaning of larger units such as phrases and sentences. For example consider the phrase *largest state* and its target representation $\arg\max(\lambda x.\text{state}(x), \lambda x.\text{size}(x))$. This can be obtained by assigning words *largest* with lambda expression $\lambda f.\arg\max(\lambda x.f(x), \lambda x.\text{size}(x))$ and *state* with $\lambda x.\text{state}(x)$, and combining them together by applying *largest* to *state*. But how did we know to assign these expressions to corresponding words and combine them in the proposed order? This is where syntactic and semantic theories of language help.

Formalisms such as context free grammar and CCG have been used to derive semantic expressions from syntactic derivations (Zettlemoyer & Collins, 2005; Wong & Mooney, 2007; Kwiatkowksi et al., 2010; Liang et al., 2011; Kwiatkowski et al., 2013). The central idea of these methods is to learn an overly general grammar that parses language to multiple semantic structures, hoping that at least one of these matches the target representation. A learning model is trained such that the most likely match is ranked on the top compared to others. We will see more details in Chapter 2.

The principle of compositionality assumes that meaning is encoded in the sentence itself. However, the domain under consideration may also influence the meaning. For example, question *"what are the states?"* in Geoquery would actually mean *"what are the states in US?"*. Here *in US* is not present in the original sentence but is taken for granted because of the domain under consideration. To deal with such cases, additional operations on top of the grammar are proposed, e.g., a bridging operation that introduces arbitary predicates into an existing logical form (Berant et al., 2013), or a floating parser that introduces predicates with no explicit alignment to input text (Pasupat & Liang, 2015). In the absence of such operations, the grammar may be forced to learn *states* would mean *states in US*.

There are also methods which treat this structural mismatch without using grammars. These methods linearize the target representation and treat semantic parsing as a sequence-to-sequence transduction problem (Wong & Mooney, 2006; Andreas et al., 2013; Dong & Lapata, 2016; Jia & Liang, 2016).

1.3 Thesis Overview

In this thesis, we propose a semantic parser which converts natural language to Freebase executable language. Freebase is a large knowledge base curated manually. Semantic parsing on Freebase helps validating natural language statements for their truthfulness, and for question answering. We focus only on the latter part, however our methods can also be used for fact checking (Vlachos & Riedel, 2014) by grounding declarative statements to Freebase queries (Reddy et al., 2014).

Based on the discussion in Section 1.2, our problem has the following dimensions: Our grounded world is Freebase, i.e., we assume all natural language provided as input to our parser can be converted to Freebase queries. Our training data is in the form of question answer pairs as shown in Figure 1.2. Because of the scale of Freebase, we chose this relatively inexpensive source of supervision than using logical forms. Unlike existing methods which use a lexicon of words in language mapped to predicates in KB, we assume a predicate in language corresponds to a distribution of predicates in KB (see Section 2.10.5). In order to obtain predicates in natural language, we rely on *ungrounded* logical forms obtained from syntax. We call these ungrounded since the predicates are defined by words and not by Freebase. To handle structural mismatches, instead of converting language to target meaning representation directly, we convert ungrounded logical forms to Freebase query language exploiting the structural similarities between them. This factorizes the semantic parsing problem first as a linguistically motivated general purpose structured prediction problem and then a domain-specific transduction problem.

Our semantic parsing framework is as follows: Our input consists of question and answer pairs. Firstly questions are parsed to a syntactic representation using syntactic parsers. We use CCG (Steedman, 1996) and dependencies (Tesnière, 1959) as our syntactic representations. The CCG syntactic parse is converted to ungrounded logical forms using an existing CCG theory of semantics (Bos et al., 2004). We propose a new theory of semantics for converting dependencies to logical forms. We treat this step as a deterministic rule-based conversion. Inspired from the structure of Freebase, we convert ungrounded logical forms to natural language graphs. This enables us to explore semantic parsing as a graph matching problem, i.e., finding the best graph of Freebase that matches the natural language graph. This step involves a statistical model which learns to rank a Freebase graph that has the same denotation as the answer to a given question higher than other graphs.

The emphasis in this thesis is on using linguistic guidance for semantic parsing. For several decades, the field of linguistics created theories of language that explain the structure of the language. The computational linguistics community has created several treebanks for these theories by annotating sentences with linguistic structures. The NLP community developed many statistical methods to learn how to convert new sentences to linguistic structures learning from treebanks. A question that remains unanswered is how useful are these theories and resources for real world applications. Are existing theories sufficient or do we need a new theory? Semantic parsing is a good testbed to answer these questions. It is a real world application which requires explicit structures that are agnostic to our assumptions since these structures are grounded and can be evaluated for their correctness.

The guiding hypothesis in this thesis is that *general-purpose syntax helps in deriving superior task-agnostic and task-specific meaning representations than induced or latent syntax*. If our hypothesis is true, one can use task-specific training data just to learn to convert linguistic structures that are universal across applications and languages to task-specific structures. This paradigm requires fewer parameters than methods which try to learn everything (e.g., syntax, semantics) from the training data itself, wasting task-specific expensive resources for problems that are well addressed by language research community.

As our syntactic representations we choose CCG and dependencies. The trade-off's in chosing any syntactic formalism are the availability of resources and the expressivity of the formalism. Dependencies clearly win over CCG on the resources front, whereas in the later chapters, it will be evident that CCG wins on expressivity criteria. Although CCG wins on expressivity, semantic parsing datasets are often noisy with ungrammtical sentences. Our results show dependencies are more resilient to ungrammaticality than CCG, therefore leading to a better performance. For additional discussion on other available syntactic formalsims, see Section 2.3.

1.3.1 Problem Formulation

Our problem can be defined as follows: Given a question q that can be answered with a knowledge base $\mathcal{K}$, our goal is to find the meaning representation g such that the denotation of g is same as the true answer a. But due to database incompleteness or incorrect annotation of answers, it is not always possible to find an exact match. So, instead we redefine the goal as finding the representation g that has the highest overlap

with the annotated answer.

$$\hat{g}_q = \arg\max_g F_1([\![g]\!]_{\mathcal{K}}, a)$$

.

Here $[\![g]\!]_{\mathcal{K}}$ indicates the denotation (the return value) of g when executed on $\mathcal{K}$, i.e., the predicted answer. Here F_1 is the harmonic mean of precision and recall of predicted answer when compared with the annotated answer a.

1.3.2 Contributions

The contributions of this thesis are

1. We propose a new framework for semantic parsing. In this framework, we define semantic parsing as a graph matching problem as opposed to the traditional convention of treating it as a grammar learning problem.
2. We evaluate if treebank syntax is useful for semantic parsing. Traditionally, syntax is treated as latent or task-dependent. We assume syntax is task-independent and is a starting point for semantic parsing.
3. While CCG syntax has a well-studied semantic interface, a semantic interface for dependency syntax does not exist. Given that dependency treebanks are widely prevalent, such an interface would enable semantic parsing applications for multiple languages. We propose a new theory for extracting logical forms from dependency syntax.
4. Because of the universal nature of dependency structures, for the first time, we evaluate semantic interfaces for dependency syntax in multiple languages. We also create new datasets that allow us to evaluate Freebase semantic parsing for multiple languages.

1.4 Chapter Structure

The structure of the thesis is as follows.

Chapter 2 presents our semantic parsing framework. Our main idea is to treat semantic parsing as a graph matching problem. We describe the steps involved in the framework, and justify the design choices in comparison with existing work. This framework enables us to evaluate different syntactic representations for semantic parsing. We evaluate this framework on existing semantic parsing datasets: WebQuestions, Free917

and GraphQuestions in the later chapters. Our framework has also helped others (Lewis & Steedman, 2014; Bisk et al., 2016a; Narayan et al., 2016).

Chapter 3 evaluates the usefulness of treebank CCG syntax for semantic parsing. Existing methods induce domain-specific CCG syntax for semantic parsing making it unclear how useful is general-purpose syntax (Zettlemoyer & Collins, 2005, 2007; Kwiatkowksi et al., 2010; Kwiatkowski et al., 2013). To answer this question, we will use CCG syntax as defined in CCGBank (Hockenmaier & Steedman, 2007). This is first such evaluation of CCG for semantic parsing. Although the literature is rife with obtaining logical forms from CCG using lambda calculus, there is no work that describes the procedure to build lambda calculus expressions automatically from CCG syntactic categories (Bos et al., 2004; Lewis & Steedman, 2013; Vo et al., 2015; Abzianidze, 2015; Martínez-Gómez et al., 2016). We hope this chapter fills that gap.

Chapter 4 presents DEPLAMBDA a new theory for obtaining logical forms from Stanford dependencies. Dependencies lack syntactic type information on the words compared to CCG making it less transparent to semantics. However dependency labels provide a clue to the semantics of the words involved. Based on this intuition, we treat dependency labels as functions that drive the semantic composition instead of word-driven semantics as in CCG. We introduce a single-type system that assigns all words and constituent phrases with lambda expressions of a uniform semantic type. We found that dependency syntax is more robust than CCG syntax in handling grammatical errors which are problematic otherwise for semantic parsing.

Chapter 5 presents UDEPLAMBDA, a semantic interface for converting Universal Dependencies (UD; Nivre et al. 2016) to logical forms. UDEPLAMBDA is designed to work for any language which has a treebank in the UD schema with minimal reliance on language-specific information. While DEPLAMBDA works only with dependency tree inputs, UDEPLAMBDA is designed to work even with graph inputs, therefore enabling it to handle long-range dependency constructions such as control. To evaluate the usefulness of UDEPLAMBDA across languages for Freebase semantic parsing, we translate existing English semantic parsing datasets to Spanish and German, and present the first multilingual semantic parsing results on Freebase. Our results show UD is a promising resource for semantics in multiple languages.

Chapter 6 presents a higher-order type system for UDEPLAMBDA to generate semantics with quantified scope. This is not empirically evaluated yet. We also present discuss its limitations.

Finally, **Chapter 7** summarizes the thesis, findings, contributions, and discusses

directions for future work.

1.5 Published Work

The framework in Chapter 2 is published in Reddy et al. (2014), and the transduction operators are introduced in Reddy et al. (2016). Experiments in Chapter 3 are partly based on Reddy et al. (2014) and Reddy et al. (2016). DEPLAMBDA of Chapter 4 is published in Reddy et al. (2016), and so is UDEPLAMBDA of Chapter 5 in Reddy et al. (2017). Chapter 6 is the supplementary material to Reddy et al. (2017).

Chapter 2

Semantic Parsing Framework

In this chapter we introduce our framework for semantic parsing. Our knowledge base of interest is Freebase. Our key insight is to represent natural language as semantic graphs whose topology shares many commonalities with Freebase. Given this representation, we conceptualize semantic parsing as a graph matching problem. Our model converts sentences to semantic graphs with guidance from syntax and subsequently grounds them to Freebase using denotations as a form of supervision.

2.1 Freebase

Freebase consists of 42 million entities and 2.5 billion facts. A fact is defined by a triple containing two entities and a relation between them. Entities represent real world concepts, and edges represent relations, thus forming a graph-like structure. A Freebase subgraph is shown in Figure 2.1 with rectangles denoting entities.

Consider the fact BARACK OBAMA is the parent of NATASHA OBAMA. This is represented in Freebase with the triples (NATASHA OBAMA, people.person.parents, BARACK OBAMA) and (BARACK OBAMA, people.person.children, NATASHA OBAMA). Here people.person.children is the inverse relation of the main relation people.person.parents. We only consider triples with main relation. In the spirit of neo-Davidsonian semantics, we split relations between entities to as an event with two arguments. For example, we represent the triple (NATASHA OBAMA, people.person.parents, BARACK OBAMA) as event r with two arguments, people.person.parents.arg$_1$ to NATASHA OBAMA and people.person.parents.arg$_2$ to BARACK OBAMA. This representation has advantages which will become clear below.

In addition to simple facts, Freebase encodes complex facts, represented by multiple

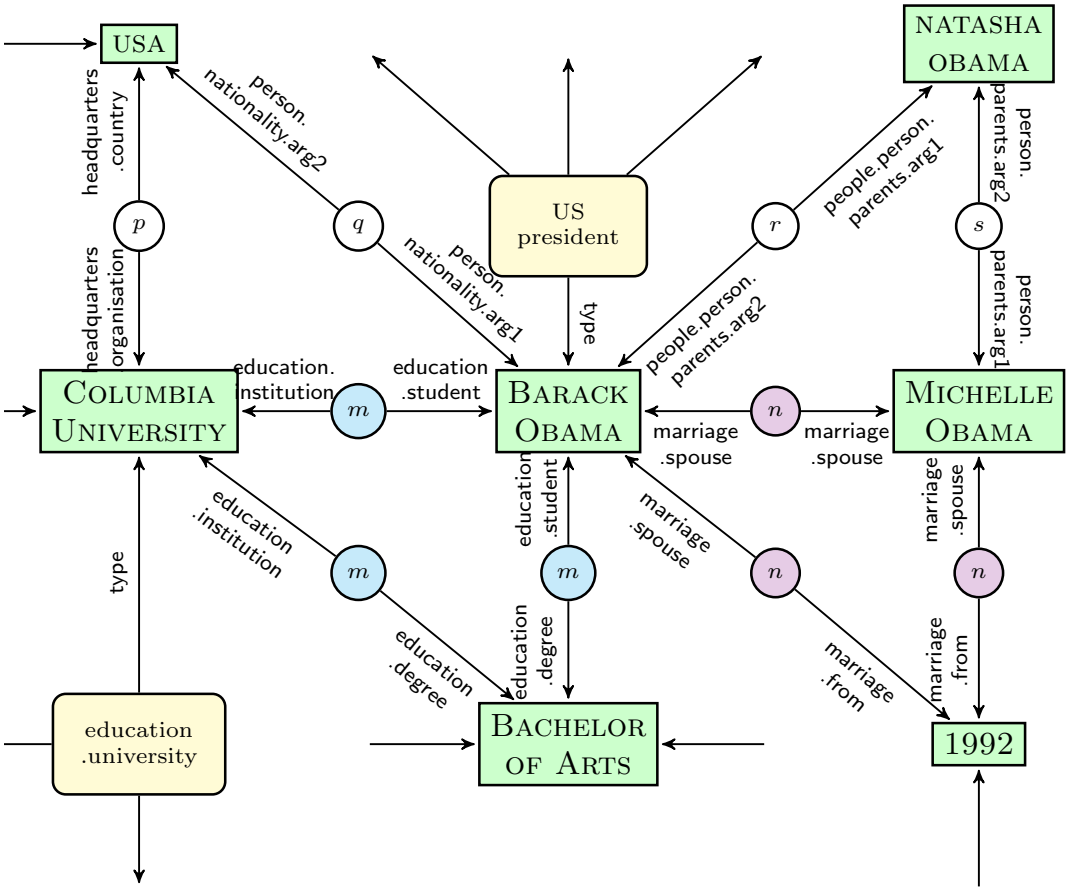


Figure 2.1: Freebase knowledge graph. Entities are represented by rectangles, relations between entities by edges, mediator nodes by circles, types by rounded rectangles.

edges. For example, in Figure 2.1, the edges connecting BARACK OBAMA, COLUMBIA UNIVERSITY and BACHELOR OF ARTS represent one single complex fact describing that Barack did his Bachelors in Columbia university. Entities in these complex facts are all connected to each other via a mediator node, e.g., circular node *m* connects Barack, Bachelors and Columbia university. This mediator node is analogous to the event variable in neo-Davidsonian semantics. Note that because of splitting simple facts to also contain a mediator node, the graph representation is uniform for both simple and complex facts.

Finally, Freebase also has entity types defining is-a relations. In Figure 2.1 types are represented by rounded rectangles (e.g., BARACK OBAMA is of type US president, and COLUMBIA UNIVERSITY is of type education.university). These are analogous to unary types in neo-Davidsonian semantics.

2.2 Overview

Our framework is shown in Figure 2.2. Given a natural language sentence, we obtain its syntactic representation using a syntactic parser. In this thesis, we use two different syntactic representations, CCG and dependency syntax. From the syntactic representation, we obtain the semantics represented in first-order logic using lambda calculus. We call these logical forms ungrounded since the predicates are domain independent. In Chapter 3, we will describe the CCG syntax-semantics interface, and in Chapter 4 we will introduce a novel semantic interface for dependency syntax. We convert the logical forms to semantic graphs, also referred to as ungrounded graphs. Using graph transformation operations, we convert ungrounded graphs to Freebase graphs, also termed as grounded graphs. For training, we only have access to question answer pairs, and not the target grounded graph structures. Therefore we use the denotations of grounded graphs to compare against the gold answers, and select the graphs that predict the correct answer as a surrogate gold standard. We train a linear model that learns to rank grounded graphs for a given ungrounded graph using its surrogate.

Existing methods approach this problem in variety of ways, mostly removing some of the steps in the proposed framework. Bordes et al. (2014) and Dong et al. (2015) present an extreme version of our pipeline which discards any form of parsing – a question is directly mapped to the answer by projecting questions and answers as vector embeddings in the same semantic space. A less extreme version of this converts the sentence to target grounded representations directly using a grammar or neural network (Zelle & Mooney, 1996; Zettlemoyer & Collins, 2005; Liang et al., 2011; Kwiatkowksi et al., 2010; Berant et al., 2013; Yih et al., 2015; Dong & Lapata, 2016). A moderate extension is to directly map syntax to the target meaning representation without going through intermediate semantic representation (Poon, 2013; Andreas & Klein, 2015). Closest to our work are Kwiatkowski et al. (2013) and Artzi et al. (2015) who convert ungrounded logical forms to target meaning representations using lambda-calculus operations instead of using graph transformations.

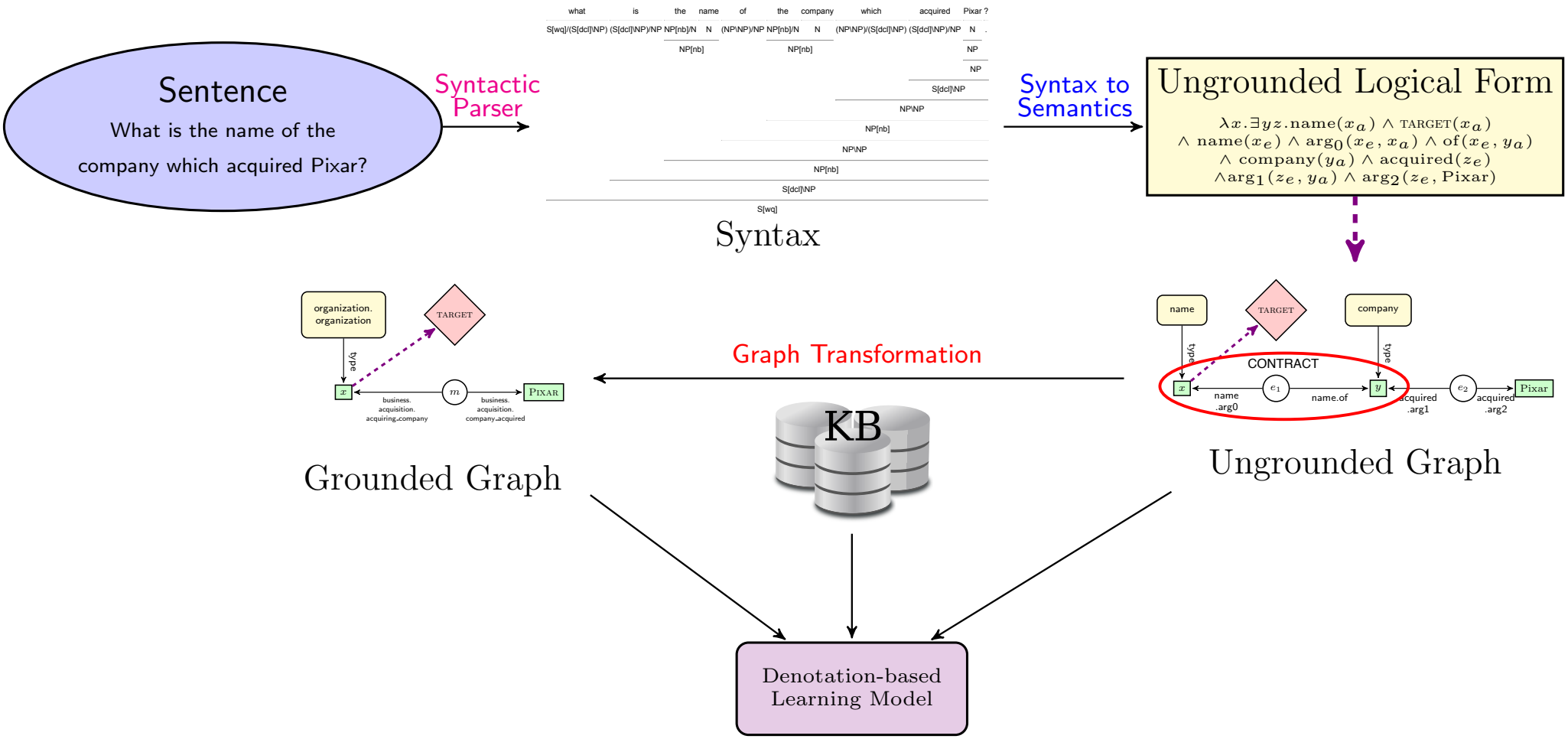


Figure 2.2: Steps involved in our Semantic Parsing framework. We first parse the sentence to a syntactic parse from which a logical form is obtained. The logical form is converted to an ungrounded graph which in turn is transformed to a Freebase graph.

In the following we provide motivation for our framework compared to existing alternatives. We first explain why natural language is represented by intermediate logical forms, and then why these are mediated by syntax. Then we justify the decision to apply graph transformations as opposed to logical form transformations.

Why intermediate logical forms? The main reason for choosing intermediate logical forms has to do with the fact that Freebase structures represent compositional structures of natural language factual statements curated by humans. Logical forms also represent the compositional structure of human languages, and these compositional structures could potentially be similar. One can exploit these similarities to guide the search for target Freebase structure of a given sentence. Consider the question, *what are the capitals of the states bordering Texas?*, and its logical form $\lambda x.\exists y.\text{capital}(x) \wedge \text{capital.of}(y,x) \wedge \text{state}(y) \wedge \text{bordering}(y,\text{Texas})$. This logical form indicates that the answer to this question x is two relations (hops) away from Texas, one hop represents the states *bordering*, and the other hop represents the *capital.of* those bordering states. Using this information, we can limit the search for Freebase graphs to $2 \pm \alpha$ hops from Texas, where α denotes few additional or fewer hops than those denoted by the logical form. Without such guidance, the search procedure becomes unrestricted, and the number of candidate graphs that would have to be considered is infinite. The number of training examples required for the latter scenario would be prohibitive, leading to scalability issues. In contrast, linguistic guidance decreases the search space.

Since our intermediate logical forms are knowledge base and task independent, they can be used with any knowledge base besides Freebase (Krishnamurthy & Mitchell, 2012, 2015), and also for other tasks such as entailment (Beltagy et al., 2016), text-based question answering (Lewis & Steedman, 2013), sentence simplification (Narayan & Gardent, 2014), summarization (Liu et al., 2015), paraphrasing (Pavlick et al., 2015), and relation extraction (Rocktäschel et al., 2015). Furthermore, our logical forms are interpretable allowing us to examine and analyze the nature of errors, e.g., whether the failure to parse to a target Freebase representation is due to the ungrammaticality of the question, or whether errors take place during the conversion stage from the logical form. Since consistency in linguistic annotations is high and cost per annotation is moderate (Bender et al., 2015), methods that benefit from linguistic information are scalable than compared to methods that rely only on task and domain specific annotations. Additionally the returns for general-purpose linguistic annotation are higher compared to annotating task specific data since the same the linguistic annotations can be reused

for multiple tasks, i.e., a one-shot solution can improve multiple tasks (Bender et al., 2015).

Several ongoing efforts aim to build task-independent semantic representations (Banarescu et al., 2013; Vanderwende et al., 2015; Akbik et al., 2015; White et al., 2016; Abzianidze et al., 2017). Our approach sheds light on the usefulness of these representations for semantic parsing.

Why syntax to intermediate logical forms? The question here is why should one use syntax to convert to intermediate logical forms rather than directly converting text to intermediate logical forms. There are many reasons for this. Logical forms are equivalent to graphs, whereas syntactic representations are most commonly trees, e.g., CCG derivations can be viewed as constituent trees, while dependencies are naturally trees. Trees offers constraints such as the number of incoming arcs to a node and projectivity, thereby restricting the search space. Whereas graph structures do not impose strong constraints on the structure leading to a larger search space compared to trees. Additionally, nodes in a syntactic tree are formed using the words in the sentence, whereas logical forms may have extra-sentential words (e.g., conjunction markers, mathematical operators).

The attractive properties of syntax resulted in mature and effective technology for syntactic parsing (Collins, 2003; Clark & Curran, 2007; Nivre et al., 2007; Chen & Manning, 2014; Dyer et al., 2015; Lee et al., 2016). Having accurate parsers for syntax is in turn a good reason to make use of syntax to derive logical forms. The improvements in parsing will translate to improvements in logical form representations.

Montague (1970) points out the role of syntax is in fact to assist in semantics. In his own words, *"I fail to see any interest in syntax except as a preliminary to semantics"*. Our approach validates how feasible is Montague's paradigm and helps in determining the merits and drawbacks of using syntax for semantics. An added advantage of using syntax is reusability. Though syntax is created to study linguistic structure of language, we recycle it for a different purpose. Syntax can also be viewed as a form of constraint type system which guides the search space for logical forms (Steedman, 2000). Existing literature also reveals it is beneficial to exploit syntax for logical form (graph) parsing (Wang et al., 2015a).

A huge advantage of syntactic representations is that these are widely available for many languages (McDonald et al., 2013; Nivre et al, 2016) compared to logical forms (Banarescu et al., 2013; Abzianidze et al., 2017), perhaps due to the ease of

annotation, strengthening the case for obtaining logical forms from syntax. Our logical form conversion assumes only the availability of syntactic treebank without recourse to annotated logical forms, thereby allowing us to work with a large number of languages (see Chapter 5).

Perhaps it is not immediately obvious why we should not dispense intermediate logical form and directly convert syntax to target meaning representation (Poon, 2013). Logical forms derived from treebank CCG categories have structural mismatches with the target representations, and it is unclear how to handle these mismatches without going through intermediate logical form representations (Kwiatkowski et al., 2013). The problem of structural mismatch is exacerbated in the case of directly converting dependency trees to target meaning representation. In Chapter 4, we will introduce a baseline which directly converts dependency trees to target structures. We observe that dependency trees without intermediate semantic representations introduce more symbols in the input making the learning problem harder, i.e., sentences with same logical form may have different dependency structures, whereas intermediate logical forms provide a reasonable abstract representation that generalizes well.

In Chapter 7, we discuss recent developments in semantic parsing that do not require either syntax or intermediate logical forms.

Why graph transformations? Here, we answer the question why we chose graph transformations instead of directly converting the intermediate logical forms to target meaning representations. The main reason is that Freebase is a graph, and a method that performs transformations on an input graph is more natural to interpret than a method which transforms intermediate logical forms to target graphs. Moreover, graphs have become a branch of computer science with solid theoretical foundations (West & others, 2001) and strong hold in NLP (Radev & Mihalcea, 2006; Riedl et al., 2017).

In addition, graphs can express whatever logical forms can express (Basile & Bos, 2013; Liang, 2013; Bos, 2016). In Section 2.9, the graph transformations we propose have analogous operations to logical form transformations, and we highlight the differences between our transformations with existing ones (Kwiatkowski et al., 2013). A major advantage of using graphs is that we can make use of existing literature in graph parsing for doing semantic parsing (Flanigan et al., 2014; Das et al., 2014; Zhou & Xu, 2015; Roth & Lapata, 2016; Damonte et al., 2016).

Below we provide more details for each step in the framework.

2.3 Natural Language Sentence to Syntax

In this step, we convert an input sentence to its syntactic representation, e.g. in Figure 2.2 we parse the input sentence *"what is the name of the company which acquired Pixar?"* using a syntactic parser to a syntactic representation. We explore two different syntactic representations, CCG and dependencies. CCG has been widely used in semantic parsing applications, however existing literature focus on inducing CCG syntax for each task separately rather than using a task-agnostic syntactic parsers. (Zettlemoyer & Collins, 2005; Kwiatkowksi et al., 2010; Kwiatkowski et al., 2013). This leaves an important question unanswered: how good is treebank syntax to perform the task at hand? A drawback of CCG is that many languages do not have CCG treebanks and it is expensive to create them. In contrast, dependencies are easy to annotate and have treebanks in many languages. To parse natural language text syntactically, we use syntactic parsers trained on existing syntactic treebanks. Using off-the-shelf syntactic parsers provides an opportunity to evaluate how useful is treebank syntax for practical tasks, and gives a glimpse of strengths and weaknesses of treebanks.

Other alternative syntacitic representations include LFG (Kaplan & Bresnan, 1982), HPSG (Pollard & Sag, 1994) and TAG (Joshi & Schabes, 1997). Similar to CCG, a drawback with these formalisms is that they do not have treebanks in many languages. Additionally, these formalisms aren't explored for grounded semantic parsing, which disallow us to compare treebank syntax with induced syntax, a major motivation for using CCG. Although dependency syntax hasn't been used for grounded semantic parsing, a major motivation for using it is that it is the most widely available syntax for many languages.

2.4 Syntax to Ungrounded Logical Form

In this step, we take the syntactic parse produced in the previous step and convert it to ungrounded logical forms, e.g., in Figure 2.2 we obtain the ungrounded logical form from CCG syntactic parse of the input sentence. We call the logical forms ungrounded since the predicates and arguments are not tied to any knowledge base. To convert syntax to ungrounded logical forms, we rely on syntax-semantic interfaces following Montague (1973). CCG is synchronous syntax-semantic interface which allows one to derive ungrounded logical forms from the syntactic derivation itself (see Chapter 3). In contrast, dependencies are not transparent to semantics and lack a semantic inter-

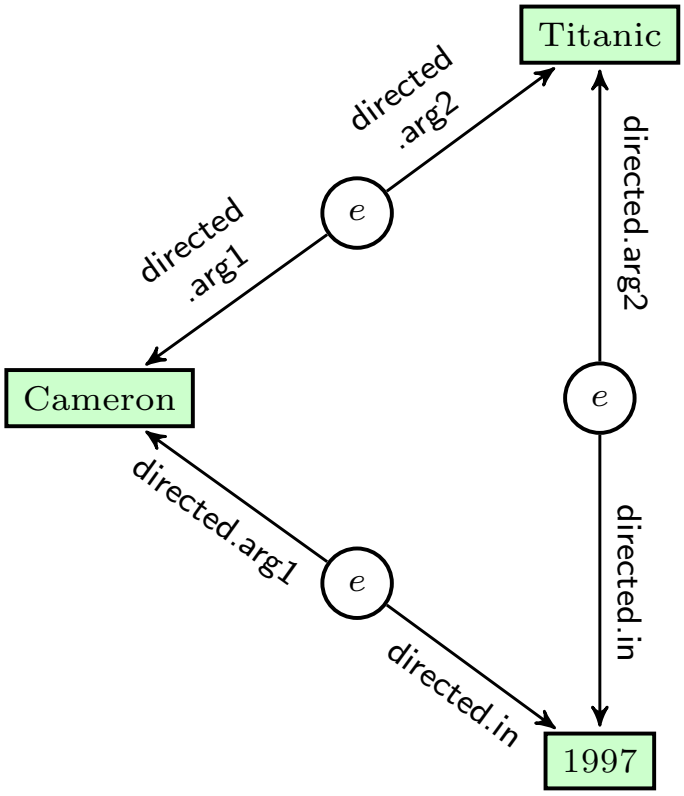
face. We will introduce novel semantic interfaces for dependencies in Chapter 4 and Chapter 5.

Traditional logic (Montague, 1973) has only two basic types: individuals and truth values. We treat numericals and time values as also individuals. The sentence *Cameron directed Titanic* would be represented as directed(Cameron, Titanic). Here the individuals are *Cameron* and *Titanic*, and *directed* takes two individuals as arguments and becomes a truth value. The notation becomes problematic with the increase in number of verbal arguments, e.g., *Cameron directed Titanic in 1997* will have directed(Cameron, Titanic, 1997), and *Cameron directed Titanic with a camera in 1997* will have $\exists x.\text{directed}(\text{Cameron}, \text{Titanic}, x, 1997) \wedge \text{camera}(x)$. In all these examples, the arity and type signature of *directed* keeps changing with each construction, even though each sentence is only adding a bit of extra information not present in the previous sentence. Parsons (1972) points out additional problems in traditional logic.

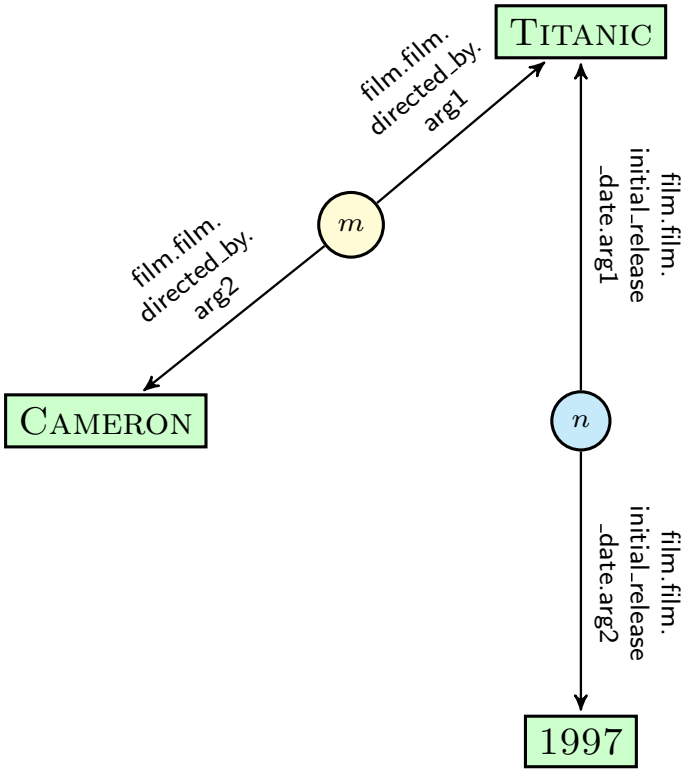
Neo-Davidsonian event semantics addresses these problems by introducing an additional basic type called event (Parsons, 1990; Schein, 1993). Following this change, the semantics of *Cameron directed Titanic with a camera in 1997* becomes $\exists ex.\text{directed}(e) \wedge \text{camera}(x) \wedge \text{arg}_1(e, \text{Cameron}) \wedge \text{arg}_2(e, \text{Titanic}) \wedge \text{with}(e, x) \wedge \text{in}(e, 1997)$. This representation makes use of event variable e and thematic roles arg_1, arg_2, *in* ... to have uniform arity predicates even when verbs have a varying number of arguments. We use neo-Davidsonian semantics to represent ungrounded logical forms. We convert syntax to logical forms in a compositional fashion by composing lambda calculus expressions of the constituent syntactic fragments. These lambda calculus expressions denote the semantics of the constituents. See Chapter 3 and Chapter 4 for details.

2.5 Ungrounded Logical form to Ungrounded Graph

As discussed earlier, graphs are a friendlier version of logical forms. And logical forms can be expressed in a graphical representation without losing expressivity. In Figure 2.2, the ungrounded logical form is converted to an ungrounded graph. We will now illustrate how we create ungrounded graphs from ungrounded logical forms. Figure 2.3(a) displays the ungrounded graph for the sentence *Cameron directed Titanic in 1997* built from $\exists e.\text{directed}(e) \wedge \text{arg}_1(e, \text{Cameron}) \wedge \text{arg}_2(e, \text{Titanic}) \wedge \text{in}(e, 1997)$. The graph representation is motivated from the structure of Freebase (Section 2.1).



(a) Ungrounded graph



(b) Grounded graph

Figure 2.3: Graph representations for the sentence *Cameron directed Titanic in 1997*.

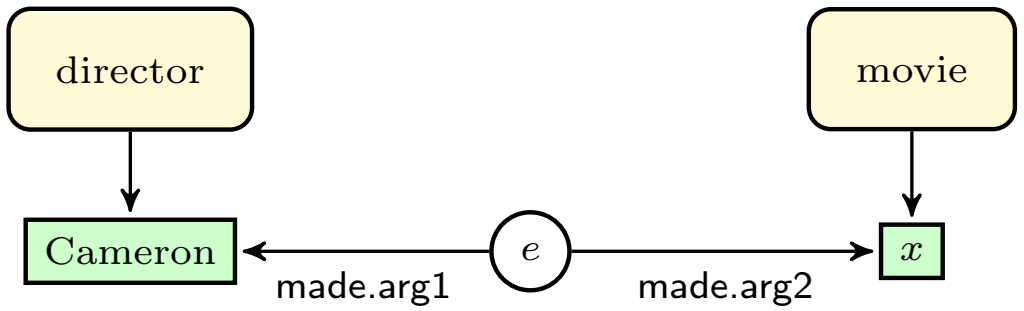


Figure 2.4: Ungrounded Graph for *The director Cameron made a movie*.

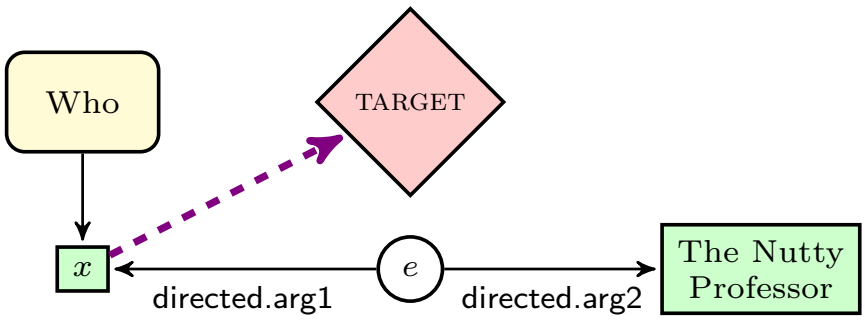
Entity Nodes (Rectangles) Entities in the logical form become nodes in the graph. These are denoted by rectangles, e.g., *Cameron* in Figure 2.3(a). In cases, where an entity is uninstantiated, nodes are represented with variables, e.g., in the sentence *The director Cameron made a movie*, the logical form $\exists ex.\text{director}(\text{Cameron}) \wedge \text{made}(e) \wedge \text{movie}(x) \wedge \text{arg}_1(e, \text{Cameron}) \wedge \text{arg}_2(e, x)$ indicates x is an uninstantiated entity corresponding to *movie*, and is represented as x (see Figure 2.4).

Mediator Nodes (Circles) Mediator nodes are denoted by circles and represent events in the logical form, e.g. e in Figure 2.3(a). They connect every pair of entities which participate in an event, thereby forming a clique. In Figure 2.3(a), entities *Cameron*, *Titanic* and *1997* form a clique evoked by the event e of *directed*.

Edges We define an *edge* as a link that connects any two entities via a mediator node. In Figure 2.3(a), the edge between Cameron and Titanic is comprised of $directed.arg_1$, e, $directed.arg_2$. The *subedge* of an edge, i.e., the link between a mediator and an entity, is formed by concatenating the event predicate and the thematic role connecting the event and the entity. The subedge label $directed.arg_1$ from mediator node e and the entity Cameron is formed by concatenating the event *directed* and the thematic role arg_1.

Type nodes (Rounded rectangles) Type nodes are denoted by rounded rectangles. They represent unary predicates in natural language. In Figure 2.4 type node *movie* indicates x is a movie, and type node *director* indicates Cameron is a director.

Math nodes (Diamonds) Math nodes are denoted by diamonds. They describe functions to be applied on the nodes/subgraphs they attach to. We define TARGET as a



(a) Who directed The Nutty Professor?

Figure 2.5: Ungrounded graph with math function TARGET.

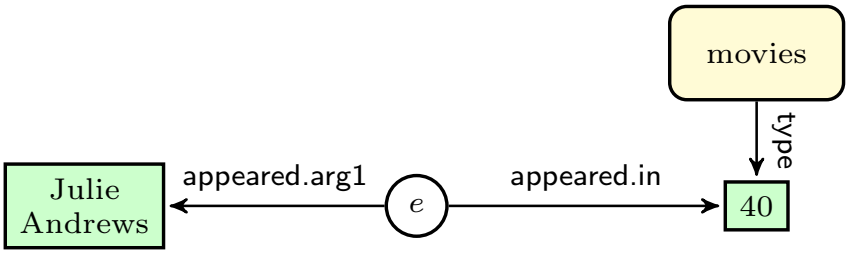
function that retrieves the denotation of the node it attaches to, i.e., the answer to a question. For example, the graph in Figure 2.5(a) represents the question *Who directed The Nutty Professor?*. Here, TARGET attaches to x representing the word *who*.

We also use an additional math function COUNT which attaches to entity nodes which have to be counted. For the sentence *Julie Andrews has appeared in 40 movies* in Figure 2.6, we have two alternate graphs for the same sentence given by two different logical forms. Figure 2.6(a) assumes thematic role *in* of *appeared* takes an integer which represents the count of movies. Whereas Figure 2.6(b) assumes thematic role *in* takes a movie z as argument, and the function COUNT takes all such movies and returns their count, which in this case is 40. We allow a sentence to have multiple ungrounded graphs in anticipation that one of these graphs is a more appropriate representation in Freebase. The learning algorithm will eventually decide which graph is more appropriate.

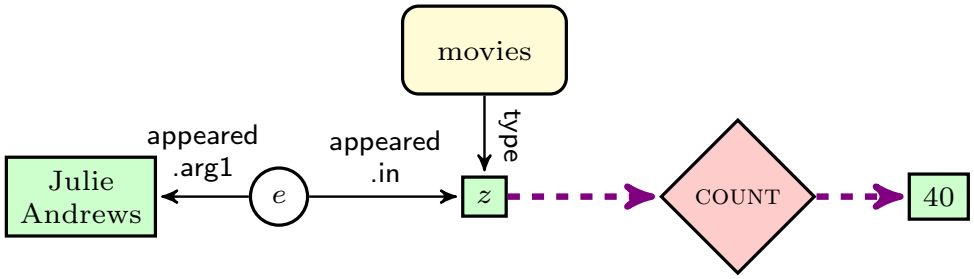
Other mathematical functions include *argmax*, *argmin*, *complement*, and comparatives such as *greater/less than*, *first*, *second*, *last* and *nth*. We do not work with these functions in this thesis.

2.6 Grounded Graphs

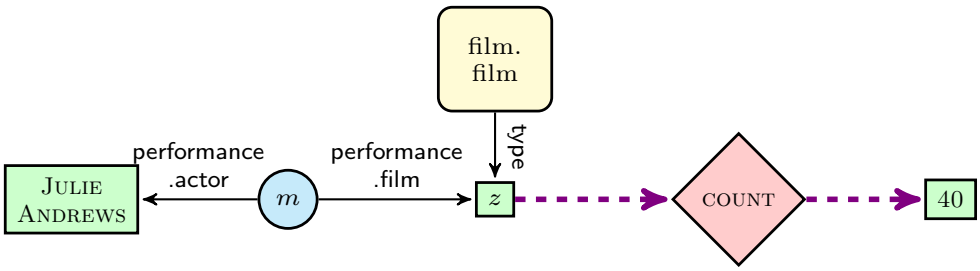
Grounded graphs are nothing but Freebase subgraphs with few operations on the graph nodes. In this section, we will see the similarities between ungrounded and grounded graphs. For each concept in the ungrounded graph, there is a corresponding concept in the grounded graphs.



(a) Ungrounded Graph



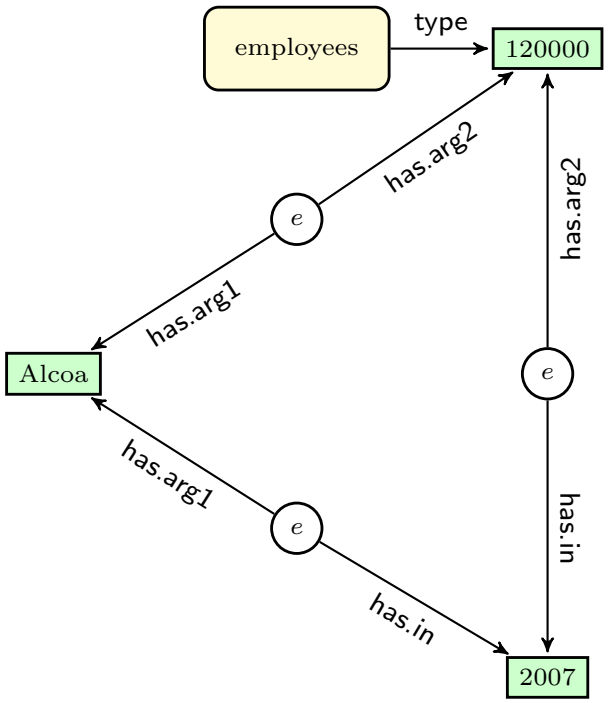
(b) Alternate Ungrounded Graph



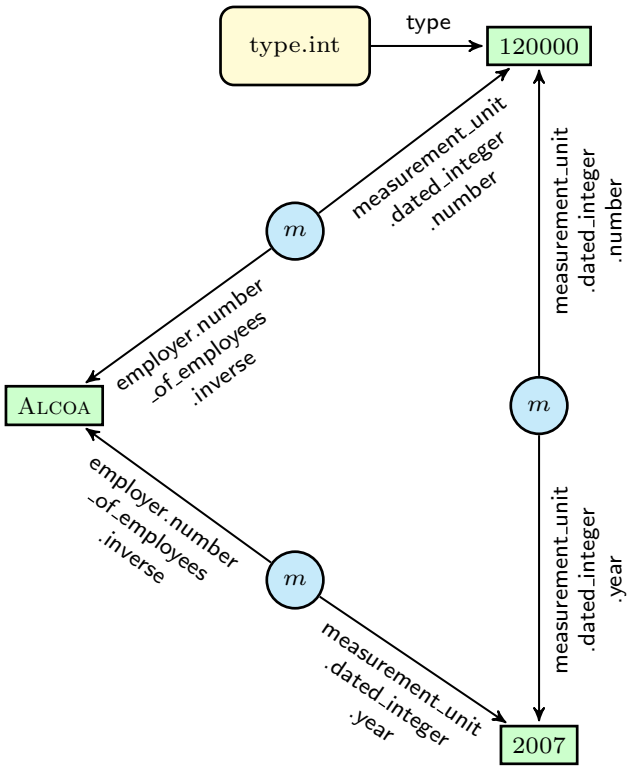
(c) Grounded graph

Figure 2.6: Graph representations for the sentence *Julie Andrews has appeared in 40 movies*. Ungrounded graph (a) directly connects *Julie Andrews* and *40*, whereas graph (b) uses the math function COUNT. Ungrounded graph (b) and grounded graph (c) have similar topology.

Entity nodes Entity nodes that are already instantiated in ungrounded graphs represent entity nodes in Freebase. We distinguish these with upper case words. In Figure 2.3(a), *Cameron* is grounded to Freebase entity CAMERON in Figure 2.3(b). Common nouns like *movies* (see Figure 2.6(b) and Figure 2.6(c)) are left as variables to be instantiated by the entities satisfying the graph.



(a) Ungrounded Graph



(b) Grounded Graph

Figure 2.7: Graph representations for *Alcoa has 120000 employees in 2007*.

Type nodes Type nodes are grounded to Freebase entity types. Type node *director* in Figure 2.4 is grounded to all possible types of Cameron (e.g., *film.director*, *film.producer*, *person.person*). In cases where an ungrounded type is attached to an entity node that is not instantiated, the type could be grounded to one of the possible grounded types that are allowed in the context. In Figure 2.6(b), the possible types of movie are all possible entity types that can occur in relation with entity Julie Andrews and one such possibility is *film.film*.

Edges An ungrounded edge between two entities can be grounded to one of the edges that link the two entities in Freebase. For example, in Figure 2.3 the possible groundings of the edge between Titanic and Cameron are the following edges between entities TITANIC and CAMERON in Freebase: (*film.directed_by.arg1, film.directed_by.arg2*) and (*film.produced_by.arg1*, *film.produced_by.arg2*). If only one entity is grounded, we use all possible edges from the grounded entity. If no entity is grounded, we allow all edges that are contextually allowed. Sometimes an ungrounded edge may not have a corresponding grounded edge. In Figure 2.3, the ungrounded edge between Cameron and 1997 does not have a corresponding grounding since Freebase does not have a relation between Cameron and 1997. Given an ungrounded graph with n edges, there are $O((k+1)^n)$ possible grounded graphs, with k being the grounded edges in the knowledge graph for each ungrounded edge together with an additional empty (no) edge.

Mediator nodes In an ungrounded graph, mediator nodes represent events in natural language. In the grounded graph, they represent Freebase fact identifiers. Fact identifiers help in identifying if neighboring edges belong to a single complex fact, which may or may not be coextensive with an ungrounded event. In Figure 2.7(a), the edges corresponding to the event identifier e are grounded to a single complex fact in Figure 2.7(b), with the fact identifier m. However, in Figure 2.3(a), the edges of the ungrounded event e are grounded to different Freebase facts, distinguished in Figure 2.3(b) by the identifiers m and n. Furthermore, the edge in 2.3(a) between CAMERON and 1997 is not grounded in 2.3(b), since no Freebase edge exists between the two entities.

2.7 Querying Freebase

We convert grounded graphs to SPARQL queries (Harris et al., 2013) in order to execute them on Freebase. Another alternative would be to directly work with graph structures by matching our graphs against the Freebase graph structure (Holzschuher & Peinl, 2013).

The SPARQL query for Figure 2.3(b) is:

```
PREFIX fb:<http://rdf.freebase.com/ns/>
ASK {
        ?m fb:film.film.directed_by.arg2 fb:CAMERON .
        ?m fb:film.film.directed_by.arg1 fb:TITANIC .
        ?n fb:film.film.initial_release_date.arg1 fb:TITANIC .
        ?n fb:film.film.initial_release_date.arg2 fb:1997 .
}.
```

This query returns either true or false depending on whether this graph exists in Freebase. The conversion is deterministic and is exactly the inverse of logical form to graph conversion (Section 2.5). We convert each edge to two triples formed by the subedges. In Figure 2.3(b), the edge between CAMERON and TITANIC, becomes the triples (?m, fb:film.film.directed_by.arg2, CAMERON) and (?m, fb:film.film.directed_by.arg1, TITANIC), formed by the subedges going form mediator *m* to CAMERON and TITANIC.

Math function TARGET is useful in retrieving the entity variables of interest. We use use the SELECT statement in such cases. The SPARQL query for the corresponding grounded graph of Figure 2.5(a) is

```
PREFIX fb:<http://rdf.freebase.com/ns/>
SELECT ?x {
        ?m fb:film.film.directed_by.arg2 fb:CAMERON .
        ?m fb:film.film.directed_by.arg1 ?x .
        ?x fb:type fb:film.film .
        fb:CAMERON fb:type fb:film.director .
}.
```

This query returns the values of *x*, i.e., the movies directed by Cameron. For questions involving counting (COUNT), we use the corresponding SPARQL predicate COUNT.

2.8 Semantic Parsing as Graph Matching

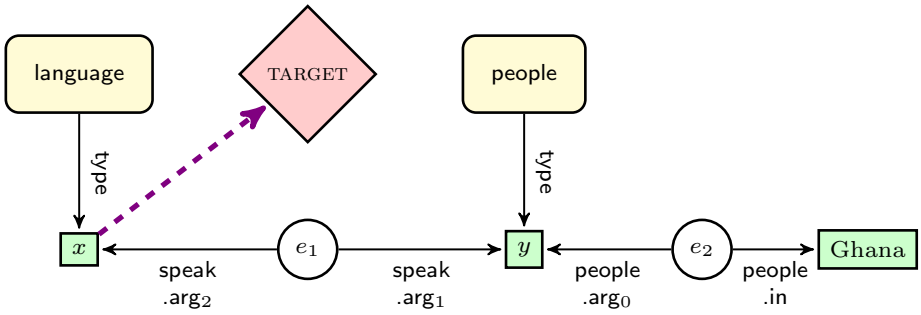
From the previous sections it is clear that ungrounded graphs and Freebase graphs have many similarities. If one replaces the nodes in an ungrounded graph with nodes in Freebase in addition to replacing the ungrounded edges/types with Freebase relations/-types, an ungrounded graph becomes a Freebase graph. In other words, there exists a Freebase graph that exactly matches an ungrounded graph. Based on this observation, we define semantic parsing as a graph matching problem. A major advantage of this proposal is that the structure of the ungrounded graph restricts the candidate graphs to be considered. In methods which convert natural language to Freebase graphs directly, the target search space is unrestricted leading to scalability issues, making them rely on augmented supervision (Jia & Liang, 2016; Kočiský et al., 2016).

In traditional semantic parsing terms, graph matching is analogous to replacing all symbols in an ungrounded logical form with grounded symbols. Working with graphs allows us to make use of graph ranking algorithms with rich features (Section 2.10). Our graph matching procedure works by transforming the ungrounded graph by replacing entity nodes with Freebase entities, edge labels with Freebase relations, and type nodes with entity types. Math nodes remain unchanged.

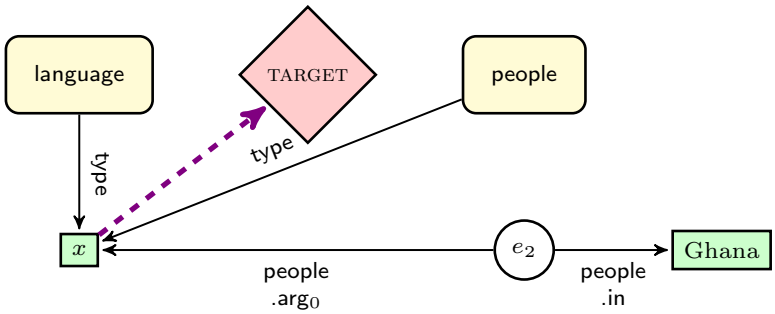
2.9 Graph Transduction Operations

A major drawback with semantic parsing as graph matching is that it assumes natural language structures and Freebase structures are isomorphic. However natural language allows the same meaning to be conveyed in sentences with different semantic structures. For example, one can ask *what is the language of Ghana?* and the same sentence can also be expressed as *What language do the people in Ghana speak?*. Although the lexical make up of these sentences is different as well as their graph structures, they express the same meaning. Freebase expresses this fact as (m, country.official_language.arg1, Ghana) and (m, country.official_language.arg2, English), a structure similar to the former question but different from latter. Figure 2.8 shows the ungrounded graph and the Freebase graphs for *What language do the people in Ghana speak?*. Due to the mismatch in structure, it is not possible to match the ungrounded graph with the target graph, thereby failing to answer this question.

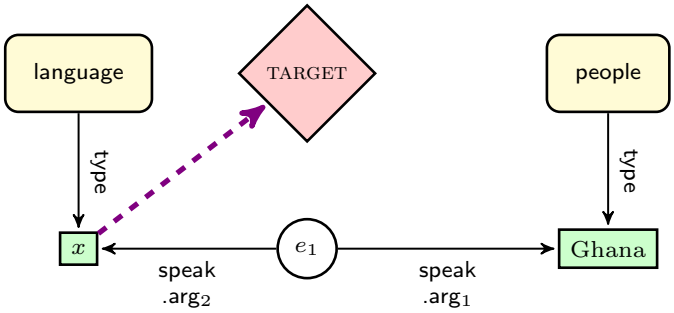
In order to address graph mismatches, we introduce two graph transduction operations: CONTRACT and EXPAND.



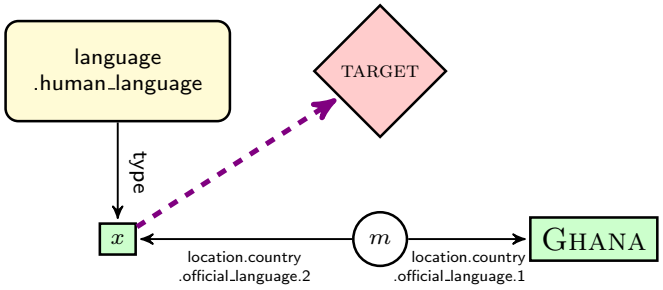
(a) Before CONTRACT.



(b) After contracting edge between x and y.



(c) After contracting edge between y and *Ghana*.



(d) Freebase graph.

Figure 2.8: The CONTRACT operation applied to the ungrounded graph for the question *What language do people in Ghana speak?*. After CONTRACT has been applied the resulting graphs are isomorphic to the representation in Freebase;

CONTRACT. The CONTRACT operation takes a pair of entity nodes connected by an edge and merges them into a single node. For example, in Figure 2.8(a) the entity nodes x and y are connected by an edge via event e. After applying CONTRACT, nodes x and y are merged to a single node to become the graph in Figure 2.8(b). Observe that all nodes that attached to y now attach to the node x after this operation. The contracted graph is can now be matched with its Freebase subgraph in Figure 2.8(d) by mapping *language* to *language.human_language*, ungrounded edge $(people.arg_0, e_2, people.arg_1)$ to the grounded edge and discarding the type *people*. Though this contraction works, this is a bad graph since the ungrounded graph indicates x is of type both language and people.

An alternative and perhaps sensible option is to contract the edge between *Ghana* and y to form the graph in Figure 2.8(c). This graph denotes people stands for the country. This graph can be successfully matched against Figure 2.8(d). We allow either of these contractions, and leave it to the learning algorithm to decide which option it prefers.

In traditional semantic parsing terms, CONTRACT operation is analogous to unifying one of the participant of an event with another participant in the same event and getting rid of the event. This mainly happens when the event indicates an equality relation between the participants, e.g., copula. Although this operation looks like a collapse operation of Kwiatkowski et al. (2013) which converts a subexpression of a logical form to a new expression of the same semantic type, a major difference is that our operation does not create new predicates or entities. For e.g., after CONTRACT operation the predicate *people* modifies entity GHANA as shown in Figure 2.8(c), whereas Kwiatkowski et al.'s collapse would create a new entity PEOPLEINGHANA. Similarly, in Figure 2.8(b), CONTRACT operation attaches *people* to individual x whereas collapse would create a new predicate *peopleSpeakLanguage* modifying x. The introducion of new predicates and entities may lead to data sparsity which is not the case with CONTRACT.

EXPAND. Since we rely on syntax to obtain logical forms, in a few cases, parser errors and ungrammatical sentences may lead to ungrounded graphs with disconnected components. Consider a scenario where the ungrammatical question *What to do Washington DC December?* results in lambda expression $\exists ex.\,\text{TARGET}(x) \wedge \text{do}(e) \wedge \text{arg}_1(e,x)$ leaving out the entities *Washington DC* and *December*. Ideally we want all entities in the input question to be present in the ungrounded graph (for entity annotation see Sec-

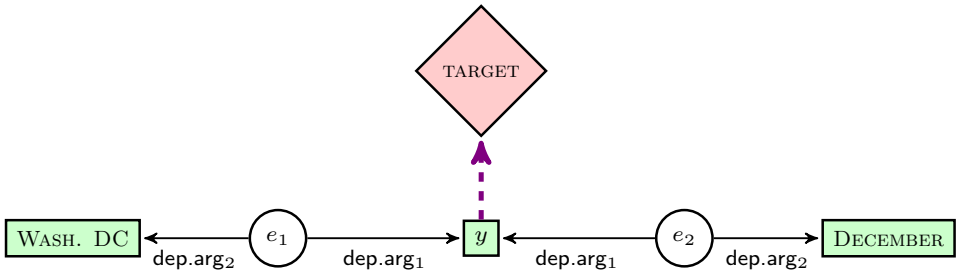


Figure 2.9: Expanded graph for the ungrammatical sentence *What to do Washington DC December?*

tion 2.11). In order to build such a graph, we create artificial edges between the question node and each entity as shown in Figure 2.9. Note that there is not much contextual information left in the graph to help in deciding correct grounded graph. Since we use sentential features such as unigrams and bigrams in addition to graph features, the learning model still has the required information to select the target graph for the given graph.

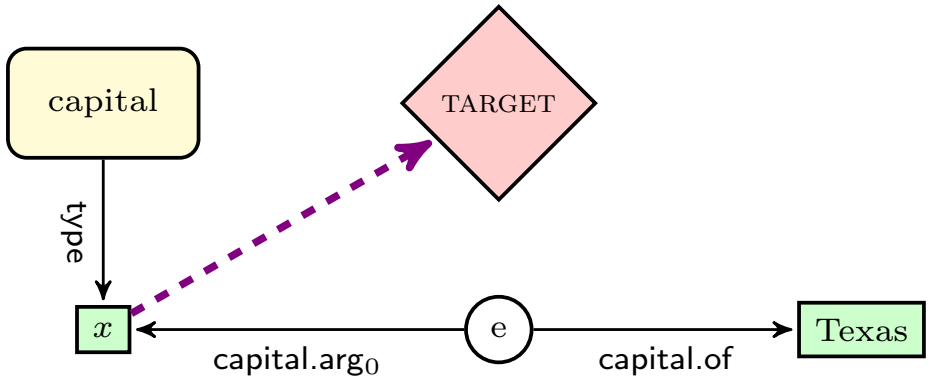
Even after these operations, there are cases where graph matching fails. Consider the sentence *Who is the grandmother of Prince William?*, the ungrounded graph contains one edge between between *who* and *Prince William*, whereas Freebase contains two edges, one for parent of Prince William, and the other for parent of William's parent. The problem here is a single lexical item *grandmother* implicitly stands for a compositional phrase parent of a parent. EXPAND can be devised to handle these cases, however making so would increase the search space. We do not handle such cases in this thesis.

This operation can be viewed as a bridging operation similar to Berant et al. (2013) (see *Structural Mismatch* paragraph in Section 1.2). Kwiatkowski et al. (2013) does not have this operation, and therefore may fail when the logical form is only partially formed without containing all the entities in the sentence.

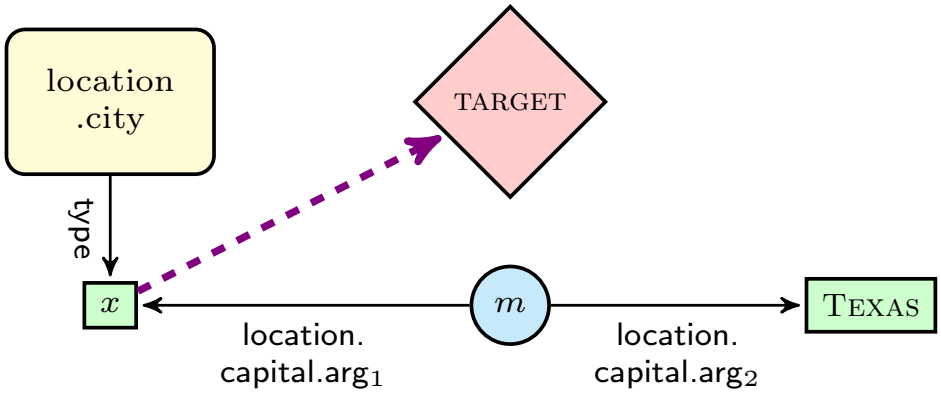
Kwiatkowski et al. (2013) has one additional operation called *split literal* operation, which splits a ternary predicate to two binary predicates. We do not require this operation since all our predicates are either binary or unary in first place. It is also not clear how to represent ternary predicates of Kwiatkowski et al. in a graph that resembles Freebase structure.

$$\exists ex.\text{TARGET}(x) \wedge \text{capital}(x) \wedge \text{capital.of.arg}_0(e,x) \wedge \text{capital.of}(e,\text{Texas})$$

(a) Semantic parse of the sentence *Austin is the capital of Texas.*

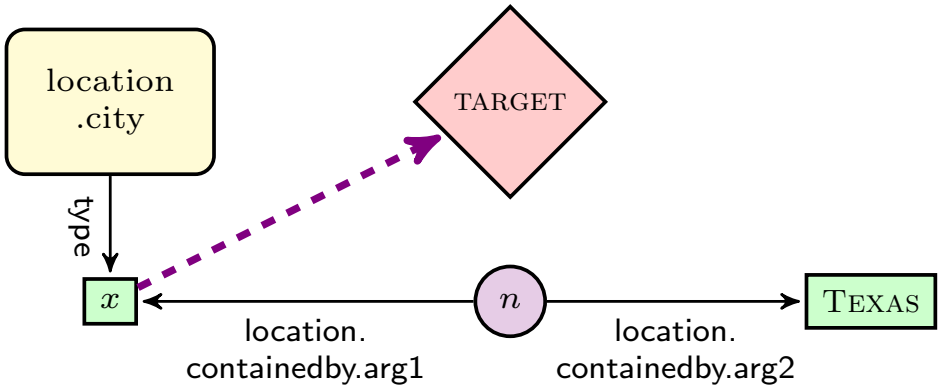


(b) Ungrounded graph.



$$[\![x]\!] = \{\text{AUSTIN}\}$$

(c) Correct Grounded Graph.



$$[\![x]\!] = \{\text{AUSTIN}, \text{DALLAS}, \text{HOUSTON} \ldots\}$$

(d) Incorrect Grounded Graph.

Figure 2.10: Logical form and the ungrounded graph of *what is the capital of Texas?*, and the grounded graphs that match the ungrounded graph.

2.10 Learning

A given ungrounded graph may match several grounded graphs in Freebase. However, only one of these represent the desired semantics. Figure 2.10 shows the ungrounded graph for *"what is the capital of Texas?"* and its matching grounded graphs. Of these, only the graph in Figure 2.10(c) results in the denotation {AUSTIN}, the desired answer. During training we use answers to the questions to prune wrongly matched grounded graphs and select plausibly correct ones by comparing the denotation of the graph with the annotated answer. In this section, we present the algorithm for building grounded graphs incrementally, and a model to rank them.

2.10.1 Building grounded graphs

We use a beam-search algorithm to convert ungrounded graphs to grounded ones. The edges and types of each ungrounded graph are placed in a queue ordered according the indices of words from which event and entities originate. At each step, we pop an element from the queue and perform one of these operations: ground the edge to Freebase, ignore the edge, or contract the edge. We rank the resulting grounded graphs using a structured perceptron model, and pick the n-best ones, where n is the beam size. The EXPAND operation is treated as a pre-processing operation on the ungrounded graph.

2.10.2 Ranking grounded graphs

Let q be a question, u be an ungrounded graph for q, g be a grounded graph, and $\mathcal{K}$ be the knowledge base. We score grounded graph g of the ungrounded graph u using a linear model which takes the features from u and g and computes the dot product between the model's weight vector θ and the feature vector, weights indicating the importance of features.

$$score = \theta \cdot \Phi(u, g, q, \mathcal{K}),$$

where $\Phi(u, g, q, \mathcal{K})$ denotes the features for the pair of ungrounded and grounded graphs. The feature function has access to the ungrounded and grounded graphs, to the question, as well as to the content of the knowledge base and the denotation $[\![g]\!]$. See Section 2.10.5 for the features employed.

We select the grounded graph with the highest score as the final model prediction.

$$(\hat{u}, \hat{g}) = \arg\max_{(u,g)} \theta \cdot \Phi(u, g, q, \mathcal{K}),$$

2.10.3 Model

We estimate the weights θ using the averaged structured perceptron algorithm (Collins, 2002). As shown in Algorithm 1, the perceptron makes several passes over sentences, and in each iteration it computes the best scoring $(\hat{g}, \hat{u})$ among the candidate graphs for a given sentence. In line 6, the algorithm updates θ with the difference (if any) between the feature representations of the best scoring graph $(\hat{g}, \hat{u})$ and the *gold standard* graph (g^+, u^+). The goal of the algorithm is to rank gold standard graphs higher than the any other graphs. The final weight vector θ is the average of weight vectors over T iterations and N sentences. This averaging procedure avoids overfitting and produces more stable results (Collins, 2002).

As seen in lines 6 and 7, this algorithm requires a gold standard graph u_i^+, g_i^+. However we only have access to answers to questions. Below we describe how we obtain a surrogate gold graph of a question.

2.10.4 Selecting Surrogate Gold Graph

Since we do not have direct access to these gold graphs, we instead rely on the set of *oracle graphs*, $O_{\mathcal{K},a_i}(q_i)$, as a proxy to obtain the surrogate gold graphs (u_i^+, g_i^+):

$$(u_i^+, g_i^+) = \arg\max_{(u_i,g_i) \in O_{\mathcal{K},a_i}(q_i)} \theta \cdot \Phi(u_i, g_i, q_i, \mathcal{K}),$$

where $O_{\mathcal{K},\mathcal{A}}(q)$ is defined as the set of pairs (u, g) derivable from the question q, whose denotation $[\![g]\!]$ has minimal F_1-loss against the gold answer a_i. We find the oracle graphs for each question a priori by performing beam-search with a beam size of 10k and only use examples with oracle $F_1 > 0.0$ for training.

2.10.5 Features

Our feature vector $\Phi(g, u, s, \mathcal{KB})$ denotes the features extracted from a question q and its corresponding graphs u and g with respect to a knowledge base $\mathcal{KB}$. The elements of the vector $(\phi_1, \phi_2, \ldots)$ take integer or real values denoting the number of times a feature appeared. We devised the following broad feature classes:

Algorithm 1: Averaged Structured Perceptron for learning to rank ungrounded and grounded graphs of a question.

Input: Question and answer pairs: $\{q_i, a_i\}_{i=1}^{N}$

$\theta \leftarrow 0$

for $t \leftarrow 1 \ldots T$ **do**

  for $i \leftarrow 1 \ldots N$ **do**

    $(\hat{g}_i, \hat{u}_i) = \arg\max_{g_i, u_i} \Phi(g_i, u_i, q_i, \mathcal{KB}) \cdot \theta$

    $(u_i^+, g_i^+) = \arg\max_{(u_i, g_i) \in O_{\mathcal{K}, a_i}(q_i)} \theta \cdot \Phi(u_i, g_i, q_i, \mathcal{K})$

    if $(u_i^+, g_i^+) \neq (\hat{u}_i, \hat{g}_i)$ **then**

      $\theta \leftarrow \theta + \Phi(g_i^+, u_i^+, s_i, \mathcal{KB}) - \Phi(\hat{g}_i, \hat{u}_i, s_i, \mathcal{KB})$

return $\frac{1}{T}\sum_{t=i}^{T}\frac{1}{N}\sum_{i=1}^{N}\theta^i{}_t$

Graph alignments Since ungrounded graphs are similar in topology to grounded graphs, we extract ungrounded and grounded edge and type alignments. So, from graphs in Figure 2.3(a) and Figure 2.3(b), we obtain the edge alignment ϕ_{edge}(directed.arg1, directed.arg2, film.directed_by.arg2, film.directed_by.arg1) and the subedge alignments ϕ_{edge}(directed.arg1, film.directed_by.arg2) and ϕ_{edge}(directed.arg2, film.directed_by.arg1). In a similar fashion we extract type alignments (e.g., ϕ_{type}(capital,location.city)).

Graph Contextual Features In addition to graph alignments, we also use contextual features from the graph which record cooccurrence of event and type predicates with grounded edge and type labels. Feature ϕ_{event} records an event word and its grounded predicates (e.g., in Figure 2.6(c) we extract features ϕ_{event}(appear, performance.film) and ϕ_{event}(appear, performance.actor). Feature ϕ_{arg} records a predicate and its argument words (e.g., ϕ_{arg}(performance.film, movie) in Figure 2.6(c)).

Transduction Features These features indicate if an edge is contracted. We use the template (MergedSubEdge, HeadSubEdge, MergedIsEntity, HeadIsEntity) to extract this feature. For example, in Figure 2.8(c), the edge between *Ghana* and *y* is contracted to form *Ghana*, resulting in the feature (people.arg$_0$, people.in, False, True). The EXPAND operation is treated as a pre-processing step and no features are used to encode its use.

Ngram features These features represent cooccurrence of words and Freebase predicates. The feature templates are (ngram, groundedRelationLeft), (ngram, groundedRela-

tionRight), and (ngram, groundedRelationLeft, groundedRelationRight). We use both unigrams and bigrams to extract these features. In Figure 2.10(c), unigrams are {*what, is, the capital, and of*} and bigrams are {*(what, is), (is, the), (the, capital), (capital, of)* ...}. The features formed with, say bigram *(capital, of)*, are (capital, of, location.capital.arg$_1$), (capital, of, location.capital.arg$_2$), (capital, of, location.capital.arg$_1$, location.capital.arg$_2$).

Lexical similarity We count the number of word stems shared by grounded and ungrounded edge labels e.g., in Figure 2.3 directed.arg1 and film.directed_by.arg2 have one stem overlap (ignoring the argument labels arg1 and arg2). For each ungrounded edge, we compute ϕ_{stem}, the aggregate stem overlap count over between the ungrounded and its corresponding grounded edge normalized using the average number of words in the ungrounded and grounded edge labels. We also have a feature for stem overlap count between the grounded edge labels and the context words.

Graph connectivity features These features penalize graphs with non-standard topology. For example, we do not want a final graph with no edges. The feature value $\phi_{hasEdge}$ is one if there exists at least one edge in the graph. We also have a feature $\phi_{nodeCount}$ for counting the number of connected nodes in the graph. Finally, feature ϕ_{colloc} captures the collocation of grounded edges (e.g., edges belonging to a single complex fact are likely to co-occur; see Figure 2.7(b)).

Entity Disambiguation Score So far, we did not mention how entity disambiguation (aka entity linking) is performed. We train an entity disambiguator to do the disambiguation (see below). We use the entity disambiguator score as a real valued feature.

2.11 Entity Disambiguation

The entity disambiguation pipeline is as follows: We first identify spans that can be potential entities in the question. For each span, we retrieve the possible entities this span may represent. Finally, we train a entity disambiguator that selects the span and its corresponding entity.

We use eight handcrafted part-of-speech patterns to identify entity span candidates. We use the Stanford CoreNLP caseless tagger for part-of-speech tagging (Manning et al., 2014). For each candidate mention span, we retrieve the top 10 entities according

to the Freebase API. We then create a lattice in which the nodes correspond to mention-entity pairs, scored by their Freebase API scores, and the edges encode the fact that no joint assignment of entities to mentions can contain overlapping spans. The 10-best entity linking lattices are scored by a structured perceptron. To train the structured perceptron, we assume in the training data, at least one gold entity is given for each question.

Finally, we generate ungrounded graphs for the top 10 paths through the lattice as ranked by structured perceptron, and treat the final entity disambiguation as part of the semantic parsing problem.

2.12 Evaluation Metric

Our semantic parsing contains various steps as described in Section 2.2. One method of evaluaton is to evaluate each step in the framework. However, we do not have access to gold ungrounded logical forms or grounded graphs. Besides this, evaluating semantic representation such as logical forms is known to be painful for the following reasons: defining a schema that is both expressive and consistent is difficult; whether one should give credit to partial as opposed to complete match is unclear; moreover logical forms are spurious that same logical form can be represented in multiple ways (Bos, 2008a).

Instead we rely on model-theoretic evaluation where the finally predicted graph is executed against Freebase, to retrieve its denotation. We compare this denotation with human annotated answer. We use average F_1 metric (Berant et al., 2013) to report the performance of our models:

$$\text{average } \mathrm{F}_1 = \sum_{i=1}^{n} \frac{2 * precision_i * recall_i}{precision_i + recall_i}$$

$$precision_i = \frac{|\cap([\![q_i]\!], a_i)|}{|[\![q_i]\!]|}$$

$$recall_i = \frac{|\cap([\![q_i]\!], a_i)|}{|a_i|}$$

Chapter 3

CCG Syntax to Logical Form

In this chapter, we will use Combinatory Categorial Grammar (CCG; Steedman 1996) as a syntactic representation for Freebase semantic parsing. Although CCG has been used earlier for grounded semantic parsing tasks starting from Zettlemoyer & Collins (2005), the existing literature focused on inducing task-specific CCG grammar rather than using treebank-based grammar. The novetly of this work is in evaluating the usefulness of treebank-based CCG grammar. Given an input sentence, we parse it with a general-purpose CCG syntactic parser trained on CCGBank (Hockenmaier & Steedman, 2007) to derive CCG syntactic parses, also known as CCG derivations. We convert CCG derivations to ungrounded logical forms exploiting the transparent syntax-semantic interface of CCG (Section 3.3). These ungrounded logical forms are converted to ungrounded graphs which are then transformed to Freebase graphs using graph transformation operations. We present results on two Freebase semantic parsing datasets: Free917 and WebQuestions. We compare with existing methods that learn domain-specific grammar as opposed to using general-purpose syntax for converting natural language to Freebase logical forms.

3.1 Motivation

CCG is an attractive syntactic formalism for semantics for the following reasons: CCG is known to handle complex syntactic phenomena such as coordination, unbounded and long-range dependencies, and non-projective constructions within the grammar itself without relying on additional post-processing steps (Clark et al., 2002; Steedman & Baldridge, 2011). It is a lexicalized formalism – the syntax and semantics assigned at the word level dictate the syntax and semantics at higher levels such as phrases

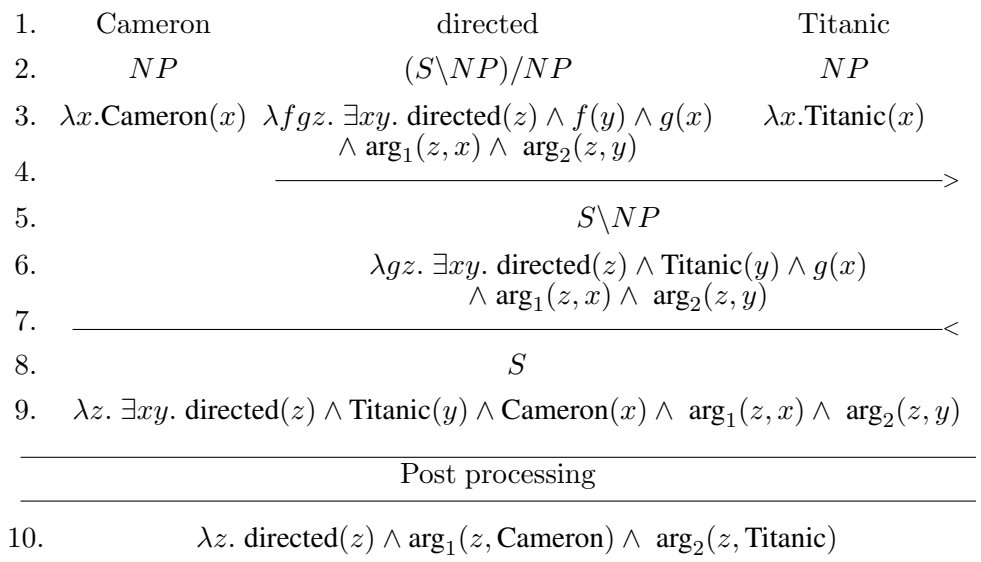


Figure 3.1: CCG syntactic derivation and its logical form derivation.

and sentences. CCG syntax is transparent to semantics, i.e., it is possible to obtain the semantic properties of a word from its syntactic properties, as well as the semantics of larger sentential fragments from their syntactic structure (see Figure 3.1). Moreover, there exists a wide-coverage CCG treebank for English (Hockenmaier & Steedman, 2007) which allowed to build highly accurate syntactic parsers (Clark et al., 2002; Clark & Curran, 2007; Lewis & Steedman, 2014; Ambati et al., 2016; Vaswani et al., 2016; Lee et al., 2016). In this thesis, semantics refer to logical forms, but CCG has also been shown useful for distributional semantics, e.g., for building higher-order tensors which capture relational properties of a word (Grefenstette & Sadrzadeh, 2011; Hermann & Blunsom, 2013; Lewis & Steedman, 2013).

CCG has been widely used for semantic parsing (Zettlemoyer & Collins, 2005, 2007; Kwiatkowksi et al., 2010; Artzi & Zettlemoyer, 2011; Matuszek et al., 2012; Krishnamurthy & Mitchell, 2012; Kwiatkowski et al., 2013; Artzi et al., 2015). However, this literature focused on learning CCG syntactic parsers using the target task as a supervision signal. These parsers do not have wide-coverage and have to be relearned for new tasks. It is not clear if CCG parsers trained on general-purpose syntax as defined in a treebank are useful for semantic parsing. We aim to answer this question in this chapter.

3.2 Combinatory Categorial Grammar

The main intuition behind CCG is that words behave like functions that take arguments and return functions. For example, a transitive verb is a function that consumes an object and returns a verb phrase which in turn is a function which consumes a subject to return a sentence. Consider the CCG derivation in Figure 3.1. Level 1 contains the words in the sentence, and level 2 contains their CCG syntactic categories. These syntactic categories describe the syntactic properties of the words in the given sentential context. Level 3 shows the lambda calculus expressions for the words based on the syntactic categories in level 2. These expressions capture lexical semantics in the given context. At level 4 the syntactic category of *directed* combines with *Titanic* using the combinator functional application (Section 3.2.3) forming the category $\mathsf{S\backslash NP}$ in level 5, and simultaneously their lambda expressions also combine to form the lambda expression at level 6. At level 7, the syntactic category of the phrase *directed Titanic* combines with *Cameron* using the combinator backward application forming the category S in level 8 and the final lambda expression in level 9. This expression is further simplified to get the final expression in level 10. We describe this derivation in more detailed below.

3.2.1 Syntactic Category

A CCG syntactic category is either of an atomic category X such as NP indicating a noun phrase, S indicating a sentence category, or a combination of two syntactic categories which form a complex category $\mathsf{X\backslash Y}$ or $\mathsf{X/Y}$, where X and Y are CCG syntactic categories, e.g., $\mathsf{S\backslash NP}$ indicates a category formed using S and NP, and $\mathsf{(S\backslash NP)/NP}$ indicates a category formed using $\mathsf{S\backslash NP}$ and NP. The category $\mathsf{X\backslash Y}$ indicates a function that returns X when it consumes Y from its left-hand side context (indicated by "\"), e.g., the category $\mathsf{S\backslash NP}$ consumes NP on the left-hand side and becomes a sentence. Similarly $\mathsf{X/Y}$ consumes Y from the right-hand context to become X. In our example, the syntactic categories of *Cameron* and *Titanic* are NPs indicating they are noun phrases. The syntactic category of the verb *directed* is $\mathsf{(S\backslash NP)/NP}$, i.e., it is a function looking for a noun phrase on its right-hand side (here *Titanic*), and once consuming this, it returns the syntactic category $\mathsf{S\backslash NP}$ (here the phrase *directed Titanic*). This in turn consumes *Cameron* to return S, the category of the sentence *Cameron directed Titanic*. One can think of syntactic categories as a compact way of describing the selectional preferences of a word in a given context.

3.2.2 Semantics

A word's syntactic category also has an equivalent typed-lambda calculus expression describing the semantic properties of the word. The syntactic category restricts the functional type of this lambda expression. Consider NP and S. Say we define the semantics of NP as a lambda expression of type $a \rightarrow t$ and S of type $e \rightarrow t$ where a, e and t indicate individual, event and truth value types, respectively. Then any syntactic category formed by combing S and NP can only be assigned a lambda expression whose functional type matches the corresponding combination of $a \rightarrow t$ and $e \rightarrow t$. For example, $(\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP}$ can only be assigned a lambda expression of type $(a \rightarrow t) \rightarrow ((a \rightarrow t) \rightarrow (e \rightarrow t))$.

In the example, *Titanic* has the lambda expression λx:a. Titanic(x). This expression indicates that it is a function which consumes an individual to become a truth value, i.e., its functional type is $(a \rightarrow t)$ as imposed by its syntactic category NP. Here that individual is an entity corresponding to the movie Titanic and not to the ship.

The lambda expression for *directed* is λf:$(a \rightarrow t)$ g:$(a \rightarrow t)$ z:e. $\exists x$:a y:a. directed(z) $\wedge$ $f(y) \wedge g(x) \wedge \text{arg}_1(z,x) \wedge \text{arg}_2(z,y)$ whose type $(a \rightarrow t) \rightarrow ((a \rightarrow t) \rightarrow (e \rightarrow t))$ matches the type requirement imposed by $(\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP}$. For simplicity we represent this expression as $\lambda fgz.\exists xy.\text{directed}(z) \wedge f(y) \wedge g(x) \wedge arg_1(z,x) \wedge arg_2(z,y)$. This expression indicates that the event *directed* has two individuals x and y participating in it, and the nature of x and y will be revealed by the functions g and f respectively. The syntactic categories of *directed* and *Titanic* are combined using forward application in line 4 (Figure 3.1), to result in the syntactic category $\mathsf{S}\backslash\mathsf{NP}$ for the phrase *directed Titanic*. Forward application corresponds to function application in lambda calculus. The lambda expression on the left is applied to the lambda expression on the right to result in a new lambda expression for the combined phrase. Here, the lambda expression of *directed* is applied to *Titanic* using function application. This is performed by replacing f with $\lambda x.\text{Titanic}(x)$. The resulting expression is $\lambda gz.\exists xy.\text{directed}(z) \wedge \text{Titanic}(y) \wedge g(x) \wedge \text{arg}_1(z,x) \wedge \text{arg}_2(z,y)$. This expression now indicates that y corresponds to the individual *Titanic*. Following the derivation further would lead to the final lambda expression of the entire sentence which is $\lambda z.\exists xy.\text{directed}(z) \wedge \text{Titanic}(y) \wedge \text{Cameron}(x) \wedge \text{arg}_1(z,x) \wedge \text{arg}_2(z,y)$.

In a post-processing step, we identify the predicates that represent entities, e.g., Titanic(x) represents the entity *Titanic* and simplify the expression further to $\lambda z.\exists xy.$ directed$(z) \wedge \text{arg}_1(z,\text{Cameron}) \wedge \text{arg}_2(z,\text{Titanic})$.

3.2.3 Combinators

CCG has many combinators to combine syntactic categories that are adjacent to each other. The goal of a CCG derivation is to find the combination of categories such that the final expression has a single category generally an S. In Figure 3.1, we have seen forward and backward applications in lines 4 and 7. These correspond to functional application in semantics. Other combinators include forward and backward compositions, crossed composition, type raising, and substitution. In this thesis we only use application, composition and type raising operations.

Forward and Backward Application Forward and backward application correspond to functional application in lambda calculus and are defined as follows:

$$\mathsf{X/Y} : f \qquad \mathsf{Y} : g \qquad \Rightarrow_{>} \qquad \mathsf{X} : f(g)$$
$$\mathsf{Y} : g \qquad \mathsf{X\backslash Y} : f \qquad \Rightarrow_{<} \qquad \mathsf{X} : f(g)$$

In our example in Figure 3.1, *directed* and *Titanic* combines in forward application (see lines 4-6), and *directed Titanic* and *Cameron* combines in backward application (lines 7-9). Forward and backward applications are denoted by the shorthand symbol $>$ (line 4) and $<$ (line 7) respectively.

Forward and Backward Composition These combinators are defined as follows:

$$\mathsf{X/Y} : f \qquad \mathsf{Y/Z} : g \qquad \Rightarrow_{>\mathbf{B}} \qquad \mathsf{X/Z} : \lambda h.f(g(h))$$
$$\mathsf{Y\backslash Z} : g \qquad \mathsf{X\backslash Y} : f \qquad \Rightarrow_{<\mathbf{B}} \qquad \mathsf{X\backslash Z} : \lambda h.f(g(h))$$

These are especially useful when application operation has to be delayed. Consider the sentence *Adam likes and Eve slightly dislikes apples* in Figure 3.2. Though the object of *dislikes* is *apples*, we do not want *dislikes* to combine with *apples* yet because *Adam likes* and *Eve slightly dislikes* are in conjunction and they should combine first before combining with *apples*. We want to delay this operation. In order to achieve this, let us first try to combine *slightly* and *dislikes*. Functional application does not permit to combine these. Instead we can use forward composition as shown in lines 4-6. Note that after this operation, the resulting category $(\mathsf{S\backslash NP})/\mathsf{NP}$ of *slightly dislikes* is still looking for the unresolved object on the right-hand side.

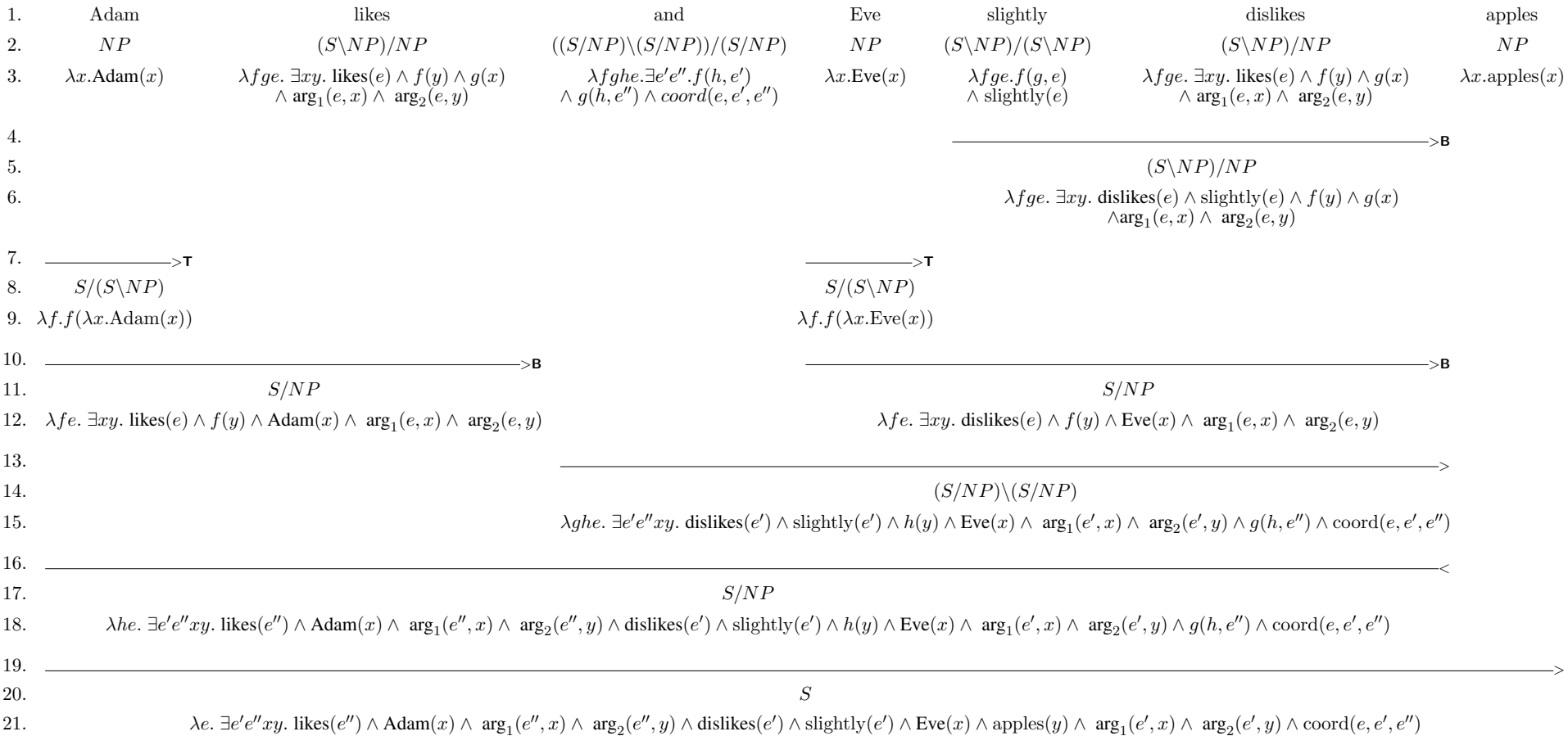


Figure 3.2: CCG syntactic and logical form derivation for the sentence *Adam likes and Eve slightly dislikes apples*.

Forward and Backward Cross compositions: Similar to forward and backward composition, crossed composition is defined as follows:

$$\mathsf{X/Y} : f \qquad \mathsf{Y\backslash Z} : g \qquad \Rightarrow_{>\mathbf{B_x}} \qquad \mathsf{X\backslash Z} : \lambda h.f(g(h))$$
$$\mathsf{Y/Z} : g \qquad \mathsf{X\backslash Y} : f \qquad \Rightarrow_{<\mathbf{B_x}} \qquad \mathsf{X/Y} : \lambda h.f(g(h))$$

Crossed composition is generally used when the functional application of a category has to be delayed due to the absence of a core argument adjacent to the category. For example, consider the sentence *"I bought today some chocolates"*, the category $(\mathsf{S\backslash NP})/\mathsf{NP}$ of *bought* is looking the for category NP, here *some chocolates*, on its immediate left-hand side, however the adverb *today* $(\mathsf{S\backslash NP})\backslash(\mathsf{S\backslash NP})$ blocks its access. In this case, *today* and *bought* combine using backward crossed composition to result in $(\mathsf{S\backslash NP})/\mathsf{NP}$ and then combines with *some chocolates*.

Type raising This combinator is defined as follows:

$$\mathsf{X} : f \qquad \Rightarrow_{>\mathbf{T}} \qquad \mathsf{T/(T\backslash X)} : \lambda h.h(f)$$
$$\mathsf{X} : f \qquad \Rightarrow_{<\mathbf{T}} \qquad \mathsf{T\backslash(T/X)} : \lambda h.h(f)$$

This operation often helps in delaying functional application, and is generally used when the main category has to be consumed by its core argument. In Figure 3.2, *likes* should have combined with *Adam* after acquiring its object. However due to the presence of conjunction, this has to be delayed, and *Adam* and *likes* should combine first. But these cannot be combined with any of the combinators proposed above. This is when type-raising is needed. Here *Adam* is type-raised to a higher-order category $\mathsf{S/(S\backslash NP)}$ (lines 7-9) which can now combine with *likes* using forward composition (lines 10-12). Type-raising is often used in combination with a composition operation.

Lemma	POS	Semantic Class	Lambda Expression
*	VB*, IN, TO, POS	EVENT	directed $\vdash (\mathsf{S_e}\backslash\mathsf{NP_x} < 1 >)/\mathsf{NP_y} < 2 >$: $\lambda fge.\exists xy.\ \text{directed}(e) \wedge g(x) \wedge f(y) \wedge \text{arg}_1(e,x) \wedge \text{arg}_2(e,y)$
*	NN, NNS	TYPE	movie $\vdash \mathsf{NP} : \lambda x.\text{movie}(x)$
*	NNP*, PRP*	ENTITY	Obama $\vdash \mathsf{NP} : \lambda x.\text{Obama}(x)$
*	RB*	EVENTMOD	annually $\vdash \mathsf{S_e}\backslash\mathsf{S_e} : \lambda fe.\text{annually}(e) \wedge f(e)$
*	JJ*	TYPEMOD	state $\vdash \mathsf{NP_x}/\mathsf{NP_x} : \lambda fx.\text{state}(x) \wedge f(x)$
be	*	COPULA	be $\vdash (\mathsf{S_x}\backslash\mathsf{NP_x})/\mathsf{NP_x}$: $\lambda fgx.f(x) \wedge g(x)$
*	CD	COUNT	twenty $\vdash \mathsf{N_x}/\mathsf{N_x} : \lambda fx.\text{COUNT}(x,20) \wedge f(x)$ twenty $\vdash \mathsf{N_x}/\mathsf{N_x} : \lambda fx.\text{equal}(x,20) \wedge f(x)$
*	CC	COORD	and $\vdash (\mathsf{NP_x}\backslash\mathsf{NP_{x'}})/\mathsf{NP_{x''}}$: $\lambda fgx.\exists x'x''.f(x'') \wedge g(x') \wedge \text{coord}(x,x',x'')$
*	WDT, WP*, WRB	QUESTION	what $\vdash \mathsf{S}[\mathsf{wq}]_\mathsf{e}/(\mathsf{S}[\mathsf{dcl}]_\mathsf{e}\backslash\mathsf{NP_x})$: $\lambda fe.\exists x.\text{TARGET}(x) \wedge f(\lambda x'.\text{equal}(x,x'),e)$
*	WDT, WP*, WRB	CLOSED	which $\vdash (\mathsf{NP_x}\backslash\mathsf{NP_x})/(\mathsf{S}[\mathsf{dcl}]_\mathsf{e}\backslash\mathsf{NP_x})$: $\lambda fgx.\exists e.f(\lambda x'.equal(x,x'),e) \wedge g(x) \wedge \text{EMPTY}(x)$

Table 3.1: Rules to classify words into semantic classes. * represents a wild card expression which matches anything. Lambda expressions for coindexed syntactic categories are shown in the rightmost column.

Word	Coindexed Category	Semantic Class	Example Lambda Expression
Adam	$\mathsf{NP_x}$	ENTITY	$\lambda x.\text{Adam}(x)$
likes	$(\mathsf{S_e}\backslash\mathsf{NP_x}<1>)/\mathsf{NP_y}<2>$	EVENT	$\lambda fge.\ \exists xy.\ \text{likes}(e)\wedge f(y)\wedge g(x)\wedge \text{arg}_1(e,x)\wedge\ \text{arg}_2(e,y)$
and	$((\mathsf{S_e}/\mathsf{NP_x})\backslash(\mathsf{S_{e'}}/\mathsf{NP_x}))/(\mathsf{S_{e''}}/\mathsf{NP_x})$	COORD	$\lambda fghe.\exists xe'e''.f(\lambda x'.equal(x,x'),e')\wedge\ g(\lambda x'.equal(x,x'),e'')\wedge\text{coord}(e,e',e'')$
Eve	$\mathsf{NP_x}$	ENTITY	$\lambda x.\text{Eve}(x)$
slightly	$(\mathsf{S_e}\backslash\mathsf{NP_x})/(\mathsf{S_e}\backslash\mathsf{NP_x})$	EVENTMOD	$\lambda fge.\exists x.f(\lambda x'.equal(x,x'),e)\wedge\ \text{slightly}(e)$
dislikes	$(\mathsf{S_e}\backslash\mathsf{NP_x}<1>)/\mathsf{NP_y}<2>$	EVENT	$\lambda fge.\ \exists xy.\ \text{dislikes}(e)\wedge f(y)\wedge g(x)\wedge \text{arg}_1(e,x)\wedge\ \text{arg}_2(e,y)$
apples	$\mathsf{NP_x}$	TYPE	$\lambda x.\text{apples}(x)$

Table 3.2: Coindexed categories, semantic classes, and lambda expressions of the words in Figure 3.2.

3.3 CCG to Logical Form

CCG syntactic parsers only provide the syntactic categories and the combinators used to combine them. But to derive the semantics of syntactic derivations, we need the logical forms corresponding to the syntactic categories. These are generally hand-written. However, this is a tedious task especially when using wide-coverage CCG syntactic categories (CCGBank has 1,286 syntactic categories in the training section of and the testing section has a few more unseen categories). In this section, we will discuss how to generate a lambda expression from syntactic categories without hand-coding all possible CCG categories. We do so by exploiting coindexed CCG categories. Coindexed categories are fine-grained versions of vanilla CCG categories with additional information such as relationship between atomic categories in the vanilla category. These are the result of creating CCGBank from Penn Treebank and are absolutely essential in order to handle long-distance dependencies (see Hockenmaier & Steedman 2007), an unavoidable side-product but a welcoming resource for semantics. Depending on the semantic class of a word, a coindexed category can be converted to a lambda expression automatically. We use a handful of rules to divide words into semantic classes. Based on the word's semantic class, we generate its lambda expression from the coindexed syntactic category. Table 3.1 shows the rules for classifying words into semantic classes and the lambda expressions for coindexed categories. Although there is a consensus among CCG community that coindexed categories can be used for generating lambda calculus expressions, there is hardly any published work or documentation on how to do this.[1] We hope this section demystifies some of the intricacies.

3.3.1 Coindexed CCG Categories

A coindexed CCG category is a vanilla CCG category enhanced with indices on the atomic categories in order to indicate their binding. Consider the vanilla category $((S\backslash NP)\backslash(S\backslash NP))/NP$. It is not clear whether the three NPs here refer to the same word or different words. However its coindexed category $((S_e\backslash NP_x)\backslash(S_e\backslash NP_x))/NP_y$ reveals that there are only two different NPs, x and y, and two of them refer to the same NP via the binding x. In addition to atomic categories, complex categories could

[1] Around the same time when this work took place, in a personal conversation with Lewis & Steedman (2013), I came to know they too used coindexed categories for lambda expression generation, but these details are absent in their paper, and unfortunately their code is also not public. A downside of Lewis & Steedman (2013) is that they use Skolem terms in their logical expressions which are unnecessary for most tasks.

also contain indices. The complete coindexed category of $((S\backslash NP)\backslash(S\backslash NP))/NP$ is $(((S_e\backslash NP_x)_e\backslash(S_e\backslash NP_x)_e)_e/NP_y)_z$. The index of a complex category indicates the ownership/origin of the category. Usually the terminal index of a category represents the current word, e.g., here z represents the current word. Similarly e in $(S\backslash NP)_e$ indicates this category is not owned by the current word but by some other word in the context. For simplicity, we only show the indices of atomic categories unless other indices are necessary, e.g., the category $(((S_e\backslash NP_x)_e\backslash(S_e\backslash NP_x)_e)_e/NP_y)_z$ can be simplified as $((S_e\backslash NP_x)\backslash(S_e\backslash NP_x))/NP_y$.

In addition to the indices, a coindexed category may also describe the predicate argument structure. Consider the category $((S_e\backslash NP_x < 1 >)/NP_y < 2 >)_e$ of the word *directed*. Here the numbers in brackets indicate the predicate argument structure between the word corresponding to the deepest atomic category (here S_e) and the words represented by other atomic categories (here NP_x and NP_y). Since the outer variable of the category is also e, here e refers to the word *directed*. This word has two arguments: argument 1 an NP in the left context indicated by x and argument 2 is another NP in the right context indicated by y.[2] We assume these argument numbers indicate meaningful argument structure if they represent words which evoke events (class EVENT in Table 3.1).

3.3.2 From Coindexed CCG Categories to Lambda Expressions

Our lambda expressions have four parts: variables, unification expressions, predicates, and arguments. Table 3.2 shows the coindexed categories and lambda expressions of our example sentence in Figure 3.2. Only words that evoke events have arguments. Let us first consider closed class words as defined in the Table 3.1. An atomic closed class category such as A_x will have the semantics $\lambda x.\text{EMPTY}(x)$, where the predicate EMPTY is a vacuous predicate, i.e., these words do not produce meaningful predicates. A complex category such as $A_x\backslash B_y$ or A_x/B_y consumes B and returns A. In lambda expression this can be defined as $\lambda f x.\text{EMPTY}(x) \wedge f(y)$, where variable f stands for B_y, and once f is consumed, the return expression is $\lambda x.\text{EMPTY}(x)\ldots$ same as the lambda expression for A_x except with an additional expression $f(y)$ in it. We call this a unification expression since the role of this expression is to glue information from

[2]This convention is slightly different from CCGBank which treats numbers in the brackets as the arguments of the current word rather than the deepest atomic category. Due to this, we had to modify a few coindexed CCG categories to obtain the desired semantics. All coindexed CCG categories are accessible at https://github.com/sivareddyg/graph-parser/blob/master/lib_data/candc_markedup.modified

neighboring context. The unification expression $f(y)$ stands for B_y and helps in getting access to the information in the actual expression of B via variable y. Let $\mathsf{A_x}|\mathsf{B_y}$ denote either $\mathsf{A_x}\backslash\mathsf{B_y}$ or $\mathsf{A_x}/\mathsf{B_y}$. Using an inductive approach, $(\mathsf{A_x}|\mathsf{B_y})|\mathsf{C_z}$ can be now be defined as $\lambda gfx.\exists yz.\text{EMPTY}(x) \wedge f(y) \wedge g(z)$, where $g(z)$ stands for $\mathsf{C_z}$.

Now consider $\mathsf{A_x}|(\mathsf{B_y}|\mathsf{C_z})$. This function takes a complex category $\mathsf{B_y}|\mathsf{C_z}$ as argument and returns $\mathsf{A_x}$. In lambda expression this can be written as $\lambda fx.\exists yz.\text{EMPTY}(x) \wedge f(\lambda z'.equal(z,z'),y)$. Here f stands for the function representing $\mathsf{B_y}|\mathsf{C_z}$. The unification expression $f(\lambda z'.equal(z,z'),y)$ indicates f consumes $\mathsf{C_z}$ (i.e., another function $\lambda z'.equal(z,z')$) and becomes B_y. Similarly $\mathsf{A_x}|(\mathsf{B_y}|(\mathsf{C_z}|\mathsf{D_w}))$ can be defined as $\lambda fx.\exists yzw.\ \text{EMPTY}(x) \wedge f(\lambda gz'.g(w) \wedge equal(z',z),y)$ where unification expression $f(\ldots)$ indicates the structure of $\mathsf{B_y}|(\mathsf{C_z}|\mathsf{D_w})$ and $\lambda gz'.g(w) \wedge equal(z,z')$ indicates the structure of $C_z|D_w$. Following an inductive approach, a lambda expression can be recursively defined for any CCG category, e.g., a relative pronoun with syntactic category $(\mathsf{NP_x}\backslash\mathsf{NP_x})/(\mathsf{S_e}\backslash\mathsf{NP_x})$ will have the expression $\lambda fgx.\exists e.\text{EMPTY}(x) \wedge f(\lambda x'.equal(x,x'),e) \wedge g(x)$, here f indicating the expression for $\mathsf{S_e}\backslash\mathsf{NP_x}$ and g for $\mathsf{NP_x}$.

For non-closed class words, instead of using the vacuous predicate EMPTY, we use the word to indicate the predicate. Consider *movie* with syntactic category $\mathsf{NP_x}$. Its lambda expression is $\lambda x.movie(x)$ instead of the closed word expression $\lambda x.\text{EMPTY}(x)$. Similarly adjectives and noun modifiers such as the word *movie* with $\mathsf{N_x}/\mathsf{N_x}$ (e.g., in the phrase *the movie Titanic*) will have $\lambda fx.\text{movie}(x) \wedge f(x)$.

In the case of event words, such as word *directed*, we also introduce predicates indicating the arguments. From its syntactic category $(\mathsf{S_e}\backslash\mathsf{NP_x} < 1 >)/\mathsf{NP_y} < 2 >$, we derive its lambda expression as $\lambda fge.\exists xy.\text{directed}(e) \wedge f(y) \wedge g(x) \wedge \text{arg}_1(e,x) \wedge \text{arg}_2(e,y)$, where $\text{arg}_1(e,x)$ and $\text{arg}_2(e,y)$ indicate the argument structure.

For a few semantic classes, we create special predicates in their lambda expression such as predicate TARGET and COUNT, specific to our application task the Freebase semantic parsing. For example, the CCG category $\mathsf{S[wq]_e}\backslash(\mathsf{S[dcl]_e}\backslash\mathsf{NP_x})$ will have an additional predicate $\text{TARGET}(x)$ in its lambda expression to indicate that this is a question, and the answer to this question is the denotation of x. The predicate COUNT indicates this sentence contains an aggregation operation.

3.3.3 CCG Derivation to Logical form

As shown in Figure 3.2, after obtaining the lambda expressions at the word level, it is straightforward to combine these expressions to obtain the final logical form using

CCG combinators. Each combinator corresponds to an operation on the logical forms as defined in Section 3.2.3.

In addition to using combinators, CCGBank also uses type-changing rules. These type-changing rules do not have a transparent semantic interpretation like the combinators. Consider the sentence, *The company Google acquired is DeepMind*, here the phrase *"The company"* has to combine with *"Google acquired"* to form the noun phrase *"The company Google acquired"*. The syntactic categories for these are NP and (S/NP) respectively and the resulting category is NP. However, none of the combinators would combine NP and (S/NP) to give NP. To address this problem, CCG uses a type changing rule to convert S/NP to NP\NP and then perform composition. But it is unclear what this transformation means in terms of lambda expression operation. We hand-code these cases indicating the transformation required.[3] In this particular case, the transformation rule would be $\lambda f g x.\exists e.f(\lambda x'.equal(x,x'),e) \wedge g(x)$ which consumes S/NP's lambda expression and returns NP\NP's lambda expression.

Our code for converting CCG derivations to logical forms can be downloaded from https://github.com/sivareddyg/graph-parser.

3.4 Experimental Setup

We next verify empirically that logical forms derived from a general-purpose CCG syntactic parser using the above method are useful for semantic parsing. We use the graph matching framework proposed in Chapter 2 for converting CCG logical forms to Freebase queries. Below, we give details on the evaluation datasets, baselines used for comparison with our model, and the implementation details.

3.4.1 Datasets

We evaluate our approach on the Free917 (Cai & Yates, 2013) and WebQuestions (Berant et al., 2013) datasets. Free917 consists of 917 questions provided by two native English speakers, one high school student and one CS undergraduate. They were asked to generate questions which can be answered by Freebase. These questions were domain-specific, and consists of 23 domains. The logical forms were later annotated by Cai & Yates. We collect the answer of each question by executing its query on Freebase. We only use these answers as supervision and ignore the annotated queries for all

[3] There are 35 hand-coded cases.

subsequent experiments. We chose this setting because existing literature hints that it is feasible to collect answers to questions for most domains rather than logical forms (Clarke et al., 2010; Liang et al., 2011; Berant et al., 2013), and so this realistic setting.

WebQuestions consists of 5810 real search engine queries starting with question words collected using Google Suggest API. Later a mechanical turk task requested that workers answer the question using only the Freebase page of the question entity, or otherwise mark it as unanswerable. Since Free917 questions are carefully annotated, we observe Free917 to be more grammatical than WebQuestions. Another main difference is that entities are already tagged in Free917 whereas for WebQuestions, we use the entity annotator described in Section 2.11.

The standard train/test splits were used for both datasets, with Free917 containing 641 train and 276 test questions and WebQuestions containing 3778 train and 2032 test questions. We tuned our model on held-out data consisting of 30% of the training questions, and used the complete training data for final testing.

3.4.2 Comparison Systems

Our goal is to evaluate the usefulness of general-purpose CCG syntactic representations for semantic parsing. To answer this, we compare our method with three related baselines: the first one induces a domain-specific CCG grammar and parser, the second one uses a hand-coded CCG grammar but learns the parser from the domain, the third one treats syntax as latent.

UBL Cai & Yates (2013) induce a CCG grammar from questions paired with logical forms using unification based learning (UBL; Kwiatkowksi et al. 2010), a method which works by splitting an input logical form to multiple lambda expressions which when composed together gives the original logical form. The constraints of this splitting are: 1) each subpart of the sentence should receive one lambda expression and a corresponding CCG category; and 2) a CCG derivation should exist such that it leads to the original logical form upon composing the lambda expressions according to CCG combinators. The constraints imposed by the CCG grammar render this search procedure tractable. A linear model learns to score the derivations that lead to the correct logical form higher than other candidates.

KCAZ13 Kwiatkowski et al. (2013) (henceforth KCAZ13) learn a semantic parser from question-answer pairs using a two-stage procedure: first, a natural language sen-

tence is converted to a domain-independent semantic parse using a small set of hand-coded CCG categories. Next this logical form is grounded onto Freebase using a set of logical-type equivalent operators. The operators explore possible ways sentence meaning could be expressed in Freebase and essentially transform logical form to match the target ontology which is Freebase in their case. Only the CCG parses that lead to correct matching are treated as correct and a linear model is trained on them. In Section 2.9 we discussed the similarities between graph transformations and logical form transformations, e.g., graph transformation CONTRACT is equivalent to collapse operation in logical forms.

SEMPRE The final comparison system is the semantic parser of Berant et al. (2013) (henceforth SEMPRE) which also uses question answer pairs for training. Unlike the above methods, this system learns a lexicon which maps words to lambda expressions which when composed leads to the final logical form directly without any intermediate representation. An additional operation called *bridging* is used when two contiguous logical forms cannot be combined. This approach uses chart-based parsing with a beam to find the derivations that could lead to the final logical form. A linear model is learned such that the derivations that lead to correct logical form are ranked higher than derivations that lead to incorrect logical forms.

A variety of approaches followed up on these basic models. We discuss these in the next chapter. In the current chapter, we solely focus on these three systems since they use similar resources and linear models like ours.

3.4.3 GRAPHPARSER

In order to evaluate the efficiency of general-purpose CCG syntax for semantic parsing, we use the semantic parsing framework introduced in Chapter 2. We call our semantic parser GRAPHPARSER since we convert NL to Freebase graphs rather than converting to Freebase logical forms directly. This conversion pipeline is shown in Figure 2.2: first an input question is parsed to a CCG syntactic derivation using a syntactic parser, and we obtain an ungrounded logical form from the derivation. This ungrounded logical form is converted to an ungrounded semantic graph which in turn is transformed to a Freebase graph.

Note that GRAPHPARSER can be used with or without the graph transduction operations, CONTRACT and EXPAND, proposed in Section 2.8. We explore combinations of

transduction operations on the development split to select the best one. These combinations include 1) using neither CONTRACT nor EXPAND; 2) using only EXPAND; 3) using only CONTRACT, and 4) using both of them. In all these cases, we use ungrounded graphs built from CCG logical forms.

3.4.4 Implementation Details

We used the EasyCCG syntactic parser (Lewis & Steedman, 2014) to obtain CCG syntactic derivations. We set the beam size to 100 (Section 2.10.1) during training and testing. We used the features introduced in Section 2.10.5. We performed the entity disambiguation for WebQuestions using Freebase API as mentioned in Section 2.11 and for Free917 we used the gold standard entity annotations that are already provided with the dataset. We obtain the oracle graphs using a beam size of 10,000 (see Section 2.10.4). Our training model has one hyperparameter: determining the number of iterations GRAPHPARSER should train for. We determined this by training on the training data and testing on the development set. The hyperparameter was set to the iteration number after which performance on development data saturated/deteriorated. Once this hyperparameter was determined, we trained on training and development data combined, for the pre-determined number of iterations, to build the final models.

3.5 Results

Our evaluation metric on WebQuestions is the average F_1 metric introduced in Section 2.12. For Free917, following previous work (Cai & Yates, 2013; Kwiatkowski et al., 2013; Berant et al., 2013), we use exact match accuracy, i.e., we treat our prediction as correct only if the predicated answer exactly matches the gold answer.

First let us study the structural properties of CCG graphs w.r.t. Freebase graphs. One may argue Freebase graphs have nothing to do with natural language structures, i.e., the target Freebase graphs cannot be obtained from CCG graphs using graph matching or transduction. To answer this, we present oracle scores with and without graph transduction operations. The oracle score indicates for a given CCG graph, if a Freebase graph can be found which results in the answer. This score is computed by randomly selecting an oracle graph for each question and comparing its denotation with the annotated answer, and taking an average over all questions (note that any graph in the set of oracle graphs of a question would have the same F_1. For more details see Section 2.10.4). Ta-

CONTRACT	EXPAND	Free917 Accuracy	WebQuestions avg. F_1
✗	✗	91.2	65.1
✗	✓	93.3	70.3
✓	✗	92.2	67.6
✓	✓	95.3	72.9

Table 3.3: Oracle results on the development splits using GRAPHPARSER with CCG ungrounded representation computed using oracle graphs. CONTRACT and EXPAND indicate the graph transduction operations introduced in Section 2.9.

ble 3.3 presents these scores. The first observation is that scores on Free917 are much higher than on WebQuestions. This is due to the noisy nature of WebQuestions. Recall that WebQuestions is created by asking humans to write answers to the question instead of asking them to annotate queries. Annotators often provided partial answers only, e.g., for the question *what are the major cities in France*, they provided *Paris* as an answer although there are many other major cities in France. In addition, some questions were annotated with incorrect answers, e.g., *what does Canada grow for food?* was answered with beers produced by Canada. Due to noise, the oracle graph may not exactly match the annotated answer.

The first row in Table 3.3 suggests that a large percentage of questions have isomorphic Freebase and CCG graph structures. This is interesting given that Freebase was created in a crowd-sourced fashion without showing annotators any natural language structures. The language structure must have influenced the annotators consciously or unconsciously. We also see that graph transduction operations give a boost in the oracle scores, indicating it is not always the case that Freebase structures are isomorphic to natural language structures (see Figure 2.8).

Table 3.4 presents end-to-end results on the development data. Although the oracle scores are high, we see that performance drops by at least 20% indicating that semantic parsing on Freebase is a hard learning task. From rows 2 and 3, we see that CONTRACT is more useful than EXPAND on Free917. Contrary to this, EXPAND seems to be more useful than CONTRACT for WebQuestions. Recall that CONTRACT is created to handle mismatches with NL structures, whereas EXPAND is created to handle parsing errors (Section 2.9). These results suggest that the CCG parser has more difficulty parsing WebQuestions than Free917, reiterating that WebQuestions is less grammatical than

CONTRACT	EXPAND	Free917 Accuracy	WebQuestions avg. F_1
✗	✗	68.3	44.7
✗	✓	69.4	47.3
✓	✗	70.4	46.5
✓	✓	71.0	48.9

Table 3.4: Prediction results on the development splits using GRAPHPARSER with CCG ungrounded representation. CONTRACT and EXPAND indicate the graph transduction operations introduced in Section 2.9.

Method	Free917 Accuracy	WebQuestions Average F_1
UBL (Cai & Yates, 2013)	59.0	–
KCAZ13 (Kwiatkowski et al., 2013)	68.0	–
SEMPRE (Berant et al., 2013)	62.0	35.7
This Work		
GRAPHPARSER	73.3	48.6

Table 3.5: Prediction results on the test splits. Here, GRAPHPARSER with CCG uses both CONTRACT and EXPAND operations.

Free917.

Table 3.5 compares GRAPHPARSER against related models on the test data. As can be seen, we outperform all comparison models by a large margin. Many factors are responsible for this. It would be unfair to say that all of this is due to CCG since GRAPHPARSER has been engineered over the years. These baselines approximately use the same resources – we differ in using a CCG parser which has been trained on a treebank. A few observations can be made from the results. UBL is known to work very well for small domains (Kwiatkowksi et al., 2010). However for Freebase, it does not seem to scale well and performs worse than SEMPRE which converts natural language to target language directly without using explicit syntax. A better option than UBL and SEMPRE is KCAZ13, which uses explicit syntax but with some knowledge on the possible syntactic categories. GRAPHPARSER can be thought of as an extreme version

of KCAZ13 where syntactic knowledge is learned in a supervised fashion, i.e., from a treebank. These results show that the use of treebank trained CCG parser is indeed useful for semantic parsing.

Error analysis on WebQuestions shows that 80% of the errors are due to two reasons: 1) lack of a strong supervised signal for pushing the best prediction from the beam to the top. This could happen for various reasons such as wrong CCG parses and as a result incorrect features that mistakenly rank incorrect Freebase graph on to top, or lack of supervised data, or unseen relations during test time; and 2) partially predicted answers due to missing additional constraints, e.g., for *what is the name of Justin Bieber's brother?*, our model correctly predicts the siblings of *Justin Bieber* however it fails to produce an additional constraint that these siblings should be males. Around 10% of the cases are due to entity annotation errors. And the rest are due to wrong annotations in WebQuestions or the ambiguous nature of certain questions, e.g., *where did X come from*, some questions are answered with nationality and some with cities where X was born. In Free917, an example for ambiguous case is *How many stores are in Nittany mall?*. While our model predicts the answer 65 using the relation shopping_center.number_of_stores, the gold standard provides the answer 25 obtained by counting all the individual stores in the mall.

3.6 Related Work

Bos et al. (2004) were the first to practically show that logical forms can be obtained from wide-coverage CCG syntactic parsers. The main idea was to hand-code logical forms associated with each syntactic category and use CCG combinators and derivation to obtain the logical forms. More recent work also followed up on this trend (Lewis & Steedman, 2013; Vo et al., 2015; Abzianidze, 2015; Martínez-Gómez et al., 2016). Unlike these approaches, we automate the logical form generation from syntactic categories making our method more robust to new unseen categories. We only hand-code categories when absolutely necessary such as in type-changing rules which are not transparent to semantics. We do not handle quantifiers such as *every*, *all*, but these can be handwritten since they are closed class words and only a handful.

Zettlemoyer & Collins (2005, 2007) were the first to use CCG syntax for grounded semantic parsing. They use a limited set of hand-coded CCG categories and learned a CCG parser that can simultaneously parse an input question to both syntactic derivation and grounded logical forms. This setting is yet to be tested on large open-domains

like Freebase. These methods may not be practical to handle all variations of language, and like UBL may not scale well to large domains. In our approach we use a general purpose CCG syntactic parser that is agnostic to the target task. Similar to ours, Artzi et al. (2015) also showed that general-purpose CCG syntax is useful to build semantic parsers for other semantic representations such as abstract meaning representation (AMR; Banarescu et al. 2013).

Reddy et al. (2014) showed that logical forms from general purpose CCG syntax are useful to learn a distantly-supervised semantic parser for Freebase. The core idea is to parse declarative sentences to ungrounded graphs using CCG, and exploit the structural similarity between the ungrounded graphs and Freebase to learn the association of natural language predicates and Freebase predicates. Similar ideas are explored in Krishnamurthy & Mitchell (2012, 2015) and Gardner & Krishnamurthy (2017).

3.7 Discussion

In this chapter, we described the CCG formalism: syntactic categories, semantics and combinators. We also described the not-so well-known but very useful coindexed syntactic categories. We discussed the parallels between coindexed categories and logical forms, and logical form generation from the coindexed categories.

The emphasis in this chapter was on evaluating if CCG syntactic derivations from a general-purpose syntactic parser help Freebase semantic parsing. We showed this by using the logical forms generated from CCG syntax to do Freebase semantic parsing using the framework presented in Chapter 2. We observed that Freebase graphs are similar in structure to CCG graphs. In comparison with induced CCG parsing methods and latent syntactic models for semantic parsing, we found our method to be more effective on Free917 and WebQuestions.

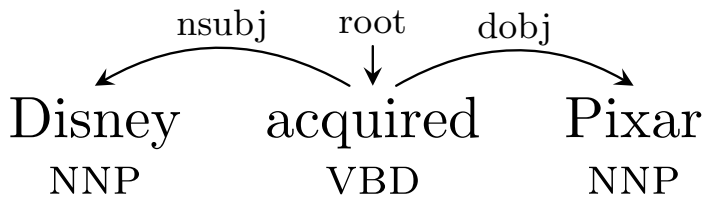
Chapter 4

Dependency Syntax to Logical Form

In this chapter, we will use dependency syntax for Freebase semantic parsing. Unlike CCG, dependency structures do not have syntactic types associated with the nodes in the structure or a type theory, making it challenging to derive logical forms from them. We propose a method, called DEPLAMBDA, that can produce logical forms from dependencies. We evaluate if these logical forms are useful for Freebase semantic parsing using the framework in Chapter 2, comparing these with CCG logical forms. Experiments on Free917 and WebQuestions datasets show that our representation is superior to the original dependency trees and, puzzlingly, outperforms CCG-based representation. We obtain the state-of-the-art results on Free917 and competitive results on WebQuestions.

4.1 Motivation

In recent years, there have been significant advances in developing fast and accurate dependency parsers for many languages (McDonald et al., 2005; Nivre et al., 2007; Martins et al., 2013; Chen & Manning, 2014; Dyer et al., 2015; Kiperwasser & Goldberg, 2016; Andor et al., 2016). Moreover dependencies are easy to annotate compared to CCG, e.g., the Stanford dependency treebank (de Marneffe et al., 2006) has around 50 dependency labels compared to >1000 syntactic categories of CCGBank. Furthermore, dependency treebanks are widely-available for many languages as opposed to any other syntactic formalism, making it a popular choice for syntactic analysis. Motivated by the desire to carry these advantages over to semantic parsing tasks, we present a robust method for mapping dependency trees to logical forms that represent underlying predicate-argument structures. By *"robust"*, we refer to the ability to gracefully handle parse errors as well as the untyped nature of dependency syntax.



(a) The dependency tree for *Disney acquired Pixar*.

$$(\text{nsubj}\ (\text{dobj}\ \text{acquired}\ \text{Pixar})\ \text{Disney})$$

(b) The s-expression for the dependency tree.

$$\lambda x.\, \exists yz.\ \text{acquired}(x_e) \wedge \text{Disney}(y_a) \wedge \text{Pixar}(z_a) \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, z_a)$$

(c) The composed lambda-calculus expression.

Figure 4.1: The dependency tree is binarized into its s-expression, which is then composed into the lambda expression representing the sentence logical form.

In this chapter, our language of interest is English, our dependency representation is Stanford dependencies, and our goal is to convert dependencies to logical forms that are useful for Freebase semantic parsing. Since our approach uses dependency trees as input, we hypothesize that it will generalize better to domains that are well covered by dependency parsers than methods that induce semantic grammars from scratch. Another question of interest is whether this representation is better than directly transducing the syntactic structure to logical form. We also compare with CCG logical forms.

The system that maps a dependency tree to its logical form using lambda calculus (henceforth DEPLAMBDA) is illustrated in Figure 4.1. First, the dependency tree is binarized via an obliqueness hierarchy to give an *s-expression* that describes the application of functions to pairs of arguments. Each node in this s-expression is then substituted for a lambda-calculus expression and the relabeled s-expression is beta-reduced to give the logical form in Figure 4.1(c). Since dependency syntax does not have an associated type theory, we introduce a type system that assigns a single type to all constituents, thus avoiding the need for type checking (Section 4.2). DEPLAMBDA uses this system to generate robust logical forms, even when the dependency structure does not mirror predicate-argument relationships in constructions such as conjunctions, prepositional

phrases, relative clauses, and wh-questions (Section 4.3).

These *ungrounded* logical forms are used for question answering against Freebase, by passing them as input to GRAPHPARSER (the framework proposed in Chapter 3), to map logical predicates to Freebase, resulting in *grounded* Freebase queries. We show that our approach achieves state-of-the-art performance on the Free917 dataset and competitive performance on the WebQuestions dataset, whereas building the Freebase queries directly from dependency trees gives significantly lower performance. We also show that our approach outperforms the CCG-based representation presented in the previous chapter.

4.2 Type System and Constraints

We use a version of the lambda calculus with three base types: individuals (**Ind**), events (**Event**), and truth values (**Bool**). Roughly speaking individuals are introduced by nouns, events are introduced by verbs, and whole sentences are functions onto truth values. For types **A** and **B**, we use $\mathbf{A} \times \mathbf{B}$ to denote the product type, while $\mathbf{A} \rightarrow \mathbf{B}$ denotes the type of functions mapping elements of **A** to elements of **B**. We will make extensive use of variables of type $\mathbf{Ind} \times \mathbf{Event}$. For any variable x of type $\mathbf{Ind} \times \mathbf{Event}$, we use $x = (x_a, x_e)$ to denote the pair of variables x_a (of type **Ind**) and x_e (of type **Event**). Here, the subscript denotes the projections $\cdot_a : \mathbf{Ind} \times \mathbf{Event} \rightarrow \mathbf{Ind}$ and $\cdot_e : \mathbf{Ind} \times \mathbf{Event} \rightarrow \mathbf{Event}$.

An important constraint on the lambda calculus system is as follows: *All natural language constituents have a lambda-calculus expression of type* $\mathbf{Ind} \times \mathbf{Event} \rightarrow \mathbf{Bool}$.

A "constituent" in this definition is either a single word, or an s-expression. S-expressions are defined formally in the next section; examples are (dobj acquired Pixar) and (nsubj (dobj acquired Pixar) Disney). Essentially, s-expressions are binarized dependency trees, which include an ordering over the different dependencies to a head (in the above the *dobj* modifier is combined before the *nsubj* modifier).

Some examples of lambda-calculus expressions for single words (lexical entries) are as follows:

$$\begin{aligned} \text{acquired} &\Rightarrow \lambda x.\, \text{acquired}(x_e) \\ \text{Disney} &\Rightarrow \lambda y.\, \text{Disney}(y_a) \\ \text{Pixar} &\Rightarrow \lambda z.\, \text{Pixar}(z_a) \end{aligned}$$

An example for a full sentence is as follows:

Disney acquired Pixar $\Rightarrow$

$$\lambda x.\, \exists yz.\, \text{acquired}(x_e) \wedge \text{Disney}(y_a) \\ \wedge \text{Pixar}(z_a) \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, z_a)$$

This is a neo-Davidsonian style of analysis. Verbs such as *acquired* make use of event variables such as x_e, whereas nouns such as *Disney* make use of individual variables such as y_a.

The restriction that all expressions are of type $\textbf{Ind} \times \textbf{Event} \rightarrow \textbf{Bool}$ simplifies the type system considerably. While it leads to difficulty with some linguistic constructions—see Section 4.3.3 for some examples—we believe the simplicity and robustness of the resulting system outweighs these concerns. It also leads to some spurious variables that are bound by lambdas or existentials, but which do not appear as arguments of any predicate: for example in the above analysis for *Disney acquired Pixar*, the variables x_a, y_e and z_e are unused. However these "spurious" variables are easily identified and discarded.

An important motivation for having variables of type $\textbf{Ind} \times \textbf{Event}$ is that a single lexical item sometimes makes use of both types of variables. For example, the noun phrase *president in 2009* has semantics

$$\lambda x.\, \exists y.\, \text{president}(x_a) \wedge \text{president\_event}(x_e) \wedge \\ \text{arg}_1(x_e, x_a) \wedge 2009(y_a) \wedge \text{prep.in}(x_e, y_a)$$

In this example *president* introduces the predicates president, corresponding to an individual, and president_event, corresponding to an event; essentially a presidency event that may have various properties. This follows the structure of Freebase closely: Freebase contains an individual corresponding to *Barack Obama*, with a *president* property, as well as an event corresponding to the *Obama presidency*, with various properties such as a start and end date, a location, and so on. The entry for *president* is then

$$\lambda x.\, \text{president}(x_a) \wedge \text{president\_event}(x_e) \wedge \text{arg}_1(x_e, x_a)$$

Note that proper nouns do not introduce an event predicate, as can be seen from the entries for *Disney* and *Pixar* above.

4.3 DepLambda

We now describe the system used to map dependency structures to logical forms. We first give an overview of the approach, then go into detail about various linguistic constructions.

4.3.1 Overview of the Approach

The transformation of a dependency tree to its logical form is accomplished through a series of three steps: *binarization*, *substitution*, and *composition*. Below, we outline these steps, with some additional remarks.

Binarization A dependency tree is mapped to an s-expression (borrowing terminology from Lisp). For example, *Disney acquired Pixar* has the s-expression

(nsubj (dobj acquired Pixar) Disney)

Formally, an s-expression has the form (exp1 exp2 exp3), where exp1 is a dependency label, and both exp2 and exp3 are either (1) a word such as *acquired*; or (2) an s-expression such as (dobj acquired Pixar).

We refer to the process of mapping a dependency tree to an s-expression as *binarization*, as it involves an ordering of modifiers to a particular head, similar to binarization of a context-free parse tree.

Substitution Each symbol (word or label) in the s-expression is assigned a lambda expression. In our running example we have the following assignments:

$$
\begin{aligned}
&\text{acquired} && \Rightarrow \lambda x.\, \text{acquired}(x_e) \\
&\text{Disney} && \Rightarrow \lambda y.\, \text{Disney}(y_a) \\
&\text{Pixar} && \Rightarrow \lambda z.\, \text{Pixar}(z_a) \\
&\text{nsubj} && \Rightarrow \lambda f g z.\, \exists x.\, f(z) \wedge g(x) \wedge \text{arg}_1(z_e, x_a) \\
&\text{dobj} && \Rightarrow \lambda f g z.\, \exists x.\, f(z) \wedge g(x) \wedge \text{arg}_2(z_e, x_a)
\end{aligned}
$$

Composition Beta-reduction is used to compose the lambda-expression terms to compute the final semantics for the input sentence. In this step expressions of the form (exp1 exp2 exp3) are interpreted as function exp1 being applied to arguments exp2 and exp3. For example, (dobj acquired Pixar) receives the following expression after composition:

$$\lambda z.\, \exists x.\, \text{acquired}(z_e) \wedge \text{Pixar}(x_a) \wedge \text{arg}_2(z_e, x_a)$$

Obliqueness Hierarchy The binarization stage requires a strict ordering on the different modifiers to each head in a dependency parse.[1] For example, in (nsubj (dobj acquired Pixar) Disney), the dobj is attached before the nsubj. The ordering is very similar to the obliqueness hierarchy in syntactic formalisms such as HPSG (Pollard & Sag, 1994).

Type for Dependency Labels. Recall from Section 4.2 that every s-expression subtree receive a logical form of type $\eta = \textbf{Ind} \times \textbf{Event} \rightarrow \textbf{Bool}$. It follows that in any s-expression (exp1 exp2 exp3), exp1 has type $\eta \rightarrow (\eta \rightarrow \eta)$, exp2 and exp3 both have type η, and the full expression has type η. Since each labeled dependency relation (e.g., nsubj, dobj, partmod) is associated with exp1 in connecting two s-expression subtrees, dependency labels always receive expressions of type $\eta \rightarrow (\eta \rightarrow \eta)$.

Mirroring Dependency Structure Whenever a dependency label receives an expression of the form

$$\lambda f g z.\, \exists x.\, f(z) \wedge g(x) \wedge \text{rel}(z_e, x_a) \tag{4.1}$$

where rel is a logical relation, the composition operation builds a structure that essentially mirrors the original dependency structure. For example nsubj and dobj receive expressions of this form, with rel = arg_1 and rel = arg_2, respectively; the final lambda expression for *Disney acquired Pixar* is

$$\lambda x.\, \exists y z.\, \text{acquired}(x_e) \wedge \text{Disney}(y_a) \\ \wedge \text{Pixar}(z_a) \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, z_a)$$

This structure is isomorphic to the original dependency structure: there are variables x_e, y_a and z_a corresponding to *acquired*, *Disney* and *Pixar*, respectively; and the subexpressions $\text{arg}_1(x_e, y_a)$ and $\text{arg}_2(x_e, z_a)$ correspond to the dependencies *acquired* $\rightarrow$ *Disney* and *acquired* $\rightarrow$ *Pixar*.

By default we assume that the predicate argument structure is isomorphic to the dependency structure and many dependency labels receive a semantics of the form shown in Equation 4.1. However, there are a number of important exceptions. As one example, the dependency label partmod receives semantics

$$\lambda f g z.\, \exists x.\, f(z) \wedge g(x) \wedge \text{arg}_1(x_e, z_a)$$

with $\text{arg}_1(x_e, z_a)$ in place of the $\text{arg}_1(z_e, x_a)$ in Equation 4.1. This reverses the dependency direction to capture the predicate-argument structure of reduced relative constructions such as *a company acquired by Disney*. Long-distance dependencies such as

[1] See *conjunctions* paragraph in Section 4.3.2 in order to understand why hierarchy is important.

in relative clauses are not explicitly indicated in a dependency tree. We will introduce a new mechanism to handle them in Section 4.3.2. In this aspect, CCG syntax is superior to dependencies since the categories inherently capture long-distance dependencies without any additional mechanism. Additional advantages of CCG are discussed in Section 4.3.3.

Post-processing We apply three post-processing steps — simple inferences over lambda-calculus expressions — to the derived logical forms. These relate to the handling of prepositions, coordination and control and are described and motivated in more detail under the respective headings below.

4.3.2 Analysis of Linguistic Constructions

In this section we describe in detail how various linguistic constructions not covered by the rule in Equation 4.1—prepositional phrases, conjunction, relative clauses, and Wh questions—are handled in the formalism.[2]

Prepositional Phrases Prepositional phrase modifiers to nouns and verbs have similar s-expressions:

(prep president (pobj in 2009))
(prep acquired (pobj in 2009))

The following entries are used in these examples:

$$\text{in} \Rightarrow \lambda x.\, \text{in}(x_e)$$
$$\text{prep} \Rightarrow \lambda f g z.\, \exists x.\, f(z) \wedge g(x) \wedge \text{prep}(z_e, x_a)$$
$$\text{pobj} \Rightarrow \lambda f g z.\, \exists x.\, f(z) \wedge g(x) \wedge \text{pobj}(z_e, x_a)$$
$$\text{president} \Rightarrow \lambda x.\, \text{president}(x_a) \wedge \text{president\_event}(x_e) \wedge \text{arg}_1(x_e, x_a)$$
$$\text{acquired} \Rightarrow \lambda x.\, \text{acquired}(x_e)$$

where the entries for prep and pobj simply mirror the original dependency structure with prep modifying the event variable z_e.

The semantics for *acquired in 2009* is as follows:

[2]The system contains 32 binarization rules (e.g., rules for obliqueness hierarchy and identifying traces) and 46 substitution rules (i.e., rules for dependency labels and parts of speech). The rules can be found at http://github.com/sivareddyg/deplambda.

$$\lambda x.\, \exists py.\, \text{acquired}(x_e) \wedge 2009(y_a) \\ \wedge \text{in}(p_e) \wedge \text{prep}(x_e, p_e) \wedge \text{pobj}(p_e, y_a)$$

We replace $\text{in}(p_e) \wedge \text{prep}(x_e, p_e) \wedge \text{pobj}(p_e, y_a)$ by $\text{prep.in}(x_e, y_a)$ as a post-processing step, effectively collapsing out the p variable while replacing the prep and pobj dependencies by a single dependency, prep.in. The final semantics are then as follows:

$$\lambda x.\, \exists y.\, \text{acquired}(x_e) \wedge 2009(y_a) \wedge \text{prep.in}(x_e, y_a)$$

In practice this step is easily achieved by identifying variables (in this case p_e) participating in prep and pobj relations. It would be tempting to achieve this step within the lambda calculus expressions themselves, but we have found the post-processing step to be more robust to parsing errors and corner cases in the usage of the prep and pobj dependency labels.

Conjunctions First consider a simple case of NP-conjunction, *Bill and Dave founded HP*. We only aim to generate the logical form with distributive reading, not the collective one, i.e., *Bill founded HP* and *Dave founded HP* rather than *Bill and Dave together founded HP*. The s-expression derived from the dependency tree is as follows:

(nsubj (dobj founded HP)
(conj-np (cc Bill and) Dave))

We make use of the following entries:

$$\text{conj-np} \Rightarrow \lambda fgx.\, \exists yz.\, f(y) \wedge g(z) \wedge \text{coord}(x, y, z)$$
$$\text{cc} \Rightarrow \lambda fgz.\, f(z)$$

The sentence *Bill and Dave founded HP* then receives the following semantics:

$$\lambda e.\, \exists xyzu.\, \text{Bill}(y_a) \wedge \text{Dave}(z_a) \wedge \text{founded}(e_e) \wedge \text{HP}(u_a) \\ \wedge \text{coord}(x, y, z) \wedge \text{arg}_1(e_e, x_a) \wedge \text{arg}_2(e_e, u_a)$$

Note how the x variable occurs in two sub-expressions: $\text{coord}(x, y, z)$, and $\text{arg}_1(e_e, x_a)$. It can be interpreted as a variable that conjoins variables y and z together. In particular, we introduce a post-processing step where the sub-expression $\text{coord}(x, y, z) \wedge \text{arg}_1(e_e, x_a)$ is replaced with $\text{arg}_1(e_e, y_a) \wedge \text{arg}_1(e_e, z_a)$, and the x variable is removed. The resulting expression is as follows:

$$\lambda e.\exists yzu.\mathrm{Bill}(y_a) \wedge \mathrm{Dave}(z_a) \wedge \mathrm{founded}(e_e) \wedge \mathrm{HP}(u_a) \\ \wedge \mathrm{arg}_1(e_e, y_a) \wedge \mathrm{arg}_1(e_e, z_a) \wedge \mathrm{arg}_2(e_e, u_a)$$

VP-coordination is treated in a very similar way. Consider the sentence *Eminem signed to Interscope and discovered 50 Cent.* This has the following s-expression:

(nsubj (conj-vp (cc s-to-I and) d-50) Eminem)

where s-to-I refers to the VP *signed to Interscope*, and d-50 refers to the VP *discovered 50 Cent.* The lambda-calculus expression for conj-vp is identical to the expression for conj-np:

$$\text{conj-vp} \Rightarrow \lambda fgx.\exists yz.\, f(y) \wedge g(z) \wedge \mathrm{coord}(x, y, z)$$

The logical form for the full sentence is then

$$\lambda e.\exists xyz.\mathrm{Eminem}(x_a) \wedge \mathrm{coord}(e, y, z) \\ \wedge \mathrm{arg}_1(e_e, x_a) \wedge \mathrm{s\_to\_I}(y) \wedge \mathrm{d\_50}(z)$$

where we use $\mathrm{s\_to\_I}(y)$ and $\mathrm{d\_50}(z)$ as shorthand for the lambda-calculus expressions for the two VPs.

After post-processing this is simplified to

$$\lambda e.\exists xyz.\mathrm{Eminem}(x_a) \wedge \mathrm{arg}_1(y_e, x_a) \\ \wedge \mathrm{arg}_1(z_e, x_a) \wedge \mathrm{s\_to\_I}(y) \wedge \mathrm{d\_50}(z)$$

Other types of coordination, such as sentence-level coordination and PP coordination, are handled with the same mechanism. All coordination dependency labels have the same semantics as conj-np and conj-vp. The only reason for having distinct dependency labels for different types of coordination is that different labels appear in different positions in the obliqueness hierarchy. This is important for getting the correct scope for different forms of conjunction. For instance, the following s-expression for the *Eminem* example would lead to an incorrect semantics:

(conj-vp (nsubj (cc s-to-I and) Eminem) d-50)

This s-expression is not possible under the obliqueness hierarchy, which places nsubj modifiers to a verb after conj-vp modifiers.

We realize that this treatment of conjunction is quite naive in comparison to that on offer in CCG. However, given the crude analysis of conjunction in dependency syntax, a more refined treatment is beyond the scope of the current approach.

Relative Clauses Our treatment of relative clauses is closely related to the mechanism for traces described by Moortgat (1988, 1991); see also Carpenter (1998) and Pereira (1990). Consider the NP *Apple which Jobs founded* with s-expression:

(rcmod Apple
(wh-dobj (BIND f (nsubj (dobj founded f) Jobs))
which))

Note that the s-expression has been augmented to include a variable f in dobj position, with *(*BIND f *...)* binding this variable at the clause level. These annotations are added using a set of heuristic rules over the original dependency parse tree.

The BIND operation is interpreted in the following way. If we have an expression of the form

$$(\text{BIND}\ \ f\ \ \lambda x.\, g(x))$$

where f is a variable and g is an expression that includes f, this is converted to

$$\lambda z.\, \exists x.\, g(x)\mid_{f=\text{EQ}(z)}$$

where $g(x)\mid_{f=\text{EQ}(z)}$ is the expression $g(x)$ with the expression $\text{EQ}(z)$ substituted for f. $\text{EQ}(z)(z')$ is true iff z and z' are equal (refer to the same entity). In addition we assume the following entries:

wh-dobj $\Rightarrow \lambda f g z.\, f(z)$
rcmod $\Rightarrow \lambda f g z.\, f(z) \wedge g(z)$

It can be verified that *(*BIND f (nsubj (dobj founded f*)* Jobs*))* has semantics

$$\lambda u.\, \exists x y z.\, \text{founded}(x_e) \wedge \text{Jobs}(y_a) \wedge \text{EQ}(u)(z) \\ \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, z_a)$$

and *Apple which Jobs founded* has semantics

$$\lambda u.\, \exists x y z.\, \text{founded}(x_e) \wedge \text{Jobs}(y_a) \wedge \text{EQ}(u)(z) \\ \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, z_a) \wedge \text{Apple}(u_a)$$

as intended. Note that this latter expression can be simplified, by elimination of the z variable, to

$$\lambda u.\, \exists x y.\, \text{founded}(x_e) \wedge \text{Jobs}(y_a) \\ \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, u_a) \wedge \text{Apple}(u_a)$$

Wh Questions Wh questions are handled using the BIND-mechanism described in the previous section. As one example, the s-expression for *Who did Jim marry* is as follows:

(wh-dobj (BIND f (nsubj (aux (dobj marry f) did)
Jim)) who)

We assume the following lambda expressions:

Who $\Rightarrow \lambda x.\,\text{TARGET}(x_a)$
did $\Rightarrow \lambda x.\,\text{TRUE}$
aux $\Rightarrow \lambda fg.\,f$
wh-dobj $\Rightarrow \lambda fgx.\,f(x) \wedge g(x)$

It can be verified that this gives the final logical form

$$\lambda x.\,\exists yz.\,\text{TARGET}(x_a) \wedge \text{marry}(y_e) \wedge \text{Jim}(z_a) \wedge \text{arg}_1(y_e, z_a) \wedge \text{arg}_2(y_e, x_a)$$

Note that the predicate TARGET is applied to the variable that is the focus of the question. A similar treatment is used for cases with the wh-element in subject position (e.g., *who married Jim*) or where the wh-element is extracted from a prepositional phrase (e.g., *who was Jim married to*).

4.3.3 Comparison to CCG

In this section we discuss some differences between our approach and CCG-based approaches for mapping sentences to logical forms. Although our focus is on CCG, the arguments are similar for other formalisms that use the lambda calculus in conjunction with a generative grammar, such as HPSG and LFG, or approaches based on context-free grammars.

Our approach differs in two important (and related) respects from CCG: (1) all constituents in our approach have the same semantic type ($\textbf{Ind} \times \textbf{Event} \rightarrow \textbf{Bool}$); (2) our formalism does not make the argument/adjunct distinction, instead essentially treating all modifiers to a given head as adjuncts.

As an example, consider the analysis of *Disney acquired Pixar* within CCG. In this case *acquired* would be assigned the following CCG lexical entry:

$$\text{S}\backslash\text{NP}/\text{NP} \Rightarrow \lambda f_2 f_1 x.\,\exists yz.\,\text{acquired}(x) \wedge f_1(y) \wedge f_2(z) \wedge \text{arg}_1(x, y) \wedge \text{arg}_2(x, z)$$

Note the explicit arguments corresponding to the subject and object of this transitive verb (f_1 and f_2, respectively). An intransitive verb such as *sleeps* would be assigned an entry with a single functional argument corresponding to the subject (f_1):

$$\text{S}\backslash\text{NP} \Rightarrow \lambda f_1 x.\, \exists y.\, \text{sleeps}(x) \wedge f_1(y) \wedge \text{arg}_1(x, y)$$

In contrast, the entries in our system for these two verbs are simply $\lambda x.\, \text{acquired}(x_e)$ and $\lambda x.\, \text{sleeps}(x_e)$. The two forms are similar, have the same semantic type, and do not include variables such as f_1 and f_2 for the subject and object.

The advantage of our approach is that it is robust, and relatively simple, in that a strict grammar that enforces type checking is not required. However, there are challenges in handling some linguistic constructions. A simple example is passive verbs. In our formalism, the passive form of *acquired* has the form $\lambda x.\, \text{acquired.pass}(x_e)$, distinct from its active form $\lambda x.\, \text{acquired}(x_e)$. The sentence *Pixar was acquired* is then assigned the logical form $\lambda x.\, \exists y.\, \text{acquired.pass}(x_e) \wedge \text{Pixar}(y_a) \wedge \text{arg}_1(x_e, y_a)$. Modifying our approach to give the same logical forms for active and passive forms would require a significant extension of our approach. In contrast, in CCG the lexical entry for the passive form of *acquired* can directly specify the mapping between subject position and the arg_2:

$$\text{S}\backslash\text{NP} \Rightarrow \lambda f_2 x.\, \exists z.\, \text{acquired}(x) \wedge f_2(z) \wedge \text{arg}_2(x, z)$$

As another example, correct handling of object and subject control verbs is challenging in the single-type system: for example, in the analysis for *John persuaded Jim to acquire Apple*, the CCG analysis would have an entry for *persuaded* that explicitly takes three arguments (in this case *John*, *Jim*, and *to acquire Apple*) and assigns *Jim* as both the direct object of *persuaded* and as the subject of *acquire*. In our approach the subject relationship to *acquire* is instead recovered in a post-processing step, based on the lexical identity of *persuaded*.

4.4 Experimental Setup

We next verify empirically that our proposed approach derives a useful logical compositional semantic representation from dependency syntax. We use the semantic parsing as graph matching framework proposed in Chapter 2 for converting dependency logical forms to Freebase queries. Below, we give details on the evaluation datasets, baselines used for comparison, our main model, and the implementation details.

4.4.1 Datasets

We use the same datasets used in the previous chapter: Free917 and WebQuestions (See Section 3.4.1). A main difference is instead of syntactically parsing these datasets using a CCG parser, we use a dependency parser (see below).

4.4.2 Comparison Systems

In addition to the dependency-based semantic representation DEPLAMBDA (Section 4.3) and previous work on these datasets, we compare to three additional baseline representations outlined below. We use GRAPHPARSER (the tool that implements framework in Chapter 2) to map these representations to Freebase. GRAPHPARSER takes the logical forms from syntax, converts them to ungrounded graphs which in turn are converted to Freebase grounded graphs.

DEPTREE In this baseline, an ungrounded graph is created directly from the original dependency tree. An event is created for each parent and its dependents in the tree. Each dependent is linked to this event with an edge labeled with its dependency relation, while the parent is linked to the event with an edge labeled arg_0. If a word is a question word, an additional TARGET predicate is attached to its entity node.

SINGLEEVENT This representation is agnostic to the syntax of the input question. In this representation, we create a single event to which all entities in the question are connected by the predicate arg_1. An additional TARGET node is connected to the event by the predicate arg_0. This is similar to the template representation of Yao (2015) and Bast & Haussmann (2015). Note that this cannot represent any compositional structure involving aggregation or multihop reasoning.

CCGGRAPH Finally, we compare to the CCG-based semantic representation presented in Chapter 3, adding the CONTRACT and EXPAND operations to increase its expressivity.

4.4.3 Implementation Details

We used the hypergraph parser of Zhang & McDonald (2014) for dependency parsing. The tagger and parser are both trained on the OntoNotes 5.0 corpus (Weischedel et al., 2011), with constituency trees converted to Stanford-style dependencies (de Marneffe

Representation	-C -E	-C +E	+C -E	+C +E
(a) Average oracle F_1				
DEPTREE	30.8	30.8	72.8	72.8
SINGLEEVENT	73.0	73.0	73.0	73.0
CCGGRAPH	65.1	70.3	67.6	72.9
DEPLAMBDA	64.8	66.3	71.8	73.0
(b) Average number of oracle graphs per question				
DEPTREE	1.5	1.5	354.6	354.6
SINGLEEVENT	1.5	1.5	1.8	1.8
CCGGRAPH	1.6	1.7	3.4	3.4
DEPLAMBDA	1.4	1.5	3.6	4.2
(c) Average F_1				
DEPTREE	19.9	19.9	42.6	42.6
SINGLEEVENT	49.0	49.0	48.2	48.2
CCGGRAPH	44.7	47.3	46.5	48.9
DEPLAMBDA	45.9	47.5	48.8	50.4

Table 4.1: Oracle statistics and accuracies on the WebQuestions development set. +(-)C: with(out) CONTRACT. +(-)E: with(out) EXPAND.

et al., 2006). To derive the CCG-based representation, we use the output of the Easy-CCG parser (Lewis & Steedman, 2014). We set the beam size to 100 (Section 2.10.1) during training and testing. We used all the features introduced in Section 2.10.5. We perform the entity disambiguation for WebQuestions using Freebase API as mentioned in Section 2.11 and for Free917 we use the gold standard entity annotations that are already provided with the dataset.

4.5 Results

We examine the different representations for question answering along two axes. First, we compare their expressiveness in terms of answer reachability assuming a perfect model. Second, we compare their performance with a learned model. Finally, we conduct a detailed error analysis of DEPLAMBDA, with a comparison to the errors made by

Representation	-C -E	-C +E	+C -E	+C +E
(a) Oracle Accuracy				
DEPTREE	26.0	26.0	95.8	95.8
SINGLEEVENT	96.3	96.3	96.3	96.3
CCGGRAPH	91.2	93.3	92.2	95.3
DEPLAMBDA	91.1	92.7	94.3	95.8
(b) Average number of oracle graphs per question				
DEPTREE	1.2	1.2	285.4	285.4
SINGLEEVENT	1.6	1.6	1.8	1.8
CCGGRAPH	1.6	1.6	2.4	2.5
DEPLAMBDA	1.5	1.5	3.3	3.4
(c) Accuracy				
DEPTREE	21.3	21.3	51.6	51.6
SINGLEEVENT	40.9	40.9	42.0	42.0
CCGGRAPH	68.3	69.4	70.4	71.0
DEPLAMBDA	69.3	71.3	72.4	73.4

Table 4.2: Oracle statistics and accuracies on the Free917 development set. +(-)C: with(out) CONTRACT. +(-)E: with(out) EXPAND.

CCGGRAPH. For WebQuestions evaluation is in terms of the average F_1-score across questions (Section 2.12), while for Free917, evaluation is in terms of exact answer accuracy (Section 3.5).

4.5.1 Expressiveness of the Representations

Table 4.1(a) and Table 4.2(a) show the oracle F_1-scores of each representation on the WebQuestions and Free917 development sets respectively. According to the first column (-C -E), the original DEPTREE representation can be directly mapped to Freebase for less than a third of the questions. Adding the CONTRACT operation (+C) improves this substantially to an oracle F_1 of about 73% on WebQuestions and 95.8% on Free917. However, this comes at the cost of massive spurious ambiguity: from Table 4.1(b) there are on average over 300 oracle graphs for a single dependency tree. Table 4.1(c) shows the results of the different representations on the WebQuestions development set. Spurious ambiguity clearly hampers learning and DEPTREE falls behind the other representations. CCGGRAPH and DEPLAMBDA align much more closely to Freebase and achieve similar oracle F_1 scores with far less spurious ambiguity. SINGLEEVENT, which cannot represent any compositional semantics, is competitive with these syntax-based representations. This might come as a surprise, but it simply reflects the fact that the dataset does not contain questions that require compositional reasoning.

4.5.2 Results on WebQuestions and Free917

We use the best settings on the development set in subsequent experiments, i.e., with CONTRACT and EXPAND enabled. Table 4.3 shows the results on the WebQuestions and Free917 test sets with additional entries for recent prior work on these datasets. The trend from the development set carries over and DEPLAMBDA outperforms the other graph-based representations, while performing slightly below the state-of-the-art model of Yih et al. (2015) ("Y&C"), which uses a separately trained entity resolution system (Yang & Chang, 2015). When using the standard Freebase API ("FB API") for entity resolution, the performance of their model drops to 48.4% F_1.

On Free917, DEPLAMBDA outperforms all other representations by a wide margin and obtains the best result to date. Interestingly, DEPTREE outperforms SINGLEEVENT in this case. We attribute this to the small training set and larger lexical variation of Free917. The structural features of the graph-based representations seem highly beneficial in this case.

Method	Free917 Accuracy	WebQuestions Average F_1
Cai & Yates (2013)	59.0	–
Berant et al. (2013)	62.0	35.7
Kwiatkowski et al. (2013)	68.0	–
Yao & Van Durme (2014)	–	33.0
Berant & Liang (2014)	68.5	39.9
Bao et al. (2014)	–	37.5
Bordes et al. (2014)	–	39.2
Yao (2015)	–	44.3
Yih et al. (2015) (FB API)	–	48.4
Bast & Haussmann (2015)	76.4	49.4
Berant & Liang (2015)	–	49.7
Yih et al. (2015) (Y&C)	–	**52.5**
This Work		
DEPTREE	53.2	40.4
SINGLEEVENT	43.7	48.5
CCGGRAPH (+C +E)	73.3	48.6
DEPLAMBDA (+C +E)	**78.0**	50.3

Table 4.3: Question-answering results on the WebQuestions and Free917 test sets.

4.5.3 Error Analysis

We categorized 100 errors made by DEPLAMBDA (+C +E) on the WebQuestions development set. In 43 cases the correct answer is present in the beam, but ranked below an incorrect answer (e.g., for *where does volga river start*, the annotated gold answer is *Valdai Hills*, which is ranked second, with *Russia, Europe* ranked first). In 35 cases, only a subset of the answer is predicted correctly (e.g, for *what countries in the world speak german*, the system predicts *Germany* from the human_language.main_country Freebase relation, whereas the gold relation human_language.countries_spoken_in gives multiple countries). Together, these two categories correspond to roughly 80% of the errors. In 10 cases, the Freebase API fails to add the gold entity to the lattice (e.g., for *who is blackwell*, the correct *blackwell* entity was missing). Due to the way WebQuestions was crowdsourced, 9 questions have incorrect or incomplete gold annotations (e.g., *what does each fold of us flag means* is answered with *USA*). The remaining 3 cases are due to structural mismatch (e.g., in *who is the new governor of florida 2011*, the graph failed to connect the target node with both *2011* and *Florida*).

Due to the ungrammatical nature of WebQuestions, CCGGRAPH fails to produce ungrounded graphs for 4.5% of the complete development set, while DEPLAMBDA is more robust with only 0.9% such errors. The CCG parser is restricted to produce a sentence tag as the final category in the syntactic derivation, which penalizes ungrammatical analyses (e.g., *what victoria beckham kids names* and *what nestle owns*). Examples where DEPLAMBDA fails due to parse errors, but CCGGRAPH succeed include *when was blessed kateri born* and *where did anne frank live before the war*. Note that the EXPAND operation mitigates some of these problems. While CCG is known for handling comparatives elegantly (e.g., *who was sworn into office when john f kennedy was assassinated*), we do not have a special treatment for them in the semantic representation. Differences in syntactic parsing performance and the somewhat limited expressivity of the semantic representation are likely the reasons for CCGGRAPH's lower performance.

4.6 Related Work

There have been extensive studies on extracting semantics from syntactic representations such as LFG (Dalrymple et al., 1995), HPSG (Copestake et al., 2001, 2005), TAG (Gardent & Kallmeyer, 2003; Joshi et al., 2007) and CCG (Baldridge & Kruijff, 2002; Bos et al., 2004; Steedman, 2012; Artzi et al., 2015). However, few have used

dependency structures for this purpose. Debusmann et al. (2004) and Cimiano (2009) describe grammar-based conversions of dependencies to semantic representations, but do not validate them empirically. Stanovsky et al. (2016) use heuristics based on linguistic grounds to convert dependencies to proposition structures. Bédaride & Gardent (2011) propose a graph-rewriting technique to convert a graph built from dependency trees and semantic role structures to a first-order logical form, and present results on textual entailment. Our work, in contrast, assumes access only to dependency trees and offers an alternative method based on the lambda calculus, mimicking the structure of knowledge bases such as Freebase; we further present extensive empirical results on recent question-answering corpora.

Structural mismatch between the source semantic representation and the target application's representation is an inherent problem with approaches using general-purpose representations. We saw this problem exacerbated when syntax is directly transduced to the target meaning representation (DEPTREE baseline). There is a growing work on converting syntactic structures to the target application's structure without going through an intermediate semantic representation, e.g., answer-sentence selection (Punyakanok et al., 2004; Heilman & Smith, 2010; Yao et al., 2013) and semantic parsing (Ge & Mooney, 2009; Poon, 2013; Parikh et al., 2015; Xu et al., 2015; Wang et al., 2015a; Andreas & Klein, 2015; Andreas et al., 2016).

4.7 Discussion

We have introduced a method for converting dependency structures to logical forms using the lambda calculus. A key idea of this work is the use of a single semantic type for every constituent of the dependency tree, which provides us with a robust way of compositionally deriving logical forms. The resulting representation is subsequently grounded to Freebase by learning from question-answer pairs. Empirically, the proposed representation was shown to be superior to the original dependency trees and more robust than logical forms derived from a CCG parser.

Chapter 5

Universal Semantic Parsing: Universal Dependencies to Logical Forms

Universal Dependencies (UD) provides a cross-linguistically uniform syntactic representation, with the aim of advancing multilingual applications of parsing and natural language understanding. In the previous chapter we introduced DEPLAMBDA, a semantic interface for (English) Stanford Dependencies, based on the lambda calculus. In this chapter, we introduce UDEPLAMBDA, a similar semantic interface for UD, which allows mapping natural language to logical forms in an almost language-independent framework. We evaluate these logical forms on Freebase semantic parsing. To facilitate multilingual evaluation, we provide German and Spanish translations of the WebQuestions and GraphQuestions datasets. Results show that UDEPLAMBDA outperforms strong baselines across languages and datasets. For English, it achieves the strongest result to date on GraphQuestions, with competitive results on WebQuestions.

5.1 Motivation

The Universal Dependencies (UD) initiative seeks to develop cross-linguistically consistent annotation guidelines as well as a large number of uniformly annotated treebanks for many languages.[1] Such resources could advance multilingual applications of parsing, improve comparability of evaluation results, enable cross-lingual learning, and more generally support natural language understanding.

Seeking to exploit the benefits of UD for natural language understanding, we introduce UDEPLAMBDA, a semantic interface for UD that maps natural language to logical

[1] http://www.universaldependencies.org/.

forms, representing underlying predicate-argument structures, in an almost language-independent manner. Our method is based on DEPLAMBDA (Chapter 4). The conversion process is illustrated in Figure 5.1. Whereas DEPLAMBDA works only for English, UDEPLAMBDA applies to any language for which UD annotations are available.[2] In this chapter, we describe the rationale behind UDEPLAMBDA and highlight important differences from DEPLAMBDA, some of which stem from the different treatment of various linguistic constructions in UD.

We evaluate these logical form on multilingual Freebase semantic parsing. To facilitate multilingual evaluation, we provide translations of the English WebQuestions (Berant et al., 2013) and GraphQuestions (Su et al., 2016) datasets to German and Spanish. We demonstrate that UDEPLAMBDA can be used to derive logical forms for these languages using a minimal amount of language-specific knowledge. Aside from developing the first multilingual semantic parsing tool for Freebase, we also experimentally show that UDEPLAMBDA outperforms strong baselines across languages and datasets. For English, it achieves the strongest result to date on GraphQuestions, with competitive results on WebQuestions. Our implementation and translated datasets are publicly available at https://github.com/sivareddyg/udeplambda.

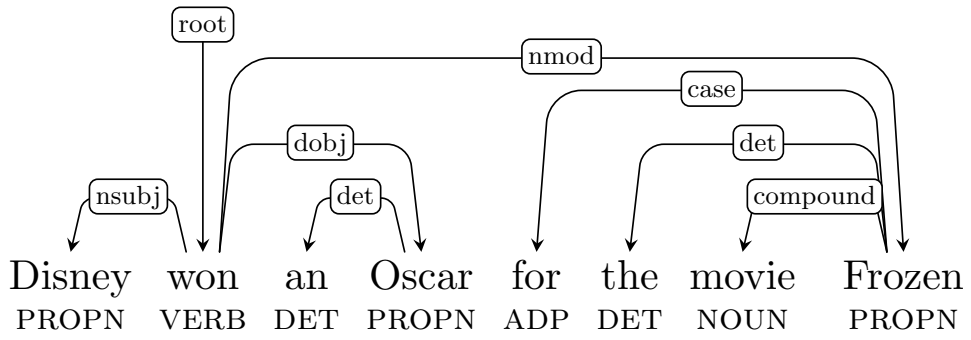
5.2 Notation

Consider the dependency tree in Figure 5.1(a) represented in UD schema. Since the schema has changed, we define a new obliqueness hierarchy. Figure 5.1(b) shows the s-expression obtained by following the hierarchy defined in Section A.2. The desired lambda expression is shown in Figure 5.1(c). Below we show the lambda expressions of some of the words and dependency labels that resulted in this logical form. For convenience, we define macros for most frequently used logical form templates.

We define the following macros for words:

ENTITY $\Rightarrow \lambda x.\,\text{word}(x_a)$; e.g. Oscar $\Rightarrow \lambda x.\,\text{Oscar}(x_a)$
EVENT $\Rightarrow \lambda x.\,\text{word}(x_e)$; e.g. won $\Rightarrow \lambda x.\,\text{won}(x_e)$
FUNCTIONAL $\Rightarrow \lambda x.\,\text{TRUE}$; e.g. an $\Rightarrow \lambda x.\,\text{TRUE}$

[2] As of v1.3, UD annotations are available for 47 languages.



(a) The dependency tree for *Disney won an Oscar for the movie Frozen* in the Universal Dependencies formalism.

(nsubj (nmod (dobj won (det Oscar an))
(case (det (comp. Frozen movie) the) for)) Disney)

(b) The binarized s-expression for the dependency tree.

$$\lambda x.\, \exists yzw.\, \text{won}(x_e) \wedge \text{Disney}(y_a) \wedge \text{Oscar}(z_a) \wedge \text{Frozen}(w_a) \wedge \text{movie}(w_a) \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, z_a) \wedge \text{nmod.for}(x_e, w_a)$$

(c) The composed lambda-calculus expression.

Figure 5.1: The mapping of a dependency tree to its logical form with the intermediate s-expression.

Similarly for dependency labels we define the following macros:

COPY $\Rightarrow \lambda fgx.\exists y.\, f(x) \wedge g(y) \wedge \text{rel}(x,y)$

e.g. `nsubj`, `dobj`, `nmod`, `advmod`

INVERT $\Rightarrow \lambda fgx.\exists y.\, f(x) \wedge g(y) \wedge \text{rel}^{\text{i}}(y,x)$

e.g. `amod`, `acl`

MERGE $\Rightarrow \lambda fgx.\, f(x) \wedge g(x)$

e.g. `compound`, `appos`, `amod`, `acl`

HEAD $\Rightarrow \lambda fgx.\, f(x)$

e.g. `case`, `punct`, `aux`, `mark`.

As an example of COPY, consider the lambda expression for `dobj` in (dobj won (det Oscar an)): $\lambda fgx.\exists y.\, f(x) \wedge g(y) \wedge \text{arg}_2(x_e, y_a)$. Since $\text{arg}_2(x_e, y_a)$ mimics the dependency structure dobj(won, Oscar), we refer to the expression kind evoked by `dobj` as COPY.

Expressions that invert the dependency direction are referred to as INVERT (e.g. `amod` in *running horse*); expressions that merge two subexpressions without introducing any relation predicates are referred to as MERGE (e.g. `compound` in *movie Frozen*); and expressions that simply return the parent expression semantics are referred to as HEAD (e.g. `case` in *for Frozen*). While this generalization applies to most dependency labels, several labels take a different logical form not listed here, some of which are discussed in Section 5.3.3. [3] Sometimes the mapping of dependency label to lambda expression may depend on surrounding part-of-speech tags or dependency labels. For example, `amod` acts as INVERT when the modifier is a verb (e.g. in *running horse*), and as MERGE when the modifier is an adjective (e.g. in *beautiful horse*).[4]

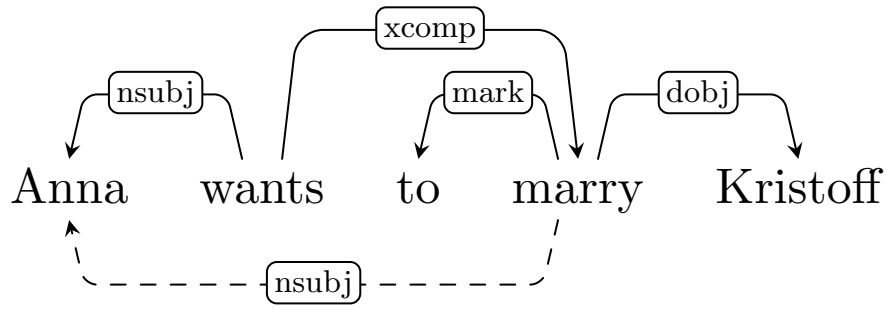
5.3 UDepLambda

We now introduce UDepLambda, a semantic interface for Universal Dependencies.[5] Whereas DepLambda only applies to English Stanford Dependencies, UDepLambda takes advantage of the cross-lingual nature of UD to facilitate an (almost) language independent semantic interface. This is accomplished by restricting the binarization,

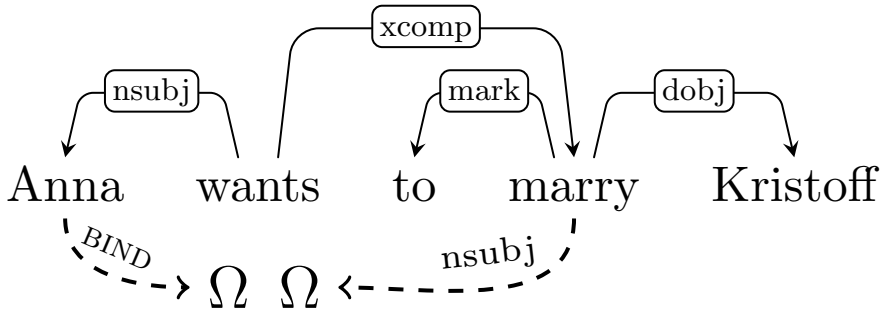
[3] Mappings are available at https://github.com/sivareddyg/udeplambda.

[4] We use Tregex (Levy & Andrew, 2006) for substitution mappings and Cornell SPF (Artzi, 2013) as the lambda-calculus implementation. For example, in *running horse*, the tregex */label:amod/=target < /postag:verb/* matches `amod` to its INVERT expression $\lambda fgx.\exists y.\, f(x) \wedge g(y) \wedge \text{amod}^{\text{i}}(y_e, x_a)$.

[5] In what follows, all references to UD are to UD v1.3.



(a) With long-distance dependency.



(b) With variable binding.

Figure 5.2: The original and enhanced dependency trees for *Anna wants to marry Kristoff*.

substitution, and composition steps described above to rely solely on information encoded in the UD representation. Importantly, UDEPLAMBDA is designed to not rely on lexical forms in a language to assign lambda expressions, but only on information contained in dependency labels and postags.

However, some linguistic phenomena are language specific (e.g. pronoun-dropping) or meaning specific (e.g. *every* and *the* in English have very different semantics, despite being both determiners) and are not encoded in the UD schema. Furthermore, some cross-linguistic phenomena, such as long-distance dependencies, are not part of the core UD representation. To circumvent this limitation, a simple *enhancement* step enriches the original UD representation before binarization takes place (Section 5.3.1). This step adds to the dependency tree missing syntactic information and long-distance dependencies, thereby creating a graph. Whereas DEPLAMBDA is not able to handle graph-structured input, UDEPLAMBDA is designed to work directly with dependency graphs (Section 5.3.2). Finally, the representation of several linguistic constructions differ between UD and SD, which requires different handling in the semantic interface (Section 5.3.3).

5.3.1 Enhancement

Both Schuster & Manning (2016) and Nivre et al. (2016) note the necessity of an enhanced UD representation to enable semantic applications. However, such enhancements are currently only available for a subset of languages in UD. Instead, we rely on a small number of enhancements for our main application—semantic parsing for question-answering—with the hope that this step can be replaced by an enhanced UD representation in the future. Specifically, we define three kinds of enhancements: (1) long-distance dependencies; (2) types of coordination; and (3) refined question word tags.

First, we identify long-distance dependencies in relative clauses and control constructions. We follow Schuster & Manning (2016) and find these using the labels `acl` (relative) and `xcomp` (control). Figure 5.2(a) shows the long-distance dependency in the sentence *Anna wants to marry Kristoff*. Here, *marry* is provided with its missing `nsubj` (dashed arc). Second, UD conflates all coordinating constructions to a single dependency label, `conj`. To obtain the correct coordination scope, we refine `conj` to `conj:verb`, `conj:vp`, `conj:sentence`, `conj:np`, and `conj:adj`, similar to Reddy et al. (2016). Finally, unlike the PTB tags (Marcus et al., 1993) used by SD, the UD part-of-speech tags do not distinguish question words. Since these are crucial to question-answering, we use a small lexicon to refine the tags for determiners (DET), adverbs (ADV) and pronouns (PRON) to DET:WH, ADV:WH and PRON:WH, respectively. Specifically, we use a list of 12 (English), 14 (Spanish) and 35 (German) words, respectively. This is the only part of UDepLambda that relies on language-specific information. We hope that, as the coverage of morphological features in UD improves, this refinement can be replaced by relying on morphological features, such as the interrogative feature (INT).

The complete list of enhancements are listed in Section A.1.

5.3.2 Graph Structures and Bind

To handle graph structures that may result from the enhancement step, such as those in Figure 5.2(a), we propose a variable-binding mechanism that differs from that of DepLambda. First, each long-distance dependency is split into independent arcs as shown in Figure 5.2(b). Here, Ω is a placeholder for the subject of *marry*, which in turn corresponds to *Anna* as indicated by the binding of Ω via the pseudo-label BIND. We treat BIND like an ordinary dependency label with semantics MERGE and process the

resulting tree as usual, via the s-expression:

(nsubj (xcomp wants (nsubj (mark
(dobj marry Kristoff) to) Ω) (BIND Anna Ω)),

with the lambda-expression substitutions:

wants, *marry* ∈ EVENT; *to* ∈ FUNCTIONAL;
Anna, *Kristoff* ∈ ENTITY;
mark ∈ HEAD; BIND ∈ MERGE;
$\text{xcomp} = \lambda fgx.\exists y.\, f(x) \wedge g(y) \wedge \text{xcomp}(x_e, y_e)$.

These substitutions are based solely on unlexicalized context. For example, the part-of-speech tag PROPN of *Anna* invokes an ENTITY expression.

The placeholder Ω has semantics $\lambda x.\text{EQ}(x, \omega)$, where $\text{EQ}(u, \omega)$ is true iff u and ω are equal (have the same denotation), which unifies the subject variable of *wants* with the subject variable of *marry*.

After substitution and composition, we get:

$$\lambda z.\exists xywv.\, \text{wants}(z_e) \wedge \text{Anna}(x_a) \wedge \text{arg}_1(z_e, x_a) \wedge \text{EQ}(x, \omega)$$
$$\wedge\, \text{marry}(y_e) \wedge \text{xcomp}(z_e, y_e) \wedge \text{arg}_1(y_e, v_a) \wedge \text{EQ}(v, \omega)$$
$$\wedge\, \text{Kristoff}(w_a) \wedge \text{arg}_2(y_e, w_a),$$

This expression may be simplified further by replacing all occurrences of v with x and removing the unification predicates EQ, which results in:

$$\lambda z.\exists xyw.\, \text{wants}(z_e) \wedge \text{Anna}(x_a) \wedge \text{arg}_1(z_e, x_a)$$
$$\wedge\, \text{marry}(y_e) \wedge \text{xcomp}(z_e, y_e) \wedge \text{arg}_1(y_e, x_a)$$
$$\wedge\, \text{Kristoff}(w_a) \wedge \text{arg}_2(y_e, w_a).$$

This expression encodes the fact that *Anna* is the arg_1 of the *marry* event, as desired. DEPLAMBDA, in contrast, cannot handle graph-structured input, since it lacks a principled way of generating s-expressions from graphs. Even given the above s-expression, BIND in DEPLAMBDA is defined in a way such that the composition fails to unify v and x, which is crucial for the correct semantics. Moreover, the definition of BIND in DEPLAMBDA does not have a formal interpretation within the lambda calculus, unlike ours.

5.3.3 Linguistic Constructions

Below, we highlight the most pertinent differences between UDEPLAMBDA and DEPLAMBDA, stemming from the different treatment of various linguistic constructions in UD versus SD. The complete list of substitution rules for UD are listed in Section A.3 and Section A.4.

Prepositional Phrases UD uses a content-head analysis, in contrast to SD, which treats function words as heads of prepositional phrases, Accordingly, the s-expression for the phrase *president in 2009* is (nmod president (case 2009 in)) in UDEPLAMBDA and (prep president (pobj in 2009)) in DEPLAMBDA. To achieve the desired semantics,

$$\lambda x.\, \exists y.\, \text{president}(x_a) \wedge \text{president\_event}(x_e) \wedge \\ \text{arg}_1(x_e, x_a) \wedge 2009(y_a) \wedge \text{prep.in}(x_e, y_a)\,,$$

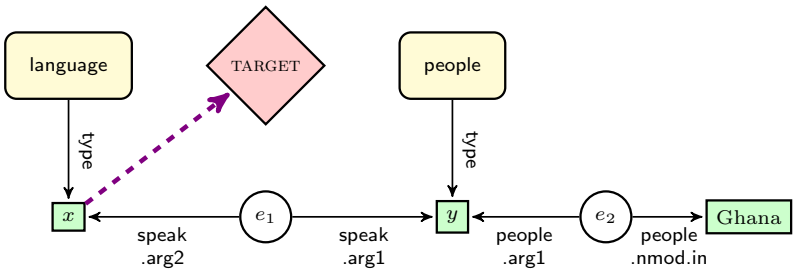
DEPLAMBDA relies on an intermediate logical form that requires some post-processing, whereas UDEPLAMBDA obtains the desired logical form directly through the following entries:

$$\textit{in} \in \text{FUNCTIONAL};\ \textit{2009} \in \text{ENTITY};\ \text{case} \in \text{HEAD};$$
$$\textit{president} = \lambda x.\, \text{president}(x_a) \wedge \text{president\_event}(x_e) \\ \wedge \text{arg}_1(x_e, x_a)\,;$$
$$\text{nmod} = \lambda f g x.\, \exists y.\, f(x) \wedge g(y) \wedge \text{nmod.in}(x_e, y_a)\,.$$

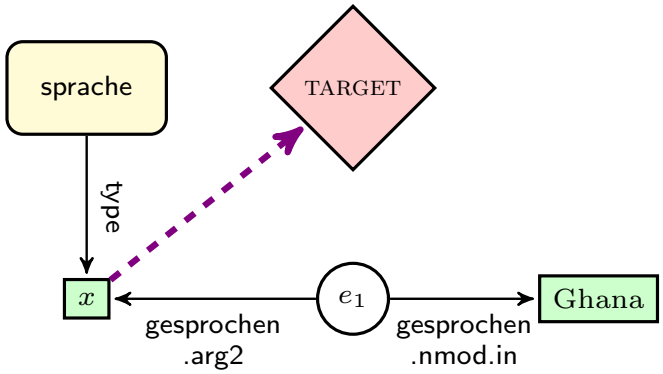
Other `nmod` constructions, such as possessives (`nmod:poss`), temporal modifiers (`nmod:tmod`) and adverbial modifiers (`nmod:npmod`), are handled similarly. Note how the common noun *president*, evokes both entity and event predicates above.

Passives DEPLAMBDA gives special treatment to passive verbs, identified by the fine-grained part-of-speech tags in the PTB tag together with dependency context (see Section 4.3.2). For example, *An Oscar was won* is analyzed as $\lambda x.\, \text{won.pass}(x_e) \wedge \text{Oscar}(y_a) \wedge \text{arg}_1(x_e, y_a)$, where won.pass represents a passive event. However, UD does not distinguish between active and passive forms.[6] While the labels `nsubjpass` or `auxpass` indicate passive constructions, such clues are sometimes missing, such as in reduced relatives. We therefore opt to not have separate entries for passives, but aim to produce identical logical forms for active and passive forms when possible (for example, by treating `nsubjpass` as direct object). With the following entries,

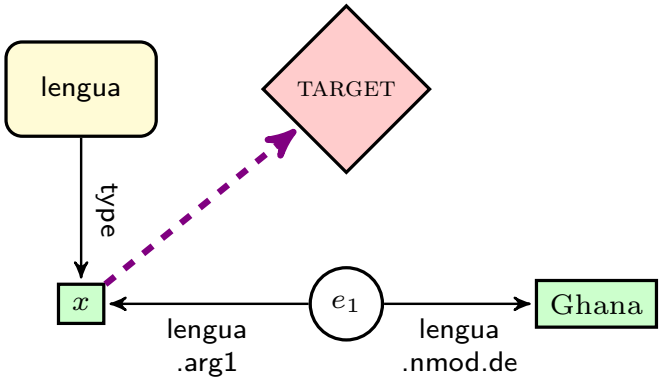
[6] UD encodes voice as a morphological feature, but most syntactic analyzers do not produce this information yet.



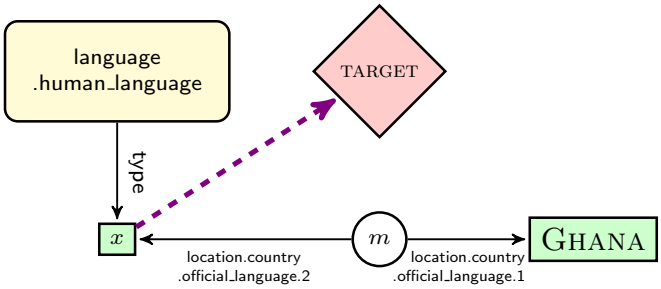
(a) English



(b) German



(c) Spanish



(d) Freebase

Figure 5.3: The ungrounded graphs for *What language do the people in Ghana speak?*, *Welche Sprache wird in Ghana gesprochen?* and *Cuál es la lengua de Ghana?*, and the corresponding grounded graph.

won ∈ EVENT; *an*, *was* ∈ FUNCTIONAL; auxpass ∈ HEAD;
nsubjpass $= \lambda fgx.\exists y.\, f(x) \wedge g(y) \wedge \text{arg}_2(x_e, y_a)$,

the lambda expression for *An Oscar was won* becomes $\lambda x.\,\text{won}(x_e) \wedge \text{Oscar}(y_a) \wedge \text{arg}_2(x_e, y_a)$, identical to that of its active form. However, not having a special entry for passive verbs may have undesirable side-effects. For example, in the reduced-relative construction *Pixar claimed the Oscar won for Frozen*, the phrase *the Oscar won ...* will receive the semantics $\lambda x.\,\text{Oscar}(y_a) \wedge \text{won}(x_e) \wedge \mathbf{arg_1}(x_e, y_a)$, which differs from that of *an Oscar was won*. We leave it to the target application to disambiguate the interpretation in such cases.

Long-Distance Dependencies As discussed in Section 5.3.2, we handle long-distance dependencies evoked by clausal modifiers (`acl`) and control verbs (`xcomp`) with the BIND mechanism, whereas DEPLAMBDA cannot handle control constructions. For `xcomp`, as seen earlier, we use the mapping $\lambda fgx.\exists y.\, f(x) \wedge g(y) \wedge \text{xcomp}(x_e, y_e)$. For `acl` we use $\lambda fgx.\exists y.\, f(x) \wedge g(y)$, to conjoin the main clause and the modifier clause. However, not all `acl` clauses evoke long-distance dependencies, e.g. in *the news that Disney won an Oscar*, the clause *that Disney won an Oscar* is a subordinating conjunction of *news*. In such cases, we instead assign `acl` the INVERT semantics.

Questions Question words are marked with the enhanced part-of-speech tags DET:WH, ADV:WH and PRON:WH, which are all assigned the semantics $\lambda x.\,\$\{\text{word}\}(x_a) \wedge \text{TARGET}(x_a)$. The predicate TARGET indicates that x_a represents the variable of interest, that is the answer to the question.

5.3.4 Limitations

In order to achieve language independence, UDEPLAMBDA has to sacrifice semantic specificity, since in many cases the semantics is carried by lexical information. Consider the sentences *John broke the window* and *The window broke*. Although it is the *window* that broke in both cases, our inferred logical forms do not canonicalize the relation between *broke* and *window*. To achieve this, we would have to make the substitution of `nsubj` depend on lexical context, such that when *window* occurs as `nsubj` with *broke*, the predicate arg_2 is invoked rather than arg_1. We do not address this problem, and leave it to the target application to infer that arg_2 and arg_1 have the same semantic function in these cases. We anticipate that the ability to make such lexicalized semantic inferences

	WebQuestions
en	What language do the people in Ghana speak?
de	Welche Sprache wird in Ghana gesprochen?
es	¿Cuál es la lengua de Ghana?
en	Who was Vincent van Gogh inspired by?
de	Von wem wurde Vincent van Gogh inspiriert?
es	¿Qué inspiró a Van Gogh?
	GraphQuestions
en	NASA has how many launch sites?
de	Wie viele Abschussbasen besitzt NASA?
es	¿Cuántos sitios de despegue tiene NASA?
en	Which loudspeakers are heavier than 82.0 kg?
de	Welche Lautsprecher sind schwerer als 82.0 kg?
es	¿Qué altavoces pesan más de 82.0 kg?

Table 5.1: Example questions and their translations.

in a task-agnostic cross-lingual framework would be highly useful and a crucial avenue for future work on universal semantics.

Other constructions that require lexical information are quantifiers like *every*, *some* and *most*, negation markers like *no* and *not*, and intentional verbs like *believe* and *said*, however these constructions are rare in our evaluation datasets. UD does not have special labels to indicate these. Although not currently implemented, we discuss how to handle quantifiers in this framework in Chapter 6. Fancellu et al. (2017) is a first step in this direction.

5.4 Experimental Setup

To study the multilingual nature of UDepLambda, we conduct an empirical evaluation on question answering against Freebase in three different languages: English, Spanish, and German. We use GraphParser to convert these logical forms to Freebase graphs. To recap, GraphParser first converts logical forms to ungrounded graphs which inturn are converted to Freebase graphs using graph transduction. shows ungrounded

graphs for the same Freebase graph in multiple languages.

We present below the datasets, baseline models and implementation details.

5.4.1 Datasets

We evaluate our approach on WebQuestions and GraphQuestions (Su et al., 2016), a recently released dataset of English questions with both their answers and grounded logical forms. While WebQuestions is dominated by simple entity-attribute questions, GraphQuestions contains a large number of compositional questions involving aggregation (e.g. *How many children of Eddard Stark were born in Winterfell?*) and comparison (e.g. *In which month does the average rainfall of New York City exceed 86 mm?*). The number of training, development and test questions is 2644, 1134, and 2032, respectively, for WebQuestions and 1794, 764, and 2608 for GraphQuestions.

To support multilingual evaluation, we created translations of WebQuestions and GraphQuestions to German and Spanish. For WebQuestions two professional annotators were hired per language, while for GraphQuestions we used a trusted pool of 20 annotators per language (with a single annotator per question). Examples of the original questions and their translations are provided in Table 5.1.

5.4.2 Comparison Systems

We compared UDepLambda to prior work and three versions of GraphParser that operate on different representations: entity cliques, dependency trees, and CCG-based semantic derivations.

SingleEvent As seen in Section 4.4.2, this baseline is agnostic to the syntax of the input question. In this representation, we create a single event to which all entities in the question are connected by the predicate arg_1. An additional TARGET node is connected to the event by the predicate arg_0. Note that this cannot represent any compositional structure involving aggregation or multihop reasoning.

DepTree An ungrounded graph is obtained directly from the original dependency tree. An event is created for each parent and its dependents in the tree. Each dependent is linked to this event with an edge labeled with its dependency relation, while the parent is linked to the event with an edge labeled arg_0. If a word is a question word, an additional TARGET predicate is attached to its entity node.

k	WebQuestions			GraphQuestions		
	en	de	es	en	de	es
1	89.6	82.8	86.7	47.2	39.9	39.5
10	95.7	91.2	94.0	56.9	48.4	51.6

Table 5.2: Structured perceptron k-best entity linking accuracies on the development sets.

CCGGRAPH This is the CCG-based semantic representation introduced in Chapter 3. Note that this baseline exists only for English.

5.4.3 Implementation Details

Here we provide details on the syntactic analyzers employed, entity resolution details, and the features used by the grounding model.

Dependency Parsing The English, Spanish, and German Universal Dependencies (UD) treebanks (v1.3; Nivre et al 2016) were used to train part of speech taggers and dependency parsers. We used a bidirectional LSTM tagger (Plank et al., 2016) and a bidirectional LSTM shift-reduce parser (Kiperwasser & Goldberg, 2016). Both the tagger and parser require word embeddings. For English, we used GloVe embeddings (Pennington et al., 2014) trained on Wikipedia and the Gigaword corpus.[7] For German and Spanish, we used SENNA embeddings (Collobert et al., 2011; Al-Rfou et al., 2013) trained on Wikipedia corpora (589M words German; 397M words Spanish).[8] Measured on the UD test sets, the tagger accuracies are 94.5 (English), 92.2 (German), and 95.7 (Spanish), with corresponding labeled attachment parser scores of 81.8, 74.7, and 82.2.

Entity Resolution As with previous chapters, we perform the entity annotation using Freebase API (Section 2.11). However, due to the recent Freebase API shutdown, we used the KG API for GraphQuestions. We observed that this leads to inferior entity linking results compared to those of Freebase. Table 5.2 shows the 1-best and 10-best entity disambiguation F_1-scores for each language and dataset.

[7] http://nlp.stanford.edu/projects/glove/.
[8] https://sites.google.com/site/rmyeid/projects/polyglot.

Method	WebQuestions en	WebQuestions de	WebQuestions es	GraphQuestions en	GraphQuestions de	GraphQuestions es
SINGLEEVENT	47.6	43.9	45.0	15.9	8.3	11.2
DEPTREE	47.8	43.9	44.5	15.8	7.9	11.0
CCGGRAPH	48.4	–	–	15.9	–	–
UDEPLAMBDA	48.3	44.2	45.7	17.6	9.0	12.4

Table 5.3: F_1-scores on the test for models trained on the training set (excluding the development set).

Features We use the features described in Section 2.10.5: *basic* features of words and Freebase relations, and *graph* features crossing ungrounded events with grounded relations, ungrounded types with grounded relations, and ungrounded answer type crossed with a binary feature indicating if the answer is a number. In addition, we add features encoding the *semantic* similarity of ungrounded events and Freebase relations. Specifically, we used the cosine similarity of the translation-invariant embeddings of Huang et al. (2015).[9]

5.5 Results

Table 5.3 shows the performance of GRAPHPARSER with these different representations. We use average F_1-score of predicted answers (Section 2.12) as the evaluation metric for both WebQuestions and GraphQuestions. We first observe that UDEPLAMBDA consistently outperforms the SINGLEEVENT and DEPTREE representations in all languages.[10]

For English, performance is almost on par with CCGGRAPH, which suggests that UDEPLAMBDA does not sacrifice too much specificity for universality. With both datasets, results are lower for German compared to Spanish. This agrees with the lower performance of the syntactic parser on the German portion of the UD treebank. Finally, while these results confirm that GraphQuestions is much harder compared to WebQuestions, we note that both datasets predominantly contain single-hop questions, as

[9] http://128.2.220.95/multilingual/data/.

[10] For the DEPTREE model, we CONTRACT each multi-hop path between the question word and an entity to a single edge. Without this constraint, DEPTREE F_1 results are 45.5 (en), 42.9 (de), and 44.2 (es) on WebQuestions, and 11.0 (en), 6.6 (de), and 2.6 (es) on GraphQuestions.

Method	GraphQ.	WebQ.
SEMPRE (Berant et al., 2013)	10.8	35.7
JACANA (Yao & Van Durme, 2014)	5.1	33.0
PARASEMPRE (Berant & Liang, 2014)	12.8	39.9
QA (Yao, 2015)	–	44.3
AQQU (Bast & Haussmann, 2015)	–	49.4
AGENDAIL (Berant & Liang, 2015)	–	49.7
DEPLAMBDA (Chapter 4)	–	50.3
STAGG (Yih et al., 2015)	–	48.4 (52.5)
BILSTM (Türe & Jojic, 2016)	–	24.9 (52.2)
MCNN (Xu et al., 2016)	–	47.0 (53.3)
AGENDAIL-RANK (Yavuz et al., 2016)	–	51.6 (52.6)
UDEPLAMBDA	17.6	49.5

Table 5.4: F_1-scores on the English GraphQuestions and WebQuestions test sets (results with additional task-specific resources in parentheses). Following prior work, for WebQuestions the union of the training and development sets were used for training.

indicated by the competitive performance of SINGLEEVENT on both datasets. GraphQuestions is harder due to the aggregation and superlative questions.

Table 5.4 compares UDEPLAMBDA with previously published models which exist only for English and have been mainly evaluated on WebQuestions. These are either symbolic like ours (first block) or employ neural networks (second block). Results for models using additional task-specific training resources, such as ClueWeb09, Wikipedia, or SimpleQuestions (Bordes et al., 2015) are shown in parentheses. On GraphQuestions, we achieve a new state-of-the-art result with a gain of 4.8 F_1-points over the previously reported best result. On WebQuestions we are 2.1 points below the best model using comparable resources, and 3.8 points below the state of the art. When compared with DEPLAMBDA, UDEPLAMBDA is 0.8 point lower in F_1-score. We attribute this difference to use of the more fine-grained Stanford Dependencies as compared to UD.

5.6 Related Work

The literature is rife with attempts to develop semantic interfaces for HPSG (Copestake et al., 2005), LFG (Kaplan & Bresnan, 1982; Dalrymple et al., 1995; Crouch & King, 2006), TAG (Kallmeyer & Joshi, 2003; Gardent & Kallmeyer, 2003; Nesson & Shieber, 2006), and CCG (Steedman, 2000; Baldridge & Kruijff, 2002; Bos et al., 2004; Artzi et al., 2015). Unlike existing semantic interfaces, UDepLambda (like DepLambda) uses dependency syntax, taking advantage of recent advances in multilingual parsing (McDonald et al., 2013; Nivre et al, 2016).

A common trend in previous work on semantic interfaces is the reliance on rich typed feature structures or semantic types coupled with strong type constraints, which can be very informative but unavoidably language specific. Creating rich semantic types from dependency trees which lack a typing system would be labor intensive and brittle in the face of parsing errors. Instead, UDepLambda relies on generic unlexicalized information present in dependency treebanks and uses a simple type system (one type for dependency labels, and one for words) along with a combinatory mechanism, which avoids type collisions. Earlier attempts at extracting semantic representations from dependencies have mainly focused on language-specific dependency representations (Spreyer & Frank, 2005; Simov & Osenova, 2011; Hahn & Meurers, 2011; Reddy et al., 2016; Falke et al., 2016; Beltagy, 2016), and multi-layered dependency annotations (Jakob et al., 2010; Bédaride & Gardent, 2011). In contrast, UDepLambda derives semantic representations for multiple languages in a common schema directly from Universal Dependencies. This work parallels a growing interest in creating other forms of multilingual semantic representations (Akbik et al., 2015; Vanderwende et al., 2015; White et al., 2016; Evang & Bos, 2016).

The initiatives which share our multilingual spirit are HPSG Grammar Matrix (Bender et al., 2002) and LFG ParGram (Butt et al., 2002). However these also aim to model syntax and not just the semantic interface, whereas in our work we take syntax for granted. Our work also relates to literature on parsing multiple languages to a common executable representation (Cimiano et al., 2013; Haas & Riezler, 2016). However, existing approaches (Kwiatkowksi et al., 2010; Jones et al., 2012; Jie & Lu, 2014) still map to the target meaning representations (more or less) directly unlike using general-purpose syntax.

5.7 Discussion

We introduced UDEPLAMBDA, a semantic interface for Universal Dependencies, and showed that the resulting semantic representation can be used for Freebase semantic parsing multiple languages. We provided translations of benchmark datasets in German and Spanish, in the hope to stimulate further multilingual research on semantic parsing and question answering in general. We have only scratched the surface when it comes to applying UDEPLAMBDA to natural language understanding tasks. In the future, we would like to explore how this framework can benefit other tasks such as summarization and machine translation.

Chapter 6

Quantifiers and Scope in UDepLambda

This chapter provides an outline of how quantification can be incorporated in the U-DepLambda framework.

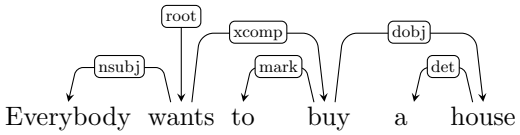
Consider the sentence *Everybody wants to buy a house*,[1] whose dependency tree in the Universal Dependencies (UD) formalism is shown in Figure 6.1(a). This sentence has two possible readings: either (1) every person wants to buy a different house; or (2) every person wants to buy the same house. The two interpretations correspond to the following logical forms:

(1) $\forall x.\,\text{person}(x_a) \rightarrow$
$[\exists zyw.\,\text{wants}(z_e) \wedge \text{arg}_1(z_e, x_a) \wedge \text{buy}(y_e) \wedge \text{xcomp}(z_e, y_e) \wedge$
$\text{house}(w_a) \wedge\ \text{arg}_1(z_e, x_a) \wedge \text{arg}_2(z_e, w_a)]$;

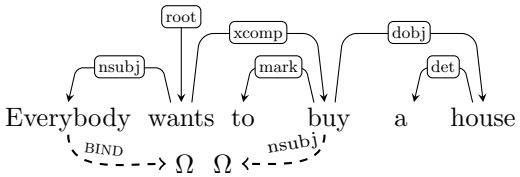
(2) $\exists w.\,\text{house}(w_a) \wedge (\forall x.\,\text{person}(x_a) \rightarrow$
$[\exists zy.\,\text{wants}(z_e) \wedge \text{arg}_1(z_e, x_a) \wedge \text{buy}(y_e) \wedge \text{xcomp}(z_e, y_e) \wedge$
$\text{arg}_1(z_e, x_a) \wedge \text{arg}_2(z_e, w_a)])$.

In (1), the existential variable w is in the scope of the universal variable x (i.e. the *house* is dependent on the *person*). This reading is commonly referred to as the *surface reading*. Conversely, in (2) the universal variable x is in the scope of the existential variable w (i.e. the *house* is independent of the *person*). This reading is also called *inverse reading*. Our goal is to obtain the surface reading logical form in (1) with U-DepLambda. We do not aim to obtain the inverse reading, although this is possible with the use of Skolemization (Steedman, 2012).

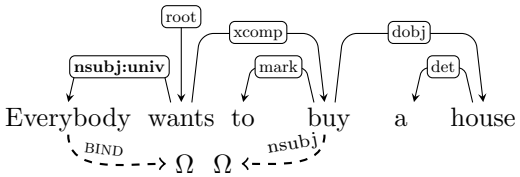
[1] Example borrowed from Schuster & Manning (2016).



(a) Original dependency tree.



(b) Enhanced dependency tree.



(c) Enhanced dependency tree with universal quantification.

Figure 6.1: The dependency tree for *Everybody wants to buy a house* and its enhanced variants.

In UDepLambda, lambda expressions for words, phrases and sentences are all of the form $\lambda x. \ldots$. But from (1), it is clear that we need to express variables bound by quantifiers, e.g. $\forall x$, while still providing access to x for composition. This demands a change in the type system since the same variable cannot be lambda bound and quantifier bound—that is we cannot have formulas of the form $\lambda x \ldots \forall x \ldots$. In this chapter, we first derive the logical form for the example sentence using the type system from the previous chapter (Section 6.1) and show that it fails to handle universal quantification. We then modify the type system slightly to allow derivation of the desired surface reading logical form (Section 6.2). This modified type system is a strict generalization of the original type system.[2] Fancellu et al. (2017) present an elaborate discussion on the modified type system, and how it can handle negation scope and its interaction with universal quantifiers.

[2]Note that this treatment has yet to be added to our implementation, which can be found at https://github.com/sivareddyg/udeplambda.

6.1 Universal Quantification with Original Type System

We will first attempt to derive the logical form in (1) using the default type system of UDEPLAMBDA. Figure 6.1(b) shows the enhanced dependency tree for the sentence, where BIND has been introduced to connect the implied `nsubj` of *buy* (BIND is explained in the main paper in Section 3.2). The s-expression corresponding to the enhanced tree is:

(nsubj (xcomp wants (mark
(nsubj (dobj buy (det house a)) Ω) to))
(BIND everybody Ω)) .

With the following substitution entries,

wants, *buy* $\in$ EVENT;
everybody, *house* $\in$ ENTITY;
a, *to* $\in$ FUNCTIONAL;
$\Omega = \lambda x.\,\mathrm{EQ}(x, \omega)$;
`nsubj` $= \lambda fgx.\,\exists y.\,f(x) \wedge g(y) \wedge \mathrm{arg}_1(x_e, y_a)$;
`dobj` $= \lambda fgx.\,\exists y.\,f(x) \wedge g(y) \wedge \mathrm{arg}_2(x_e, y_a)$;
`xcomp` $= \lambda fgx.\,\exists y.\,f(x) \wedge g(y) \wedge \mathrm{xcomp}(x_e, y_a)$;
`mark` $\in$ HEAD;
BIND $\in$ MERGE,

the lambda expression after composition becomes:

$$\lambda z.\ \exists xywv.\,\mathrm{wants}(z_e) \wedge \mathrm{everybody}(x_a) \wedge \mathrm{arg}_1(z_e, x_a) \wedge \mathrm{EQ}(x, \omega) \wedge \mathrm{buy}(y_e) \wedge \mathrm{xcomp}(z_e, y_e) \wedge \mathrm{arg}_1(y_e, v_a) \wedge \mathrm{EQ}(v, \omega) \wedge \mathrm{arg}_1(x_e, y_a) \wedge \mathrm{house}(w_a) \wedge \mathrm{arg}_2(y_e, w_a)\,.$$

This expression encodes the fact that x and v are in unification, and can thus be further simplified to:

$$(3)\ \lambda z.\,\exists xyw.\,\mathrm{wants}(z_e) \wedge \mathrm{everybody}(x_a) \wedge \mathrm{arg}_1(z_e, x_a) \wedge \mathrm{buy}(y_e) \wedge \mathrm{xcomp}(z_e, y_e) \wedge \mathrm{arg}_1(y_e, x_a) \wedge \mathrm{arg}_1(x_e, y_a) \wedge \mathrm{house}(w_a) \wedge \mathrm{arg}_2(y_e, w_a)\,.$$

However, the logical form (3) differs from the desired form (1). As noted above, UDEPLAMBDA with its default type, where each s-expression must have the type $\eta = \mathbf{Ind} \times \mathbf{Event} \to \mathbf{Bool}$, cannot handle quantifier scoping.

6.2 Universal Quantification with Higher-order Type System

Following Champollion (2010), we make a slight modification to the type system. Instead of using expressions of the form $\lambda x. \ldots$ for words, we use either $\lambda f. \exists x. \ldots$ or $\lambda f. \forall x. \ldots$, where f has type η. As argued by Champollion, this higher-order form makes quantification and negation handling sound and simpler in Neo-Davidsonian event semantics. Following this change, we assign the following lambda expressions to the words in our example sentence:

$$
\begin{aligned}
&\textit{everybody} = \lambda f. \forall x. \text{person}(x) \rightarrow f(x)\,;\\
&\textit{wants} = \lambda f. \exists x. \text{wants}(x_e) \wedge f(x)\,;\\
&\textit{to} = \lambda f. \text{TRUE}\,;\\
&\textit{buy} = \lambda f. \exists x. \text{buy}(x_e) \wedge f(x)\,;\\
&a = \lambda f. \text{TRUE}\,;\\
&\textit{house} = \lambda f. \exists x. \text{house}(x_a) \wedge f(x)\,;\\
&\Omega = \lambda f. f(\omega)\,.
\end{aligned}
$$

Here *everybody* is assigned universal quantifier semantics. Since the UD representation does not distinguish quantifiers, we need to rely on a small (language-specific) lexicon to identify these. To encode quantification scope, we enhance the label `nsubj` to `nsubj:univ`, which indicates that the subject argument of *wants* contains a universal quantifier, as shown in Figure 6.1(c).

This change of semantic type for words and s-expressions forces us to also modify the semantic type of dependency labels, in order to obey the single-type constraint of DepLambda. Thus, dependency labels will now take the form $\lambda PQf. \ldots$, where P is the parent expression, Q is the child expression, and the *return expression* is of the form $\lambda f. \ldots$. Following this change, we assign the following lambda expressions to dependency labels:

$$
\begin{aligned}
&\texttt{nsubj:univ} = \lambda PQf. Q(\lambda y. P(\lambda x. f(x) \wedge \text{arg}_1(x_e, y_a)))\,;\\
&\texttt{nsubj} = \lambda PQf. P(\lambda x. f(x) \wedge Q(\lambda y. \text{arg}_1(x_e, y_a)))\,;\\
&\texttt{dobj} = \lambda PQf. P(\lambda x. f(x) \wedge Q(\lambda y. \text{arg}_2(x_e, y_a)))\,;\\
&\texttt{xcomp} = \lambda PQf. P(\lambda x. f(x) \wedge Q(\lambda y. \text{xcomp}(x_e, y_a)))\,;\\
&\texttt{det}, \texttt{mark} = \lambda PQf. P(f)\,;\\
&\text{BIND} = \lambda PQf. P(\lambda x. f(x) \wedge Q(\lambda y. \text{EQ}(y, x)))\,.
\end{aligned}
$$

Notice that the lambda expression of `nsubj:univ` differs from `nsubj`. In the former, the lambda variables inside Q have wider scope over the variables in P (i.e. the universal quantifier variable of *everybody* has scope over the event variable of *wants*) contrary to the latter.

The new s-expression for Figure 6.1(c) is

(nsubj:univ (xcomp wants (mark
(nsubj (dobj buy (det house a)) Ω) to))
(BIND everybody Ω)) .

Substituting with the modified expressions, and performing composition and simplification leads to the expression:

$$\begin{aligned}(6)\ \lambda f.\forall x\,.\,&\text{person}(x_a) \rightarrow \\ &[\exists zyw.\, f(z) \wedge \text{wants}(z_e) \wedge \text{arg}_1(z_e, x_a) \wedge \text{buy}(y_e) \\ &\quad \wedge \text{xcomp}(z_e, y_e) \wedge \text{house}(w_a) \\ &\quad \wedge \text{arg}_1(z_e, x_a) \wedge \text{arg}_2(z_e, w_a)]\,.\end{aligned}$$

This expression is identical to (1) except for the outermost term λf. By applying (6) to $\lambda x.\text{TRUE}$, we obtain (1), which completes the treatment of universal quantification in UDepLambda.

Chapter 7

Conclusions

In this thesis, we tackled one of the important questions of NLP, namely whether existing general-purpose syntactic representations are useful for natural language understanding. Our hypothesis in this thesis was that *general-purpose syntax helps in deriving superior task-agnostic and task-specific meaning representations than induced or latent syntax*. Specifically, we proposed a framework to exploit syntactic representations for Freebase semantic parsing. Our framework factorized the semantic parsing problem as a linguistically motivated structured prediction problem, i.e., syntactic parsing, and then as a domain-specific transduction problem. Our main motivation is that the NLP community has developed several resources for wide-coverage syntactic parsing, and it is beneficial to build on top of these rather than learning syntax from scratch based on application specific supervision signals. This is particularly relevant for semantic parsing tasks since collecting training data for these is expensive, and the available limited data can be used for learning to convert linguistic structures that are universal across languages to task-specific structures without wasting resources on figuring out what the linguistic structures are. The experiments in Chapter 3, Chapter 4 and Chapter 5 indicate that general-purpose syntax is indeed useful for semantic parsing.

Our contributions are as follows: We showed how to derive general-purpose meaning representations from CCG structures (Chapter 3) and dependencies (Chapter 4). We showed that these meaning representations can be converted to grounded meaning representations using the framework outlined in Chapter 2. Our experimental results in Chapter 3, Chapter 4 and Chapter 5 show that general-purpose meaning representations are more useful than meaning representations from induced syntax (UBL and KCAZ13) and syntax-agnostic representations (SINGLEEVENT). This indicates our hypothesis is true within the experimental setup presented in this thesis: linear model

(Section 2.10), traditional beam search parsing (Section 2.10.1), and for datasets Free-917, GraphQuestions and weakly for WebQuestions. This field is changing rapidly and recent methods show it is possible to build better semantic parsing algorithms that ignore syntax but these assume certain experimental conditions (see discussion on finding 4 below). We summarize the findings of this thesis as follows:

1. General-purpose syntax can derive general-purpose meaning representations.
2. General-purpose syntax should not be used to derive task-specific meaning representations directly.
3. General-purpose meaning representations can derive task-specific meaning representations.
4. Syntax helps in guiding search and extracting good features.

General-purpose syntax can derive general-purpose meaning representations In this thesis, we used two syntactic representations, CCG and dependency syntax, to derive general-purpose meaning representations. There is a vast literature on CCG (Steedman, 1996, 2000, 2012) showing that rich logical forms can be derived from syntax, not only with predicate-argument structures but also with quantifier and negation scope, and sentence-level coreference. Some phenomena like intentionality, and discourse level coreference are yet to be studied in detail since existing implementations have had only limited success so far (Bos, 2008b). In this work, we only used CCG to extract logical forms without quantifier and negation scope since the evaluation datasets did not demand scope.

We presented DEPLAMBDA and UDEPLAMBDA to extract logical forms from dependencies. Although we only evaluated the logical forms without scope information, we presented a higher-order type system to handle quantifier scope for completeness (Chapter 6). Recent work shows negation scope and its interaction with quantifiers can be handled in UDEPLAMBDA (Fancellu et al., 2017). A current limitation of UDEPLAMBDA is its inability to handle context-sensitive dependency labels due to the constraint that it should not use lexical information (Section 5.3.4). This constraint has to be relaxed to handle wide-coverage semantics. More work is also necessary to study the expressive power of UDEPLAMBDA, e.g., whether it can produce all logical forms that CCG can produce.

Other alternative syntactic formalisms for extracting semantics are LFG (Kaplan & Bresnan, 1982), HPSG (Pollard & Sag, 1994) and TAG (Joshi & Schabes, 1997), but these are outside the scope of this thesis.

General-purpose syntax should not be used to derive task-specific meaning representations directly As we saw with the DEPTREE model in Chapter 4 and Chapter 5, transducing syntax to target meaning representation is worse than going through a general-purpose meaning representation. As the complexity of the target representation increases (e.g., quantifier scope), this approach may become infeasible.

But one question that remains is whether it is possible learn lambda calculus that converts general-purpose CCG and dependency syntax to a target representation. This is feasible if the target representation is within the expressive power of these formalisms. Consider a hypothetical scenario where a robot (e.g., a pawn movements in a board game) only understands the following meaning representation language of the word: fbf, $fbffbffbf$, $fbffbffbffbffbffbffbffbffbffbf$, ..., where f and b represent $forward$ and *backward* respectively. Vijay-Shanker & Weir (1994) show that no CCG grammar can generate this language. It would be pointless to aim to generate this meaning representation from CCG. Whereas if one relies on an intermediate representation, one can define additional transduction operations that increase the expressive power of the original grammar by modifying the intermediate representation.

Even when a formalism can express the target meaning representation, an important consideration is whether the grammar at hand can produce the target meaning representation. For example, a grammar for strings $\{ww : w \in (a,b)^*\}$ cannot produce the strings in $\{ww^r : w \in (a,b)^*\}$ where w^r is the reverse string of w. As seen in Table 3.3, Table 4.1 and Table 4.2, there are structures in Freebase that our default grammar could not produce (see the column -C -E).

General-purpose meaning representations can derive task-specific meaning representations As noted above, transduction operations can be defined on general-purpose representations in order to map to a target meaning representation. However this will be felicitous only if the general-purpose representation is mostly isomorphic to the target representation, which is the case with Freebase (see -C -E column of Table 4.1 and Table 4.2). Moreover, from a practical perspective, only a bounded number of transduction operations should be used in order to restrict the search space of the possible target graphs.

Syntax helps in guiding search and extracting good features The main advantage of using syntax-based meaning representations is that they guide the search procedure limiting the target search space. In weakly supervised scenarios such as question and

answer pairs, this is ideal. Consider the question *what are the capitals of the states bordering Texas?*. The general-purpose logical form from syntax will indicate there are two hops between the answer entity and *Texas*. Accordingly, while searching for the target graph, we can restrict our search within a few hops from *Texas*. This is even more useful in cases involving additional operations such as aggregation, comparison, and superlative constructions. Consider the question *what are the highest mountains of the states bordering Texas?*. Its general-purpose logical form will indicate first get the states bordering *Texas*, and then get all the mountains in these states, and finally return the highest mountain of each state. Although the ungrounded structure (i.e., from the logical form) and the target graph corresponding to this question may not be isomorphic, it provides clues to what kind of graphs to search for and their approximate matches. This saves us from exploring the entire target search space. Unfortunately, our evaluation datasets are not compositional enough to convincingly show that syntax-based methods are beneficial for search. As seen in Table 4.1, a simple template-based method like SINGLEEVENT gets a high oracle score, indicating that the datasets lack structural diversity.

Even then we find that syntax can help in other ways, namely in extracting rich features which are indicative of the semantics, e.g., predicates in the ungrounded logical forms have a strong association with relations in Freebase. When there is limited variation of structures in the input, models that exploit domain-specific knowledge (templates) along with features from syntax might be ideal.

Work on semantic parsing has grown rapidly over the last few years. Neural models which do not use any syntactic features have been shown to perform almost as well and in some cases better compared to our models (Dong & Lapata, 2016; Jia & Liang, 2016; Kočiský et al., 2016). Although these models are capable of memorizing frequent compositional forms, a key challenge is modeling unseen compositions. The absence of explicit compositional mechanisms make this hard.

We discuss in the next section some avenues for future work which might prove the importance of syntax-mediated semantic parsing.

7.1 Future Work

Datasets Existing datasets for open-domain semantic parsing are non-compositional, small and structurally not diverse. Current work in building datasets for semantic parsing focuses on small closed domains (Wang et al., 2015b). In order to test if linguistic

theories of syntax-semantics help, or to design better semantic parsing algorithms that scale, building a large compositional open-domain dataset is crucial.

Automating steps in UDepLambda In this work, we hand-coded the lambda calculus rules of UDepLambda to produce simple logical forms. In order to scale this to wide-coverage semantics, the manual rule-based component should be replaced with a model that learns these rules from general-purpose meaning representations like AMR (Banarescu et al., 2013), PropBank (Palmer et al., 2005) or the Groningen Meaning Bank (Basile et al., 2012; Abzianidze et al., 2017).

Expressive power of UDepLambda In this thesis, UDepLambda's expressivity is limited to propositional logical forms, i.e., logical forms without quantifiers or negation scope. Although Chapter 6 and Fancellu et al. (2017) show UDepLambda can handle universal and negation scoping, the bounds of its expressive power are not clear. Evaluating UDepLambda on wide-coverage logical forms with scope would reveal limitations and problems unforeseen.

Lexicalization of UDepLambda The strength of UDepLambda is that it can parse any new language in UD to logical form. This is possible because UDepLambda does not use lexical information. Due to this it cannot handle lexicalized semantic phenomenon as discussed in Section 5.3.4. We must relax this constraint in order to produce accurate semantics. Additionally, the strength of UDepLambda lies in its robust single-type system. It is not clear if single-type limits expressivity. Relaxing the type system to allow multiple types like in CCG would render UDepLambda at least as expressive as CCG.

A single semantic parsing model for multiple languages UDepLambda produces logical forms for all languages using the same lambda calculus. In Chapter 5, we used these logical forms to train different models for English, German and Spanish in order to do the grounding. We do not foresee any difficulties in training one single grounding model for all languages, e.g., in our linear model, some features such as ungrounded and grounded predicate similarity can be shared across languages, and certain features such as association of ungrounded and grounded predicates can be language specific.

Joint modeling of dependencies and semantic parsing Like previous work on jointly learning CCG syntax and semantics (Zettlemoyer & Collins, 2005, 2007; Kwiatkowksi et al., 2010), a model that jointly learns to dependency parse and semantic parse would be feasible with UDepLambda. This would enable learning domain-specific dependency parsers and semantic parsers simultaneously. The dependency parser would help in interpreting the grounding decisions.

Evaluation on other semantic tasks Besides semantic parsing, logical forms are shown to be useful for many tasks exclusively focused on English such as other tasks such as entailment (Beltagy et al., 2016), machine-reading (Sachan & Xing, 2016), text-based question answering (Lewis & Steedman, 2013), sentence simplification (Narayan & Gardent, 2014), summarization (Liu et al., 2015), paraphrasing (Pavlick et al., 2015), and relation extraction (Rocktäschel et al., 2015). We anticipate UDepLambda logical forms would be useful for these tasks in multiple languages too.

7.2 Final Remarks

In this thesis, we addressed a long-standing question regarding the importance of general-purpose syntax for monolingual and multilingual semantic parsing. In addition to this, our major contribution is a new theory of semantics for dependency syntax. Universal Dependencies is the result of several thousands of expert human hours, and our work shows its practicality for a semantic task. Technical contributions aside, we hope the code will help in utilizing dependency syntax for semantic tasks beyond the scope of thesis.

Semantic parsing is not only an important research problem, but also a practically useful one. Given that it reduces the cognitive burden in using computing devices, it is welcoming news to many people who own such devices. Better semantic parsing means more device interaction, and more interaction means more data that helps improving semantic parsing. Eventually, semantic parsing is going to be vital for natural interaction between humans and computers.

Appendix A

A.1 Enhancement Rules for Universal Dependencies

```
rulegroup {
 name: "Question words"
 priority: 1
 rule {
  name: "question words invoking COUNT"
  priority: 1
  # Cu\'{a}ntas ciudades
  # Howmany cities (how many is a single word in spanish)
  # change the postag of from ADV to ADV:COUNT:WH.
  tregex: "/w-.*-(?:cu\'{a}nto|cu\'{a}nta|cu\'{a}ntos|cu\'{a}ntas
  |cuanto|cuanta|cuantos|cuantas)$/=word !>> /^l-acl.*$/ $
      /t-(?:PRON|DET|ADV|ADJ)/=postag"
  transformation {
   target: "postag"
   action: CHANGE_LABEL
   label: "{postag}:COUNT:WH"
  }
 }
 rule {
  name: "Enhance question words pos tags"
  priority: 5
 # "when" did aldi originate?
  # change the postag of "when" from ADV to ADV:WH.
  tregex: "/w-.*-(?:what|when|who|where|which|how|whom|whose|why
  |qui\'{e}n|qu\'{e}|d\'{o}nde|cu\'{a}ndo|cu\'{a}nto|cu\'{a}nta
  |cu\'{a}ntos|cu\'{a}ntas|cu\'{a}l|qui\'{e}nes|cu\'{a}les|c\'{o}mo
  |ad\'{o}nde|porqu\'{e}|quien|que|donde|cuando|cuanto|cuanta
  |cuantos|cuantas|cual|quienes|cuales|como|adonde|porque|wer
  |welche|welcher|wen|wem|welchen|welchem|wessen|was
```

```
  |welches|warum|weshalb|weswegen|wieso|wie
  |wof\"{u}r|wofuer|wozu|womit|wodurch|worum|wor\"{u}ber
  |worueber|wobei|wovon|wo|wohin|woher|woran|worin|worauf
  |worunter|wovor|wohinter|wann|wan|wieviele
  |wieviel|wievele|wievievel)$/=word !>> /^l-acl.*$/ $
      /t-(?:PRON|DET|ADV|ADJ)/=postag"
  transformation {
   target: "postag"
   action: CHANGE_LABEL
   label: "{postag}:WH"
  }
}
# Below are special cases when COUNT phenomenon requires wideer context
rule {
  name: "nmod indicating counting"
  # "number" of "people"
  # "count" of "films"
  priority: 10
  tregex: "/^l-nmod$/=relation $
      /^w-.*-(count|number|total|anzahl|n\'{u}mero|numero)$/ $ /^t-NOUN$/
      < /^l-case$/"
  transformation {
   target: "relation"
   action: CHANGE_LABEL
   label: "{relation}:count"
  }
}
rule {
  name: "advmod in how many"
  # "how" "many"
  priority: 1
  tregex: "/^l-advmod$/=relation < /^w-.*-(?:how|wie)$/ $
      /^w-.*-(?:many|viel|viele|vielen)$/"
  transformation {
   target: "relation"
   action: CHANGE_LABEL
   label: "{relation}:count"
  }
}
rule {
  name: "advmod in how many"
  # "how" many "people"
```

```
67    # how attached to people instead of many. This is likely a parser
          mistake.
68    priority: 1
69    tregex: "/^l-advmod$/=relation < /^w-.*-(?:how|wie)$/ $ (/^l-.*$/ <
          /^w-.*-(?:many|viel|viele|vielen)$/)"
70    transformation {
71      target: "relation"
72      action: CHANGE_LABEL
73      label: "{relation}:count"
74    }
75   }
76   # Questions without question words
77   rule {
78    name: "dobj when posed as a question"
79    # "give" me the "capital" of UK?
80    # This happens when subj is not seen with the verb.
81    priority: 5
82    tregex: "/^l-dobj$/=relation !$ /^l-(?:nsubj|nsubjpass)$/ $
          /^w-.*-(?:find|give|list|locate|look|name|tell|finden|finde
83    |geben|gib|liste|suchen|schauen|nennen|erz\"{a}hlen|encontrar
84    |encontrarnos|bien|dame|darnos|dar|lista|localizar|localiza
85    |localizarme|mira|mirame|buscar|nombre|nombrar
86    |dime|cu\'{e}ntanos)$/ $ /^t-VERB$/"
87    transformation {
88      target: "relation"
89      action: CHANGE_LABEL
90      label: "{relation}:wh"
91    }
92   }
93 }
94 rulegroup {
95   name: "Common Dependency Parser Errors. Use sparingly."
96   priority: 1
97   rule {
98    name: "questions wrongly tagged as mark"
99    priority: 1
100  # "when" did aldi originate?
101   # change the dependency relation of "when" to "advmod".
102   # (l-root w-4-originate t-VERB (l-mark w-1-when t-ADV) (l-aux w-2-did
          t-AUX) (l-nsubj w-3-aldi t-PROPN) (l-punct w-5-? t-PUNCT))
103   tregex: "/^l-mark$/=mark < (/w-.*-(?:what|when|who|where|which|how|whom|
104   whose|why|qui\'{e}n|qu\'{e}|d\'{o}nde|cu\'{a}ndo|cu\'{a}nto
```

```
  |cu\’{a}nta|cu\’{a}ntos|cu\’{a}ntas|cu\’{a}l|qui\’{e}nes
  |cu\’{a}les|c\’{o}mo|ad\’{o}nde|porqu\’{e}|quien|que
  |donde|cuando|cuanto|cuanta|cuantos|cuantas|cual
  |quienes|cuales|como|adonde|porque|wer|welche|welcher
  |wen|wem|welchen|welchem|wessen|was|welches|warum
  |weshalb|weswegen|wieso|wie|wof\"{u}r|wofuer|wozu
  |womit|wodurch|worum|wor\"{u}ber|worueber|wobei|wovon
  |wo|wohin|woher|woran|worin|worauf|worunter|wovor
  |wohinter|wann|wan|wieviele|wieviel|wievele|wievievel)$/=qword $
      /t-(?:PRON|DET|ADV|ADJ).*$/) > /^l-root$/"
  transformation {
   target: "mark"
   action: CHANGE_LABEL
   label: "l-advmod"
  }
 }
 rule {
  name: "questions wrongly parsed as advcl"
  priority: 1
# "Where" to "hang" out in Chicago?
  # (l-root w-1-where t-ADV (l-advcl w-3-hang t-VERB (l-mark w-2-to
      t-PART) (l-compound:prt w-4-out t-ADP) (l-nmod w-6-chicago t-PROPN
      (l-case w-5-in t-ADP))) (l-punct w-7-? t-PUNCT))
  # change the dependency relation of "hang" to "acl".
  tregex: "/^l-advcl$/=advcl $ (/w-.*-(?:what|when|who|where|which|how|whom
  |whose|why|qui\’{e}n|qu\’{e}|d\’{o}nde|cu\’{a}ndo|cu\’{a}nto|cu\’{a}nta
  |cu\’{a}ntos|cu\’{a}ntas|cu\’{a}l|qui\’{e}nes|cu\’{a}les|c\’{o}mo
  |ad\’{o}nde|porqu\’{e}|quien|que|donde|cuando|cuanto|cuanta|cuantos
  |cuantas|cual|quienes|cuales|como|adonde|porque|wer|welche|welcher
  |wen|wem|welchen|welchem|wessen|was|welches|warum|weshalb|weswegen
  |wieso|wie|wof\"{u}r|wofuer|wozu|womit|wodurch|worum|wor\"{u}ber
  |worueber|wobei|wovon|wo|wohin|woher|woran|worin|worauf|worunter
  |wovor|wohinter|wann|wan|wieviele|wieviel|wievele|wievievel)$/=qword $
      /t-(?:PRON|DET|ADV|ADJ).*$/) > /^l-root$/"
  transformation {
   target: "advcl"
   action: CHANGE_LABEL
   label: "l-acl"
  }
 }
 rule {
  name: "questions not parsed as nmod"
```

```
143   priority: 1
144 # what city did Obama come from?
145   # city should be nmod here
146   # TODO: Ideally this should produce same parse as "Obama comes from what
          city", whereas the current rule procduces "Obama comes [from] what
          city" where [from] is invisible.
147   # (l-root w-5-come t-VERB (l-dobj w-2-city t-NOUN (l-det w-1-what
          t-DET)) (l-aux w-3-did t-AUX) (l-nsubj w-4-obama t-PROPN) (l-nmod
          w-6-from t-ADP) (l-punct w-7-? t-PUNCT))
148   tregex: "/^l-.*$/=root < (/^l-.*$/=pobj <<
          (/w-.*-(?:what|when|who|where|which|how|whom|whose|why
149   |qui\'{e}n|qu\'{e}|d\'{o}nde|cu\'{a}ndo|cu\'{a}nto|cu\'{a}nta
150   |cu\'{a}ntos|cu\'{a}ntas|cu\'{a}l|qui\'{e}nes|cu\'{a}les|c\'{o}mo
151   |ad\'{o}nde|porqu\'{e}|quien|que|donde|cuando|cuanto|cuanta
152   |cuantos|cuantas|cual|quienes|cuales|como|adonde|porque|wer
153   |welche|welcher|wen|wem|welchen|welchem|wessen|was|welches
154   |warum|weshalb|weswegen|wieso|wie|wof\"{u}r|wofuer|wozu
155   |womit|wodurch|worum|wor\"{u}ber|worueber|wobei|wovon|wo
156   |wohin|woher|woran|worin|worauf|worunter|wovor|wohinter
157   |wann|wan|wieviele|wieviel|wievele|wievievel)$/=qword $
          /t-(?:PRON|DET|ADV|ADJ).*$/)) < (/^l-nmod$/ < /^t-ADP$/ !>
          /^l-acl.*$/)"
158   transformation {
159     target: "pobj"
160     action: CHANGE_LABEL
161     label: "l-nmod"
162   }
163 }
164 rule {
165   name: "questions not parsed nmod"
166   priority: 1
167 # where did Obama come from?
168   # where should be nmod here
169   # TODO: Ideally this should produce same parse as "Obama comes from
          where", whereas the current rule procduces "Obama comes [from]
          where". Here [from] is invisible.
170   tregex: "/^l-.*$/=root < (/^l-.*$/=pobj <
          (/w-.*-(?:what|when|who|where|which|how|whom|whose|why
171   |qui\'{e}n|qu\'{e}|d\'{o}nde|cu\'{a}ndo|cu\'{a}nto|cu\'{a}nta|cu\'{a}ntos
172   |cu\'{a}ntas|cu\'{a}l|qui\'{e}nes|cu\'{a}les|c\'{o}mo|ad\'{o}nde
173   |porqu\'{e}|quien|que|donde|cuando|cuanto|cuanta|cuantos
174   |cuantas|cual|quienes|cuales|como|adonde|porque|wer|welche
```

```
175   |welcher|wen|wem|welchen|welchem|wessen|was|welches|warum
176   |weshalb|weswegen|wieso|wie|wof\"{u}r|wofuer|wozu|womit
177   |wodurch|worum|wor\"{u}ber|worueber|wobei|wovon|wo|wohin
178   |woher|woran|worin|worauf|worunter|wovor|wohinter|wann|wan
179   |wieviele|wieviel|wievele|wievievel)$/=qword $
          /t-(?:PRON|DET|ADV|ADJ).*$/)) < (/^l-nmod$/ < /^t-ADP$/ !>
          /^l-acl.*$/)"
180   transformation {
181    target: "pobj"
182    action: CHANGE_LABEL
183    label: "l-nmod"
184   }
185  }
186 }
187 rulegroup {
188  name: "Conjunctions"
189  priority: 1
190  rule {
191   name: "np conjunction"
192   # "Bill" and "Dave"
193   priority: 1
194   tregex: "/^l-conj$/=relation $ /^t-(?:NOUN|PROPN|PRON|ADJ)$/ <
          /^t-(?:NOUN|PROPN|PRON)$/"
195   transformation {
196    target: "relation"
197    action: CHANGE_LABEL
198    label: "l-conj-np"
199   }
200  }
201  rule {
202   name: "adj conjunction"
203   # "red" and "blue"
204   priority: 1
205   tregex: "/^l-conj$/=relation $ /^t-(?:ADJ)$/ < /^t-(?:ADJ)$/"
206   transformation {
207    target: "relation"
208    action: CHANGE_LABEL
209    label: "l-conj-adj"
210   }
211  }
212  rule {
213   name: "sentence conjunction"
```

```
  # Cameron "directed" Titanic and Spielber "produced" Transformers.
  priority: 1
  tregex: "/^l-conj$/=relation $ /^t-(?:VERB)$/ < /^l-dobj$/ < /^l-nsubj$/
      < /^t-(?:VERB)$/"
  transformation {
    target: "relation"
    action: CHANGE_LABEL
    label: "l-conj-sent"
  }
}
rule {
  name: "vp conjunction"
  # "directed" Titanic and "produced" Titanic.
  priority: 5
  tregex: "/^l-conj$/=relation $ /^t-(?:VERB)$/ < /^l-dobj$/ <
      /^t-(?:VERB)$/"
  transformation {
    target: "relation"
    action: CHANGE_LABEL
    label: "l-conj-vp"
  }
}
rule {
  name: "vp conjunction"
  # John "ate" apple and "suffered".
  priority: 10
  tregex: "/^l-conj$/=relation $ /^t-(?:VERB)$/ $,, /^l-dobj$/ <
      /^t-(?:VERB)$/"
  transformation {
    target: "relation"
    action: CHANGE_LABEL
    label: "l-conj-vp"
  }
}
rule {
  name: "verb conjunction"
  # "directed" and "produced"
  priority: 15
  tregex: "/^l-conj$/=relation $ /^t-(?:VERB)$/ < /^t-(?:VERB)$/"
  transformation {
    target: "relation"
    action: CHANGE_LABEL
```

```
253     label: "l-conj-verb"
254    }
255   }
256 }
257 rulegroup {
258   name: "reduced relative and relative clause extractions"
259   priority: 1
260   rule {
261    name: "pobj extraction"
262    priority: 1
263    # The "country" which Darwin "belongs" to, is UK.
264    # "country" Darwin was "born" in
265    # country "which" Darwin was born in
266   # TODO: Ideally this should produce the same parse as "Darwin was born in
         country", however, we produce "Darwin was born[in] country" where
         [in] is invisible.
267    # (l-root w-1-country t-NOUN (l-acl:relcl w-5-bear t-VERB (l-dobj
         w-2-which t-DET) (l-nsubjpass w-3-darwin t-PROPN) (l-auxpass w-4-was
         t-AUX) (l-nmod w-6-at t-ADP)))
268    tregex: "/^l-acl.*$/=target > /^l-.*$/=origin $ /^w-.*$/=originword <
         (/^l-nmod$/ < /^t-ADP$/)"
269    transformation {
270     target: "origin"
271     action: ADD_CHILD
272     label: "l-BIND"
273     child: "v-{originword}"
274    }
275    transformation {
276     target: "target"
277     action: ADD_CHILD
278     label: "l-nmod"
279     child: "v-{originword}"
280    }
281   }
282   rule {
283    name: "nsubj extraction"
284    priority: 5
285    tregex: "/^l-acl.*$/=target > /^l-.*$/=origin $ /^w-.*$/=originword !<
         /^l-(?:nsubj|nsubjpass)$/"
286    # The company "bought" Youtube, owns Gmail. -- construction exists in
         Telugu
287    transformation {
```

```
288     target: "origin"
289     action: ADD_CHILD
290     label: "l-BIND"
291     child: "v-{originword}"
292   }
293   transformation {
294     target: "target"
295     action: ADD_CHILD
296     label: "l-nsubj"
297     child: "v-{originword}"
298   }
299 }
300 rule {
301   name: "nsubj extraction"
302   priority: 5
303   tregex: "/^l-acl.*$/=target > /^l-.*$/=origin $ /^w-.*$/=originword <
          (/^l-nsubj$/ < /^t-(?:DET|ADV)$/)"
304   # The "company" which "bought" Youtube owns Gmail.
305   transformation {
306     target: "origin"
307     action: ADD_CHILD
308     label: "l-BIND"
309     child: "v-{originword}"
310   }
311   transformation {
312     target: "target"
313     action: ADD_CHILD
314     label: "l-nsubj"
315     child: "v-{originword}"
316   }
317 }
318 rule {
319   name: "nsubjpass extraction"
320   priority: 5
321   tregex: "/^l-acl.*$/=target > /^l-.*$/=origin $ /^w-.*$/=originword <
          (/^l-nsubjpass$/ < /^t-(?:DET|ADV)$/)"
322   # The "company" which is "based" in
323   transformation {
324     target: "origin"
325     action: ADD_CHILD
326     label: "l-BIND"
327     child: "v-{originword}"
```

```
  }
  transformation {
    target: "target"
    action: ADD_CHILD
    label: "l-nsubjpass"
    child: "v-{originword}"
  }
}
rule {
  name: "dobj extraction"
  priority: 10
  tregex: "/^l-acl.*$/=target > /^l-.*$/=origin $ /^w-.*$/=originword !<
      /^l-dobj$/"
  # The "movie" Cameron "directed"
  transformation {
    target: "origin"
    action: ADD_CHILD
    label: "l-BIND"
    child: "v-{originword}"
  }
  transformation {
    target: "target"
    action: ADD_CHILD
    label: "l-dobj"
    child: "v-{originword}"
  }
}
rule {
  name: "dobj extraction"
  priority: 10
  tregex: "/^l-acl.*$/=target > /^l-.*$/=origin $ /^w-.*$/=originword <
      (/^l-dobj$/ < /^t-(?:DET|ADV)$/)"
  # The "movie" which Cameron "directed"
  transformation {
    target: "origin"
    action: ADD_CHILD
    label: "l-BIND"
    child: "v-{originword}"
  }
  transformation {
    target: "target"
    action: ADD_CHILD
```

```
    label: "l-dobj"
    child: "v-{originword}"
   }
  }
  rule {
   name: "no extraction"
   priority: 15
   tregex: "/^l-acl.*$/=relation"
  # the "issues" as he "sees" them
   transformation {
    target: "relation"
    action: CHANGE_LABEL
    label: "l-acl-other"
   }
  }
}
rulegroup {
  name: "xcomp constructions"
  priority: 1
  rule {
   name: "object of controller as subject to the controlled verb"
   # I want John to buy a laptop
   priority: 1
   tregex: "/^l-xcomp$/=target !< /^l-nsubj$/ $ (/^l-dobj$/=origin <
       /^w-.*$/=originword)"
   transformation {
    target: "origin"
    action: ADD_CHILD
    label: "l-BIND"
    child: "v-{originword}"
   }
   transformation {
    target: "target"
    action: ADD_CHILD
    label: "l-nsubj"
    child: "v-{originword}"
   }
  }
  rule {
   name: "subject of controller as subject to the controlled verb"
   # I want to buy laptop
   # Always assume there is an extraction.
```

```
409   priority: 5
410   tregex: "/^l-xcomp$/=target !< /^l-nsubj$/ $ (/^l-nsubj$/=origin <
          /^w-.*$/=originword)"
411   transformation {
412     target: "origin"
413     action: ADD_CHILD
414     label: "l-BIND"
415     child: "v-{originword}"
416   }
417   transformation {
418     target: "target"
419     action: ADD_CHILD
420     label: "l-nsubj"
421     child: "v-{originword}"
422   }
423  }
424 }
```

A.2 Obliqueness Hierarchy of Universal Dependency Labels

```
relation {
  name: "l-BIND"
  priority: 9
}
relation {
  name: "l-conj-verb"
  priority: 10
}
relation {
  name: "l-amod"
  priority: 10
}
relation {
  name: "l-compound"
  priority: 10
}
relation {
  name: "l-det"
  priority: 15
}
relation {
  name: "l-cc"
  priority: 15
}
relation {
  name: "l-conj-adj"
  priority: 16
}
relation {
  name: "l-conj-np"
  priority: 16
}
relation {
  name: "l-appos"
  priority: 16
}
relation {
  name: "l-acl:relcl"
  priority: 16
}
relation {
  name: "l-acl"
  priority: 16
}
relation {
  name: "l-acl-other"
  priority: 16
}
relation {
  name: "l-cop"
  priority: 20
}
relation {
  name: "l-dobj"
  priority: 20
}
relation {
  name: "l-dobj:wh"
  priority: 20
}
relation {
  name: "l-dobj-wh"
  priority: 20
}
relation {
  name: "l-auxpass"
  priority: 30
}
relation {
  name: "l-aux"
  priority: 30
}
relation {
  name: "l-xcomp"
  priority: 30
}
relation {
  name: "l-nmod:poss"
  priority: 40
}
relation {
  name: "l-nmod:npmod"
  priority: 40
}
relation {
  name: "l-nmod:tmod"
  priority: 40
}
relation {
  name: "l-nmod:count"
  priority: 40
}
relation {
  name: "l-nmod"
  priority: 40
}
relation {
  name: "l-advmod"
  priority: 40
}
relation {
  name: "l-advmod:count"
  priority: 40
}
relation {
  name: "l-advcl"
  priority: 40
}
relation {
  name: "l-ccomp"
  priority: 40
}
relation {
  name: "l-conj-vp"
  priority: 45
}
relation {
  name: "l-nsubj"
  priority: 60
}
relation {
  name: "l-nsubjpass"
  priority: 60
}
relation {
  name: "l-nsubj-wh"
  priority: 60
}
relation {
  name: "l-case"
  priority: 70
}
relation {
  name: "l-conj-sent"
  priority: 80
}
relation {
  name: "l-punct"
  priority: 90
}
relation {
  name: "the-rest"
  priority: 100
}
```

Figure A.1: The lower the priority value, the higher is its precedence in the hierarchy, e.g., the hierarchy for coordination constructions is conj-sent $<$ conj-vp $<$ conj-verb.

A.3 Substitution Rules for Universal Dependency Labels

```
rulegroup {
 name: "nmod"
 priority: 5
 rule {
  name: "nmod with a case marker"
  # "saw" with a "telescope"
  # "cat" in a "hat"
  priority: 5
  tregex: "/^l-(nmod|nmod:poss|nmod:count)$/=relation < (/^l-case$/ <
      (/^w-.*$/=casemarker))"
  transformation {
   target: "relation"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $f:w (lambda $g:w (lambda $e:v (exists:ex $x:v (and:c
       ($f $e) ($g $x) (p_EVENT.ENTITY_l-nmod.{casemarker}:b $e $x))))))"
  }
 }
 rule {
  name: "nmod indicating counting"
  # "number" of "people"
  # "count" of "films"
  priority: 5
  tregex: "/^l-nmod:count$/=relation"
  transformation {
   target: "relation"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
       ($f $x) ($g $y) (p_COUNT:b $y $x))))))"
  }
 }
 rule {
  name: "nmod and its variants"
  # don't confuse this relation with nn in stanford dependencies.
  # no seprate handling for tmod or npmod.
  # 65 "years" "old"
  # 6 "feet" "long"
  # "$5" a "share"
  # TODO: handle nmod seperately when the child is a verb, e.g., "eased" a
```

```
        "fraction"
   priority: 10
   tregex: "/^l-(?:nmod|nmod.*)$/=relation"
   transformation {
    target: "relation"
    action: ASSIGN_LAMBDA
    lambda: "(lambda $f:w (lambda $g:w (lambda $e:v (exists:ex $x:v (and:c
        ($f $e) ($g $x) (p_EVENT.ENTITY_l-nmod:b $e $x))))))"
   }
  }
}
rulegroup {
  name: "acl"
  priority: 10
  rule {
   name: "acl, acl:relcl"
  # country Darwin belongs to
  # movie which Cameron directed
  # the movie which won Oscar
   priority: 1
   tregex: "/^l-(?:acl|acl:relcl)$/=relation"
   transformation {
    target: "relation"
    action: ASSIGN_LAMBDA
    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
        ($f $x) ($g $y))))))"
   }
  }
  rule {
   name: "other acl-other cases"
  # Cases which do not involve extractions
  # the "issues" as he "sees" them
   priority: 1
   tregex: "/^l-acl-other$/=relation"
   transformation {
    target: "relation"
    action: ASSIGN_LAMBDA
    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
        ($f $x) ($g $y) (p_EVENT.ENTITY_{relation}:b $y $x))))))"
   }
  }
}
```

```
rulegroup {
 name: "compound, name"
 priority: 10
 rule {
  name: "verb particle"
  # hang out
  priority: 1
  tregex: "/^l-(?:compound:prt)$/=relation"
  transformation {
   target: "relation"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $f:w (lambda $g:w (lambda $x:v ($f $x))))"
  }
 }
 rule {
  name: "compound between two proper nouns"
  # "Barack" "Obama"
  priority: 1
  tregex: "/^l-(?:compound|name)$/=relation $ /t-PROPN/ < /t-PROPN/"
  transformation {
   target: "relation"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (and:c ($f $x) ($g
       $x)))))"
  }
 }
 rule {
  name: "compound from a noun to proper noun"
  # "Hilton" "hotel"
  priority: 5
  tregex: "/^l-compound$/=relation < /t-PROPN/"
  transformation {
   target: "relation"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
       ($f $x) ($g $y) (p_EVENT.ENTITY_{relation}:b $x $y))))))"
  }
 }
 rule {
  name: "compound from noun to noun"
  # "coffee" "table"
  priority: 10
```

```
114   tregex: "/^l-compound$/=relation"
115   transformation {
116     target: "relation"
117     action: ASSIGN_LAMBDA
118     lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (and:c ($f $x) ($g
            $x)))))"
119   }
120  }
121 }
122 rulegroup {
123  name: "advmod"
124  priority: 10
125  rule {
126   name: "advmod in how many"
127   # "how" "many"
128   priority: 1
129   tregex: "/^l-advmod:count$/=relation"
130   transformation {
131     target: "relation"
132     action: ASSIGN_LAMBDA
133     lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
            ($f $x) ($g $y) (p_COUNT:b $x $y))))))"
134   }
135  }
136  rule {
137   name: "advmod attached to question word"
138   # "in" a "hat"
139   priority: 5
140   tregex: "/^l-advmod$/=relation < /^t-.*:WH$/"
141   transformation {
142     target: "relation"
143     action: ASSIGN_LAMBDA
144     lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
            ($f $x) ($g $y) (p_EVENT.ENTITY_{relation}:b $x $y))))))"
145   }
146  }
147  rule {
148   name: "other advmod"
149   priority: 10
150   tregex: "/^l-advmod$/=relation"
151   transformation {
152     target: "relation"
```

```
153     action: ASSIGN_LAMBDA
154     lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (and:c ($f $x) ($g
            $x)))))"
155   }
156  }
157 }
158 rulegroup {
159  name: "case"
160  priority: 10
161  rule {
162   name: "case"
163   # "in" a "hat"
164   priority: 10
165   tregex: "/^l-case$/=relation"
166   transformation {
167    target: "relation"
168    action: ASSIGN_LAMBDA
169    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v ($f $x))))"
170   }
171  }
172 }
173 rulegroup {
174  name: "cc"
175  priority: 10
176  rule {
177   name: "cc"
178   # "Bill" "and" Dave
179   priority: 10
180   tregex: "/^l-cc$/=relation"
181   transformation {
182    target: "relation"
183    action: ASSIGN_LAMBDA
184    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
            ($f $x) ($g $y))))))"
185   }
186  }
187 }
188 rulegroup {
189  name: "conj"
190  priority: 10
191  rule {
192   name: "conj-np, conj-vp, conj-verb"
```

```
193    # "Bill" and "Dave"
194    priority: 10
195    tregex: "/^l-conj-(?:np|vp|verb|sent)$/=relation"
196    transformation {
197      target: "relation"
198      action: ASSIGN_LAMBDA
199      lambda: "(lambda $f:w (lambda $g:w (lambda $z:v (exists:ex $x:v
           (exists:ex $y:v (and:c ($f $x) ($g $y) (p_CONJ:tri $z $x $y)))))))"
200    }
201   }
202   rule {
203    name: "conj-adj"
204    # "quick" and "fast"
205    # TODO: The reply was quick and fast.
206    priority: 10
207    tregex: "/^l-conj-adj$/=relation"
208    transformation {
209      target: "relation"
210      action: ASSIGN_LAMBDA
211      lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (and:c ($f $x) ($g
           $x)))))"
212    }
213   }
214 }
215 rulegroup {
216   name: "cop"
217   priority: 10
218   rule {
219    name: "case"
220    # John "is" "old".
221    priority: 10
222    tregex: "/^l-cop$/=relation"
223    transformation {
224      target: "relation"
225      action: ASSIGN_LAMBDA
226      lambda: "(lambda $f:w (lambda $g:w (lambda $x:v ($f $x))))"
227    }
228   }
229 }
230 rulegroup {
231   name: "nsubj"
232   priority: 10
```

```
 rule {
  name: "nsubj from question word to number"
  # "what" is the "number" of cities in France?
  # This could be a parse mistake since the head word should be the number
      here?
  priority: 5
  tregex: "/^l-nsubj$/=relation < /^l-nmod:count$/"
  transformation {
   target: "relation"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (and:c ($f $x) ($g
       $x)))))"
  }
 }
 rule {
  name: "nsubj with cop"
  # what "companies" are in "CA"?
  priority: 10
  tregex: "/^l-nsubj$/=relation $ /^l-cop$/ $ (/^l-case$/ <
      /^w-.*$/=casemarker)"
  transformation {
   target: "relation"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $f:w (lambda $g:w (lambda $e:v (exists:ex $x:v (and:c
       ($f $e) ($g $x) (p_EVENT.ENTITY_arg1.{casemarker}:b $e $x))))))"
  }
 }
 rule {
  name: "nsubj"
  # John shouted.
  priority: 20
  tregex: "/^l-nsubj$/=relation"
  transformation {
   target: "relation"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $f:w (lambda $g:w (lambda $e:v (exists:ex $x:v (and:c
       ($f $e) ($g $x) (p_EVENT.ENTITY_arg1:b $e $x))))))"
  }
 }
}
rulegroup {
 name: "dobj"
```

```
270  priority: 5
271  rule {
272   name: "dobj when posed as a question"
273   # "give" me the "capital" of UK?
274   # This happens when subj is not seen with the verb.
275   priority: 5
276   tregex: "/^l-dobj:wh$/=relation"
277   transformation {
278    target: "relation"
279    action: ASSIGN_LAMBDA
280    lambda: "(lambda $f:w (lambda $g:w (lambda $e:v (exists:ex $x:v (and:c
          ($f $e) ($g $x) (p_EVENT.ENTITY_arg2:b $e $x) (p_TARGET:u $x))))))"
281   }
282  }
283  rule {
284   name: "dobj"
285   # "read" the "article".
286   priority: 10
287   tregex: "/^l-dobj$/=relation"
288   transformation {
289    target: "relation"
290    action: ASSIGN_LAMBDA
291    lambda: "(lambda $f:w (lambda $g:w (lambda $e:v (exists:ex $x:v (and:c
          ($f $e) ($g $x) (p_EVENT.ENTITY_arg2:b $e $x))))))"
292   }
293  }
294 }
295 rulegroup {
296  name: "nsubjpass"
297  priority: 10
298  rule {
299   name: "case"
300   # John shouted.
301   priority: 10
302   tregex: "/^l-nsubjpass$/=relation"
303   transformation {
304    target: "relation"
305    action: ASSIGN_LAMBDA
306    lambda: "(lambda $f:w (lambda $g:w (lambda $e:v (exists:ex $x:v (and:c
          ($f $e) ($g $x) (p_EVENT.ENTITY_arg2:b $e $x))))))"
307   }
308  }
```

```
}
rulegroup {
 name: "appos"
 priority: 10
 rule {
  name: "appositives"
  # "John", my "friend",
  priority: 10
  tregex: "/^l-appos$/=relation"
  transformation {
   target: "relation"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (and:c ($f $x) ($g
       $x)))))"
  }
 }
}
rulegroup {
 name: "det"
 priority: 10
 rule {
  name: "articles, question words"
  # Cu\'{a}ntas ciudades
  # Howmany cities (how many is a single word in spanish)
  priority: 1
  tregex: "/^l-det$/=relation $ /^t-NOUN$/ < (/^w-.*$/=word $
      /^t-.*:COUNT:WH$/)"
  transformation {
   target: "relation"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
       ($f $x) ($g $y) (p_COUNT:b $x $y) (p_TYPE_{word}:u $y))))))"
  }
 }
 rule {
  name: "articles, question words"
  # The city
  # which city
  priority: 5
  tregex: "/^l-det$/=relation"
  transformation {
   target: "relation"
```

```
348     action: ASSIGN_LAMBDA
349     lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (and:c ($f $x) ($g
            $x)))))"
350   }
351  }
352 }
353 rulegroup {
354  name: "amod"
355  priority: 10
356  rule {
357   name: "verb acting as adjectival modifier"
358   # I ate a hand "made" "sandwich".
359   priority: 1
360   tregex: "/^l-amod$/=relation < /t-VERB/"
361   transformation {
362    target: "relation"
363    action: ASSIGN_LAMBDA
364    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
           ($f $x) ($g $y) (p_EVENT.ENTITY_{relation}:b $y $x))))))"
365   }
366  }
367  rule {
368   name: "COUNT predicates"
369   # "cu\'{a}nta" "hijos"
370   # how-many people
371   priority: 10
372   tregex: "/^l-amod$/=relation $ /^t-NOUN$/ < (/^w-.*$/=word $
          /^t-.*:COUNT:WH$/)"
373   transformation {
374    target: "relation"
375    action: ASSIGN_LAMBDA
376    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
           ($f $x) ($g $y) (p_COUNT:b $x $y) (p_TYPE_{word}:u $y))))))"
377   }
378  }
379  rule {
380   name: "adjectival modifiers attached to verbs"
381   # "red" "car"
382   priority: 15
383   tregex: "/^l-amod$/=relation"
384   transformation {
385    target: "relation"
```

```
386     action: ASSIGN_LAMBDA
387     lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (and:c ($f $x) ($g
            $x)))))"
388   }
389  }
390 }
391 rulegroup {
392  name: "auxilary verbs"
393  priority: 10
394  rule {
395   name: "punct"
396   # "Yahoo" "!"
397   priority: 10
398   tregex: "/^l-aux.*$/=relation"
399   transformation {
400    target: "relation"
401    action: ASSIGN_LAMBDA
402    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v ($f $x))))"
403   }
404  }
405 }
406 rulegroup {
407  name: "xcomp"
408  priority: 10
409  rule {
410   name: "xcomp"
411   # Sue "asked" George to "respond" to her offfer
412   priority: 1
413   tregex: "/^l-xcomp$/=relation !< /t-(?:NOUN|PROPN)/"
414   transformation {
415    target: "relation"
416    action: ASSIGN_LAMBDA
417    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
            ($f $x) ($g $y) (p_EVENT.EVENT_{relation}:b $x $y))))))"
418   }
419  }
420  rule {
421   name: "xcomp when attached to a noun/propn"
422   # Elizabeth became queen.
423   # An additional arg2 predicate is created between the control verb and
        the noun.
424   priority: 5
```

```
425   tregex: "/^l-xcomp$/=relation < /t-(?:NOUN|PROPN)/"
426   transformation {
427    target: "relation"
428    action: ASSIGN_LAMBDA
429    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
           ($f $x) ($g $y) (p_EVENT.EVENT_{relation}:b $x $y)
           (p_EVENT.ENTITY_arg2:b $x $y))))))"
430   }
431  }
432 }
433 rulegroup {
434  name: "other complements"
435  priority: 10
436  rule {
437   name: "ccomp, advcl"
438   # I am "certain" that he "did" it.
439   # If you "know" who did it , you should "tell" the teacher
440   # TODO: in advcl, include the conditional predicates, e.g., if, when,
        inorder to, before, after etc.
441   priority: 10
442   tregex: "/^l-(?:ccomp|advcl|parataxis)$/=relation"
443   transformation {
444    target: "relation"
445    action: ASSIGN_LAMBDA
446    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
           ($f $x) ($g $y) (p_EVENT.EVENT_{relation}:b $x $y))))))"
447   }
448  }
449 }
450 rulegroup {
451  name: "mark"
452  priority: 10
453  rule {
454   name: "mark"
455   # Forces engaged in fighting "after" insurgents attacked
456   # He says "that" you like to "swim"
457   priority: 10
458   tregex: "/^l-mark$/=relation"
459   transformation {
460    target: "relation"
461    action: ASSIGN_LAMBDA
462    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v ($f $x))))"
```

```
463    }
464   }
465 }
466 rulegroup {
467   name: "punctuations"
468   priority: 10
469   rule {
470     name: "punct"
471     # "Yahoo" "!"
472     priority: 10
473     tregex: "/^l-punct$/=relation"
474     transformation {
475       target: "relation"
476       action: ASSIGN_LAMBDA
477       lambda: "(lambda $f:w (lambda $g:w (lambda $x:v ($f $x))))"
478     }
479   }
480 }
481 rulegroup {
482   name: "BIND"
483   priority: 10
484   rule {
485     name: "BIND"
486     priority: 10
487     tregex: "/^l-BIND$/=relation"
488     transformation {
489       target: "relation"
490       action: ASSIGN_LAMBDA
491       lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (and:c ($f $x) ($g
              $x)))))"
492     }
493   }
494 }
495 rulegroup {
496   name: "defaults"
497   priority: 100
498   rule {
499     name: "punct"
500     # "Yahoo" "!"
501     priority: 10
502     tregex: "/^l-.*$/=relation"
503     transformation {
```

```
    target: "relation"
    action: ASSIGN_LAMBDA
    lambda: "(lambda $f:w (lambda $g:w (lambda $x:v (exists:ex $y:v (and:c
        ($f $x) ($g $y) (p_EVENT.ENTITY_{relation}:b $x $y))))))"
   }
  }
}
```

A.4 Lexical Substitution Rules for Universal Dependencies

```
rulegroup {
 name: "verbs and auxilary verbs"
 priority: 10
 rule {
  name: "verb"
  # saw
  priority: 1
  tregex: "/^w-.*$/=word $ /^t-(?:VERB)$/"
  transformation {
   target: "word"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $x:v (p_EVENT_{word}:u $x))"
  }
 }
 rule {
  name: "auxilary verbs"
  # was
  priority: 1
  tregex: "/^w-.*$/=word $ /^t-(?:AUX)$/"
  transformation {
   target: "word"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $x:v (p_EMPTY:u $x))"
  }
 }
}
rulegroup {
 name: "nouns and proper nouns"
 priority: 10
 rule {
```

```
  name: "noun with incoming l-compound from commoun nouns"
  # "cofee" table
  # should not match "boy" in "boy David"
  priority: 1
  tregex: "/^w-.*$/=word $ /^t-(?:NOUN)$/ > (l-compound $ /w-.*/=head !$
      /t-PROPN/)"
  transformation {
   target: "word"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $x:v (p_TYPEMOD_{word}.{head}:u $x))"
  }
 }
 rule {
  name: "noun with incoming l-compound"
  # "boy" David
  priority: 5
  tregex: "/^w-.*$/=word $ /^t-(?:NOUN)$/ > l-compound"
  transformation {
   target: "word"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $x:v (p_TYPEMOD_{word}:u $x))"
  }
 }
 rule {
  name: "nouns with an outgoing arc"
  # "movie" in 2010
  priority: 10
  tregex: "/^w-.*$/=word $ /^t-(?:NOUN|PROPN)$/ $ /l-.*/"
  transformation {
   target: "word"
   action: ASSIGN_LAMBDA
   lambda: "(lambda $x:v (and:c (p_TYPE_{word}:u $x) (p_EVENT_{word}:u
       $x) (p_EVENT.ENTITY_arg0:b $x $x)))"
  }
 }
 rule {
  name: "rest of the nouns"
  # movie
  priority: 15
  tregex: "/^w-.*$/=word $ /^t-(?:NOUN|PROPN)$/"
  transformation {
   target: "word"
```

```
    action: ASSIGN_LAMBDA
    lambda: "(lambda $x:v (p_TYPE_{word}:u $x))"
   }
  }
}
rulegroup {
  name: "articles, question words, advmods, pronouns, adjectives"
  priority: 10
  rule {
   name: "question words with parent nouns"
   # ["which" city] as opposed to [the car "which"] or ["which" is]
   priority: 1
   tregex: "/^w-.*$/=word $ /t-.*:WH$/ > /l-(?:det|amod)/"
   transformation {
    target: "word"
    action: ASSIGN_LAMBDA
    lambda: "(lambda $x:v (and:c (p_TYPEMOD_{word}:u $x) (p_TARGET:u $x)))"
   }
  }
  rule {
   name: "question words with no parent noun but has outgoing arcs"
   # "where" in california
   priority: 5
   tregex: "/^w-.*$/=word $ /t-.*:WH$/ $ /^l-.*$/"
   transformation {
    target: "word"
    action: ASSIGN_LAMBDA
    lambda: "(lambda $x:v (and:c (p_TYPE_{word}:u $x) (p_EVENT_{word}:u
        $x) (p_EVENT.ENTITY_arg0:b $x $x) (p_TARGET:u $x)))"
   }
  }
  rule {
   name: "question words with no parent noun"
   # "what" is the capital of US?
   priority: 10
   tregex: "/^w-.*$/=word $ /t-.*:WH$/"
   transformation {
    target: "word"
    action: ASSIGN_LAMBDA
    lambda: "(lambda $x:v (and:c (p_TYPE_{word}:u $x) (p_TARGET:u $x)))"
   }
  }
```

```
112 rule {
113   name: "adverbs that have arguments"
114   # He is "in" today"
115   priority: 15
116   tregex: "/^w-.*$/=word $ /^l-(?:nsubj|dobj|nmod.*|ccomp|xcomp|advcl)$/ $
          /^t-ADV$/"
117   transformation {
118     target: "word"
119     action: ASSIGN_LAMBDA
120     lambda: "(lambda $x:v (p_EVENT_{word}:u $x))"
121   }
122 }
123 rule {
124   name: "adjectives that have arguments"
125   # John is "happy"
126   # 65 years "old"
127   priority: 15
128   tregex: "/^w-.*$/=word $ /^l-(?:nsubj|dobj|nmod.*|ccomp|xcomp|advcl)$/ $
          /^t-ADJ$/"
129   transformation {
130     target: "word"
131     action: ASSIGN_LAMBDA
132     lambda: "(lambda $x:v (p_EVENT_{word}:u $x))"
133   }
134 }
135 rule {
136   name: "advmod"
137   # "genetically" modified
138   # "most" recent game
139   priority: 20
140   tregex: "/^w-.*$/=word $ /^t-ADV$/"
141   transformation {
142     target: "word"
143     action: ASSIGN_LAMBDA
144     lambda: "(lambda $x:v (p_EVENTMOD_{word}:u $x)))"
145   }
146 }
147 rule {
148   name: "adjectives"
149   priority: 20
150   tregex: "/^w-.*/=word $ /^t-ADJ$/"
151   transformation {
```

```
    target: "word"
    action: ASSIGN_LAMBDA
    lambda: "(lambda $x:v (p_TYPEMOD_{word}:u $x))"
   }
  }
  rule {
   name: "pronouns with outgoing arcs"
   priority: 20
   tregex: "/w-.*/=word $ /^t-PRON$/ $ /^l-.*$/"
   transformation {
    target: "word"
    action: ASSIGN_LAMBDA
    lambda: "(lambda $x:v (and:c (p_TYPE_{word}:u $x) (p_EVENT_{word}:u
        $x) (p_EVENT.ENTITY_arg0:b $x $x)))"
   }
  }
  rule {
   name: "pronouns with no outgoing arcs"
   priority: 25
   tregex: "/w-.*/=word $ /^t-PRON$/"
   transformation {
    target: "word"
    action: ASSIGN_LAMBDA
    lambda: "(lambda $x:v (p_TYPE_{word}:u $x))"
   }
  }
  rule {
   name: "articles"
   # "the" car
   priority: 25
   tregex: "/^w-.*$/=word $ /^t-DET$/"
   transformation {
    target: "word"
    action: ASSIGN_LAMBDA
    lambda: "(lambda $x:v (p_EMPTY:u $x))"
   }
  }
}
rulegroup {
  name: "prepositions"
  priority: 10
  rule {
```

```
193   name: "prepositions that have arguments.. probably due to parse errors"
194   # What has MarioLopez been in?
195   # Here dep(in) = 0:root, dep(What) = in:dobj, dep(MarioLopez) = in:nsubj
196   priority: 1
197   tregex: "/^w-.*$/=word $ /^l-(?:nsubj|dobj|nmod.*|ccomp|xcomp|advcl)$/ $
          /^t-ADP$/"
198   transformation {
199     target: "word"
200     action: ASSIGN_LAMBDA
201     lambda: "(lambda $x:v (p_EVENT_{word}:u $x))"
202   }
203  }
204  rule {
205   name: "prepositions"
206   priority: 10
207   tregex: "/^w-.*/=word $ /^t-ADP$/"
208   transformation {
209     target: "word"
210     action: ASSIGN_LAMBDA
211     lambda: "(lambda $x:v (p_EMPTY:u $x))"
212   }
213  }
214 }
215 rulegroup {
216  name: "conjunction words"
217  # TODO: handle if, when, since, before etc.
218  priority: 10
219  rule {
220   name: "conjunctions"
221   priority: 10
222   tregex: "/^w-.*/=word $ /^t-(?:CONJ|SCONJ)$/"
223   transformation {
224     target: "word"
225     action: ASSIGN_LAMBDA
226     lambda: "(lambda $x:v (p_EMPTY:u $x))"
227   }
228  }
229 }
230 rulegroup {
231  name: "punctuations"
232  priority: 20
233  rule {
```

```
234    name: "punctuations"
235    priority: 10
236    tregex: "/^w-.*/=word > l-punct"
237    transformation {
238      target: "word"
239      action: ASSIGN_LAMBDA
240      lambda: "(lambda $x:v (p_EMPTY:u $x))"
241    }
242  }
243 }
244 rulegroup {
245  name: "default words"
246  priority: 30
247  rule {
248    name: "words with children"
249    priority: 10
250    tregex: "/^w-.*/=word $ /l-.*/"
251    transformation {
252      target: "word"
253      action: ASSIGN_LAMBDA
254      lambda: "(lambda $x:v (and:c (p_TYPE_{word}:u $x) (p_EVENT_{word}:u
           $x) (p_EVENT.ENTITY_arg0:b $x $x)))"
255    }
256  }
257  rule {
258    name: "words"
259    priority: 20
260    tregex: "/^w-.*/=word"
261    transformation {
262      target: "word"
263      action: ASSIGN_LAMBDA
264      lambda: "(lambda $x:v (p_TYPEMOD_{word}:u $x))"
265    }
266  }
267 }
268 rulegroup {
269  name: "virtual"
270  priority: 1
271  rule {
272    name: "virtual word"
273    priority: 1
274    tregex: "/v-w-.*/=target"
```

```
275   transformation {
276     target: "target"
277     action: ASSIGN_LAMBDA
278     lambda: "(lambda $x:v (p_EQUAL:b $x {target}:v))"
279   }
280  }
281  rule {
282   name: "virtual default"
283   priority: 10
284   tregex: "/v-.*/=target"
285   transformation {
286     target: "target"
287     action: ASSIGN_LAMBDA
288     lambda: "{target}:w"
289   }
290  }
291 }
```

Bibliography

Abzianidze, Lasha. A Tableau Prover for Natural Logic and Language. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 2492–2502, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

Abzianidze, Lasha, Bjerva, Johannes, Evang, Kilian, Haagsma, Hessel, van Noord, Rik, Ludmann, Pierre, Nguyen, Duc-Duy, and Bos, Johan. The Parallel Meaning Bank: Towards a Multilingual Corpus of Translations Annotated with Compositional Meaning Representations. *arXiv preprint arXiv:1702.03964*, 2017.

Akbik, Alan, chiticariu, laura, Danilevsky, Marina, Li, Yunyao, Vaithyanathan, Shivakumar, and Zhu, Huaiyu. Generating High Quality Proposition Banks for Multilingual Semantic Role Labeling. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pp. 397–407, Beijing, China, July 2015. Association for Computational Linguistics.

Al-Rfou, Rami, Perozzi, Bryan, and Skiena, Steven. Polyglot: Distributed Word Representations for Multilingual NLP. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pp. 183–192, Sofia, Bulgaria, 2013.

Ambati, Bharat Ram, Deoskar, Tejaswini, and Steedman, Mark. Shift-Reduce CCG Parsing using Neural Network Models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 447–453, San Diego, California, June 2016. Association for Computational Linguistics.

Andor, Daniel, Alberti, Chris, Weiss, David, Severyn, Aliaksei, Presta, Alessandro, Ganchev, Kuzman, Petrov, Slav, and Collins, Michael. Globally Normalized Transition-Based Neural Networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2442–2452, Berlin, Germany, August 2016. Association for Computational Linguistics.

Andreas, Jacob and Klein, Dan. Alignment-Based Compositional Semantics for Instruction Following. In *Proceedings of Empirical Methods on Natural Language Processing*, pp. 1165–1174, 2015.

Andreas, Jacob, Vlachos, Andreas, and Clark, Stephen. Semantic Parsing as Machine Translation. In *Proceedings of Association for Computational Linguistics*, pp. 47–52, 2013.

Andreas, Jacob, Rohrbach, Marcus, Darrell, Trevor, and Klein, Dan. Learning to Compose Neural Networks for Question Answering. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1545–1554, San Diego, California, June 2016. Association for Computational Linguistics.

Artzi, Yoav. Cornell SPF: Cornell Semantic Parsing Framework. *arXiv:1311.3011 [cs.CL]*, 2013.

Artzi, Yoav and Zettlemoyer, Luke. Bootstrapping Semantic Parsers from Conversations. In *Proceedings of Empirical Methods on Natural Language Processing*, pp. 421–432, Edinburgh, Scotland, July 2011.

Artzi, Yoav and Zettlemoyer, Luke. Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions. *Transations of the Association for Computational Linguistics*, 1(1):49–62, 2013.

Artzi, Yoav, Das, Dipanjan, and Petrov, Slav. Learning Compact Lexicons for CCG Semantic Parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1273–1283, Doha, Qatar, October 2014. Association for Computational Linguistics.

Artzi, Yoav, Lee, Kenton, and Zettlemoyer, Luke. Broad-coverage CCG Semantic Parsing with AMR. In *Proceedings of Empirical Methods on Natural Language Processing*, pp. 1699–1710, 2015.

Baldridge, Jason and Kruijff, Geert-Jan. Coupling CCG and Hybrid Logic Dependency Semantics. In *Proceedings of Association for Computational Linguistics*, pp. 319–326, 2002.

Banarescu, Laura, Bonial, Claire, Cai, Shu, Georgescu, Madalina, Griffitt, Kira, Hermjakob, Ulf, Knight, Kevin, Koehn, Philipp, Palmer, Martha, and Schneider, Nathan. Abstract Meaning Representation for Sembanking. In *Linguistic Annotation Workshop and Interoperability with Discourse*, pp. 178–186, Sofia, Bulgaria, August 2013.

Banko, Michele, Cafarella, Michael J, Soderland, Stephen, Broadhead, Matthew, and Etzioni, Oren. Open Information Extraction from the Web. In *IJCAI*, volume 7, pp. 2670–2676, 2007.

Bao, Junwei, Duan, Nan, Zhou, Ming, and Zhao, Tiejun. Knowledge-Based Question Answering as Machine Translation. In *Proceedings of Association for Computational Linguistics*, pp. 967–976, 2014.

Basile, Valerio and Bos, Johan. Aligning Formal Meaning Representations with Surface Strings for Wide-Coverage Text Generation. In *Proceedings of the 14th European Workshop on Natural Language Generation*, pp. 1–9, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

Basile, Valerio, Bos, Johan, Evang, Kilian, and Venhuizen, Noortje. Developing a large semantically annotated corpus. In *LREC 2012, Eighth International Conference on Language Resources and Evaluation*, Istanbul, Turkey, May 2012.

Bast, Hannah and Haussmann, Elmar. More Accurate Question Answering on Freebase. In *Proceedings of ACM International Conference on Information and Knowledge Management*, pp. 1431–1440, 2015.

Bédaride, Paul and Gardent, Claire. Deep Semantics for Dependency Structures. In *Proceedings of Conference on Intelligent Text Processing and Computational Linguistics*, pp. 277–288, 2011.

Beltagy, Islam. *Natural Language Semantics Using Probabilistic Logic*. PhD thesis, Department of Computer Science, The University of Texas at Austin, December 2016.

Beltagy, Islam, Roller, Stephen, Cheng, Pengxiang, Erk, Katrin, and Mooney, Raymond J. Representing meaning with a combination of logical and distributional models. *Computational Linguistics*, 2016.

Bender, Emily M, Flickinger, Dan, and Oepen, Stephan. The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In *Proceedings of the 2002 workshop on Grammar engineering and evaluation-Volume 15*, pp. 1–7. Association for Computational Linguistics, 2002.

Bender, Emily M., Flickinger, Dan, Oepen, Stephan, Packard, Woodley, and Copestake, Ann. Layers of Interpretation: On Grammar and Compositionality. In *Proceedings of the 11th International Conference on Computational Semantics*, pp. 239–249, London, UK, April 2015. Association for Computational Linguistics.

Berant, Jonathan and Liang, Percy. Semantic Parsing via Paraphrasing. In *Proceedings of Association for Computational Linguistics*, pp. 1415–1425, 2014.

Berant, Jonathan and Liang, Percy. Imitation Learning of Agenda-Based Semantic Parsers. *Transactions of the Association for Computational Linguistics*, 3:545–558, 2015.

Berant, Jonathan, Chou, Andrew, Frostig, Roy, and Liang, Percy. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of Empirical Methods on Natural Language Processing*, pp. 1533–1544, 2013.

Bisk, Yonatan, Reddy, Siva, Blitzer, John, Hockenmaier, Julia, and Steedman, Mark. Evaluating Induced CCG Parsers on Grounded Semantic Parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2022–2027, Austin, Texas, November 2016a. Association for Computational Linguistics.

Bisk, Yonatan, Yuret, Deniz, and Marcu, Daniel. Natural Language Communication with Robots. In *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 751–761, San Diego, CA, June 2016b.

Bollacker, Kurt, Evans, Colin, Paritosh, Praveen, Sturge, Tim, and Taylor, Jamie. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250. ACM, 2008.

Bordes, Antoine, Chopra, Sumit, and Weston, Jason. Question Answering with Subgraph Embeddings. In *Proceedings of Empirical Methods on Natural Language Processing*, pp. 615–620, 2014.

Bordes, Antoine, Usunier, Nicolas, Chopra, Sumit, and Weston, Jason. Large-scale simple question answering with memory networks. CoRR abs/1506.02075, 2015.

Bos, Johan. Let's not argue about semantics. In *Proceedings of LREC*, pp. 2835–2840, 2008a.

Bos, Johan. Wide-Coverage Semantic Analysis with Boxer. In Bos, Johan and Delmonte, Rodolfo (eds.), *Semantics in Text Processing. STEP 2008 Conference Proceedings*, volume 1 of *Research in Computational Semantics*, pp. 277–286. College Publications, 2008b.

Bos, Johan. Expressive Power of Abstract Meaning Representations. *Computational Linguistics*, 42(3):527–535, September 2016. ISSN 0891-2017.

Bos, Johan, Clark, Stephen, Steedman, Mark, Curran, James R., and Hockenmaier, Julia. Wide-Coverage Semantic Representations from a CCG Parser. In *Proceedings of International Conference on Computational Linguistics*, pp. 1240–1246, 2004.

Branavan, S.R.K., Chen, Harr, Zettlemoyer, Luke, and Barzilay, Regina. Reinforcement Learning for Mapping Instructions to Actions. In *Proceedings of Association for Computational Linguistics*, pp. 82–90, 2009.

Butt, Miriam, Dyvik, Helge, King, Tracy Holloway, Masuichi, Hiroshi, and Rohrer, Christian. The parallel grammar project. In *Proceedings of the 2002 workshop on Grammar engineering and evaluation-Volume 15*, pp. 1–7. Association for Computational Linguistics, 2002.

Cai, Qingqing and Yates, Alexander. Large-scale Semantic Parsing via Schema Matching and Lexicon Extension. In *Proceedings of Association for Computational Linguistics*, pp. 423–433, 2013.

Carlson, Andrew, Betteridge, Justin, Kisiel, Bryan, Settles, Burr, Hruschka, Jr., Estevam R., and Mitchell, Tom M. Toward an Architecture for Never-ending Language Learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pp. 1306–1313, Atlanta, Georgia, 2010. AAAI Press.

Carpenter, Bob. *Type-Logical Semantics*. MIT Press, Cambridge, MA, USA, 1998. ISBN 0-262-03248-1.

Champollion, Lucas. Quantification and negation in event semantics. *Baltic International Yearbook of Cognition, Logic and Communication*, 6(1):3, 2010.

Chen, Danqi and Manning, Christopher. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 740–750, Doha, Qatar, October 2014. Association for Computational Linguistics.

Chen, David L and Mooney, Raymond J. Learning to sportscast: a test of grounded language acquisition. In *Proceedings of the 25th international conference on Machine learning*, pp. 128–135. ACM, 2008.

Chen, David L. and Mooney, Raymond J. Learning to Interpret Natural Language Navigation Instructions from Observations. In *Proceedings of Association for the Advancement of Artificial Intelligence*, pp. 1–2, 2011.

Choi, Eunsol, Kwiatkowski, Tom, and Zettlemoyer, Luke. Scalable Semantic Parsing with Partial Ontologies. In *Proceedings of Association for Computational Linguistics*, pp. 1311–1320, 2015.

Cimiano, Philipp. Flexible Semantic Composition with DUDES. In *Proceedings of International Conference on Computational Semantics*, pp. 272–276, 2009.

Cimiano, Philipp, Lopez, Vanessa, Unger, Christina, Cabrio, Elena, Ngomo, Axel-Cyrille Ngonga, and Walter, Sebastian. Multilingual question answering over linked data (QALD-3): Lab overview. In *Information Access Evaluation. Multilinguality, Multimodality, and Visualization*, volume 8138, Valencia, Spain, 2013. Springer.

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

Clark, Stephen, Hockenmaier, Julia, and Steedman, Mark. Building Deep Dependency Structures using a Wide-Coverage CCG Parser. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pp. 327–334, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073138.

Clarke, James, Goldwasser, Dan, Chang, Ming-Wei, and Roth, Dan. Driving Semantic Parsing from the World's Response. In *Proceedings of CoNLL*, pp. 18–27, 2010.

Collins, Michael. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of Empirical Methods on Natural Language Processing*, pp. 1–8, 2002.

Collins, Michael. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637, 2003.

Collobert, Ronan, Weston, Jason, Bottou, Leon, Karlen, Michael, Kavukcuoglu, Koray, and Kuks, Pavel. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

Copestake, Ann and Jones, Karen Sparck. Natural language interfaces to databases. *The Knowledge Engineering Review*, 5(04):225–249, 1990.

Copestake, Ann, Lascarides, Alex, and Flickinger, Dan. An Algebra for Semantic Construction in Constraint-based Grammars. In *Proceedings of Association for Computational Linguistics*, pp. 140–147, 2001.

Copestake, Ann, Flickinger, Dan, Pollard, Carl, and Sag, Ivan A. Minimal Recursion Semantics: An Introduction. *Research on Language and Computation*, 3(2-3):281–332, 2005.

Crouch, Dick and King, Tracy Holloway. Semantics via f-structure rewriting. In *Proceedings of the LFG'06 Conference*, pp. 145. CSLI Publications, 2006.

Dalrymple, Mary, Lamping, John, Pereira, Fernando C. N., and Saraswat, Vijay A. Linear Logic for Meaning Assembly. In *Proceedings of Computational Logic for Natural Language Processing*, 1995.

Damonte, Marco, Cohen, Shay B, and Satta, Giorgio. An incremental parser for abstract meaning representation. *arXiv preprint arXiv:1608.06111*, 2016.

Das, Dipanjan, Chen, Desai, Martins, André FT, Schneider, Nathan, and Smith, Noah A. Frame-semantic parsing. *Computational linguistics*, 40(1):9–56, 2014.

de Marneffe, Marie-Catherine, Maccartney, Bill, and Manning, Christopher D. Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of LREC*, pp. 449–454, 2006.

Debusmann, Ralph, Duchier, Denys, Koller, Alexander, Kuhlmann, Marco, Smolka, Gert, and Thater, Stefan. A Relational Syntax-Semantics Interface Based on Dependency Grammar. In *Proceedings of International Conference on Computational Linguistics*, pp. 176–182, 2004.

Dong, Li and Lapata, Mirella. Language to Logical Form with Neural Attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 33–43, Berlin, Germany, August 2016. Association for Computational Linguistics.

Dong, Li, Wei, Furu, Zhou, Ming, and Xu, Ke. Question Answering over Freebase with Multi-Column Convolutional Neural Networks. In *Proceedings of Association for Computational Linguistics*, pp. 260–269, 2015.

Dyer, Chris, Ballesteros, Miguel, Ling, Wang, Matthews, Austin, and Smith, Noah A. Transition-Based Dependency Parsing with Stack Long Short-Term Memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 334–343, Beijing, China, July 2015. Association for Computational Linguistics.

Evang, Kilian and Bos, Johan. Cross-lingual Learning of an Open-domain Semantic Parser. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics*, pp. 579–588, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee.

Fader, Anthony, Zettlemoyer, Luke, and Etzioni, Oren. Paraphrase-Driven Learning for Open Question Answering. In *Proceedings of Association for Computational Linguistics*, pp. 1608–1618, 2013.

Falke, Tobias, Stanovsky, Gabriel, Gurevych, Iryna, and Dagan, Ido. Porting an Open Information Extraction System from English to German. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 892–898, Austin, Texas, November 2016. Association for Computational Linguistics.

Fancellu, Federico, Reddy, Siva, Lopez, Adam, and Webber, Bonnie. Universal Dependencies to Logical Forms with Negation Scope. *arXiv Preprint*, 2017.

Flanigan, Jeffrey, Thomson, Sam, Carbonell, Jaime, Dyer, Chris, and Smith, Noah A. A Discriminative Graph-Based Parser for the Abstract Meaning Representation. In *Proceedings of Association for Computational Linguistics*, pp. 1426–1436, 2014.

Gardent, Claire and Kallmeyer, Laura. Semantic Construction in Feature-based TAG. In *Proceedings of European Chapter of the Association for Computational Linguistics*, pp. 123–130, 2003. ISBN 1-333-56789-0.

Gardner, Matt and Krishnamurthy, Jayant. Open-Vocabulary Semantic Parsing with both Distributional Statistics and Formal Knowledge. In *Proceedings of Association for the Advancement of Artificial Intelligence*, 2017.

Ge, Ruifang and Mooney, Raymond. Learning a Compositional Semantic Parser using an Existing Syntactic Parser. In *Proceedings of Association for Computational Linguistics*, pp. 611–619, 2009.

Ge, Ruifang and Mooney, Raymond J. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the ninth conference on computational natural language learning*, pp. 9–16. Association for Computational Linguistics, 2005.

Goldwasser, Dan and Roth, Dan. Learning from Natural Instructions. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pp. 1794–1800, Barcelona, Spain, 2011.

Goldwasser, Dan, Reichart, Roi, Clarke, James, and Roth, Dan. Confidence Driven Unsupervised Semantic Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 1486–1495, Portland, Oregon, USA, June 2011.

Green Jr, Bert F, Wolf, Alice K, Chomsky, Carol, and Laughery, Kenneth. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, pp. 219–224. ACM, 1961.

Grefenstette, Edward and Sadrzadeh, Mehrnoosh. Experimental Support for a Categorical Compositional Distributional Model of Meaning. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pp. 1394–1404, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.

Haas, Carolin and Riezler, Stefan. A Corpus and Semantic Parser for Multilingual Natural Language Querying of OpenStreetMap. In *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 740–750, San Diego, California, June 2016. Association for Computational Linguistics.

Hahn, Michael and Meurers, Detmar. On deriving semantic representations from dependencies: A practical approach for evaluating meaning in learner corpora. In *Proceedings of the Int. Conference on Dependency Linguistics (Depling 2011)*, pp. 94–103, Barcelona, 2011.

Harris, Steve, Seaborne, Andy, and Prud'hommeaux, Eric. SPARQL 1.1 query language. *W3C recommendation*, 21(10), 2013.

Heilman, Michael and Smith, Noah A. Tree Edit Models for Recognizing Textual Entailments, Paraphrases, and Answers to Questions. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pp. 1011–1019, 2010. ISBN 1-932432-65-5.

Hermann, Karl Moritz and Blunsom, Phil. The Role of Syntax in Vector Space Models of Compositional Semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 894–904, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

Hockenmaier, Julia and Steedman, Mark. CCGbank: A corpus of CCG derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396, September 2007. ISSN 0891-2017. doi: 10.1162/coli.2007.33.3.355.

Holzschuher, Florian and Peinl, René. Performance of Graph Query Languages: Comparison of Cypher, Gremlin and Native Access in Neo4J. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pp. 195–204, Genoa, Italy, 2013. ACM. ISBN 978-1-4503-1599-9.

Huang, Kejun, Gardner, Matt, Papalexakis, Evangelos, Faloutsos, Christos, Sidiropoulos, Nikos, Mitchell, Tom, Talukdar, Partha P., and Fu, Xiao. Translation Invariant Word Embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1084–1088, Lisbon, Portugal, 2015.

Jakob, Max, Lopatková, Markéta, and Kordoni, Valia. Mapping between Dependency Structures and Compositional Semantic Representations. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, 2010.

Jia, Robin and Liang, Percy. Data Recombination for Neural Semantic Parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 12–22, Berlin, Germany, August 2016. Association for Computational Linguistics.

Jie, Zhanming and Lu, Wei. Multilingual Semantic Parsing : Parsing Multiple Languages into Semantic Representations. In *Proceedings of International Conference*

on Computational Linguistics, pp. 1291–1301, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.

Jones, Bevan Keeley, Johnson, Mark, and Goldwater, Sharon. Semantic Parsing with Bayesian Tree Transducers. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pp. 488–496, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

Joshi, Aravind K and Schabes, Yves. Tree-adjoining grammars. In *Handbook of Formal Languages*, pp. 69–123. Springer, 1997.

Joshi, Aravind K., Kallmeyer, Laura, and Romero, Maribel. Flexible Composition In LTAG: Quantifier Scope and Inverse Linking. In Bunt, Harry and Muskens, Reinhard (eds.), *Computing Meaning*, volume 83 of *Studies in Linguistics and Philosophy*, pp. 233–256. Springer Netherlands, 2007. ISBN 978-1-4020-5957-5.

Kallmeyer, Laura and Joshi, Aravind. Factoring predicate argument and scope semantics: Underspecified semantics with LTAG. *Research on Language and Computation*, 1(1-2):3–58, 2003.

Kaplan, Ronald M and Bresnan, Joan. Lexical-functional grammar: A formal system for grammatical representation. *Formal Issues in Lexical-Functional Grammar*, pp. 29–130, 1982.

Kate, Rohit J., Wong, Yuk Wah, and Mooney, Raymond J. Learning to Transform Natural to Formal Languages. In *AAAI*, pp. 1062–1068. AAAI Press / The MIT Press, 2005. ISBN 1-57735-236-X.

Kiperwasser, Eliyahu and Goldberg, Yoav. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016. ISSN 2307-387X.

Kočiský, Tomáš, Melis, Gábor, Grefenstette, Edward, Dyer, Chris, Ling, Wang, Blunsom, Phil, and Hermann, Karl Moritz. Semantic Parsing with Semi-Supervised Sequential Autoencoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1078–1087, Austin, Texas, November 2016. Association for Computational Linguistics.

Krishnamurthy, Jayant. Probabilistic Models for Learning a Semantic Parser Lexicon. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 606–616, San Diego, California, June 2016. Association for Computational Linguistics.

Krishnamurthy, Jayant and Kollar, Thomas. Jointly Learning to Parse and Perceive: Connecting Natural Language to the Physical World. *Transations of the Association for Computational Linguistics*, 1(1):193–206, 2013.

Krishnamurthy, Jayant and Mitchell, Tom. Weakly Supervised Training of Semantic Parsers. In *Proceedings of Empirical Methods on Natural Language Processing*, pp. 754–765, 2012.

Krishnamurthy, Jayant and Mitchell, Tom M. Learning a Compositional Semantics for Freebase with an Open Predicate Vocabulary. *Transactions of the Association for Computational Linguistics*, 3:257–270, 2015.

Kwiatkowksi, Tom, Zettlemoyer, Luke, Goldwater, Sharon, and Steedman, Mark. Inducing Probabilistic CCG Grammars from Logical Form with Higher-Order Unification. In *Proceedings of Empirical Methods on Natural Language Processing*, pp. 1223–1233, 2010.

Kwiatkowski, Tom, Choi, Eunsol, Artzi, Yoav, and Zettlemoyer, Luke. Scaling Semantic Parsers with On-the-Fly Ontology Matching. In *Proceedings of Empirical Methods on Natural Language Processing*, pp. 1545–1556, 2013.

Lee, Kenton, Lewis, Mike, and Zettlemoyer, Luke. Global Neural CCG Parsing with Optimality Guarantees. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2366–2376, Austin, Texas, November 2016. Association for Computational Linguistics.

Lehmann, Jens, Isele, Robert, Jakob, Max, Jentzsch, Anja, Kontokostas, Dimitris, Mendes, Pablo N, Hellmann, Sebastian, Morsey, Mohamed, Van Kleef, Patrick, Auer, Sören, and others. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

Lenat, Douglas B, Prakash, Mayank, and Shepherd, Mary. CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI magazine*, 6(4):65, 1985.

Levy, Roger and Andrew, Galen. Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *Proceedings of LREC*, pp. 2231–2234, 2006.

Lewis, Mike and Steedman, Mark. Combined Distributional and Logical Semantics. *Transactions of the Association for Computational Linguistics*, 1:179–192, 2013.

Lewis, Mike and Steedman, Mark. A* CCG Parsing with a Supertag-factored Model. In *Proceedings of Empirical Methods on Natural Language Processing*, pp. 990–1000, 2014.

Liang, Percy. Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*, 2013.

Liang, Percy, Jordan, Michael, and Klein, Dan. Learning Dependency-Based Compositional Semantics. In *Proceedings of Association for Computational Linguistics*, pp. 590–599, 2011.

Liu, Fei, Flanigan, Jeffrey, Thomson, Sam, Sadeh, Norman, and Smith, Noah A. Toward Abstractive Summarization Using Semantic Representations. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pp. 1077–1086, 2015.

MacMahon, Matt, Stankiewicz, Brian, and Kuipers, Benjamin. Walk the Talk: Connecting Language, Knowledge, and Action in Route Instructions. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI'06, pp. 1475–1482, Boston, Massachusetts, 2006. AAAI Press. ISBN 978-1-57735-281-5.

Manning, Christopher D., Surdeanu, Mihai, Bauer, John, Finkel, Jenny, Bethard, Steven J., and McClosky, David. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60, 2014.

Marcus, Mitchell P., Marcinkiewicz, Mary Ann, and Santorini, Beatrice. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.

Martins, Andre, Almeida, Miguel, and Smith, Noah A. Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 617–622, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

Martínez-Gómez, Pascual, Mineshima, Koji, Miyao, Yusuke, and Bekki, Daisuke. ccg2lambda: A Compositional Semantics System. In *Proceedings of ACL-2016 System Demonstrations*, pp. 85–90, Berlin, Germany, August 2016. Association for Computational Linguistics.

Matuszek, Cynthia, FitzGerald, Nicholas, Zettlemoyer, Luke, Bo, Liefeng, and Fox, Dieter. A Joint Model of Language and Perception for Grounded Attribute Learning. In *Proceedings of International Conference on Machine Learning*, 2012.

Matuszek, Cynthia, Herbst, Evan, Zettlemoyer, Luke, and Fox, Dieter. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pp. 403–415. Springer, 2013.

McDonald, Ryan, Pereira, Fernando, Ribarov, Kiril, and Hajic, Jan. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pp. 523–530, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics.

McDonald, Ryan, Nivre, Joakim, Quirmbach-Brundage, Yvonne, Goldberg, Yoav, Das, Dipanjan, Ganchev, Kuzman, Hall, Keith, Petrov, Slav, Zhang, Hao, Täckström, Oscar, Bedini, Claudia, Bertomeu Castelló, Núria, and Lee, Jungmee. Universal Dependency Annotation for Multilingual Parsing. In *Proceedings of Association for Computational Linguistics*, pp. 92–97, 2013.

Montague, Richard. Universal grammar. *Theoria*, 36(3):373–398, 1970.

Montague, Richard. The Proper Treatment of Quantification in Ordinary English. In Hintikka, K.J.J., Moravcsik, J.M.E., and Suppes, P. (eds.), *Approaches to Natural Language*, volume 49 of *Synthese Library*, pp. 221–242. Springer Netherlands, 1973. ISBN 978-90-277-0233-3.

Moortgat, Michael. *Categorical Investigations. Logical and Linguistic Aspects of the Lambek Calculus*. Foris, Dordrecht, 1988.

Moortgat, Michael. Generalized Quantification and Discontinuous Type Constructors. Technical report, University of Utrecht, 1991.

Narayan, Shashi and Gardent, Claire. Hybrid Simplification using Deep Semantics and Machine Translation. In *Proceedings of Association for Computational Linguistics*, pp. 435–445, 2014.

Narayan, Shashi, Reddy, Siva, and Cohen, Shay B. Paraphrase Generation from Latent-Variable PCFGs for Semantic Parsing. In *INLG 2016 - Proceedings of the Ninth International Natural Language Generation Conference, September 5-8, 2016, Edinburgh, UK*, pp. 153–162, 2016.

Neelakantan, Arvind, Le, Quoc V., Abadi, Martín, McCallum, Andrew, and Amodei, Dario. Learning a Natural Language Interface with Neural Programmer. *CoRR*, abs/1611.08945, 2016.

Nesson, Rebecca and Shieber, Stuart M. Simpler TAG Semantics Through Synchronization. In *Proceedings of the 11th Conference on Formal Grammar*, pp. 129–142, Malaga, Spain, 2006. Center for the Study of Language and Information.

Nivre, Joakim, Hall, Johan, Nilsson, Jens, Chanev, Atanas, Eryigit, Gülsen, Kübler, Sandra, Marinov, Svetoslav, and Marsi, Erwin. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02): 95–135, 2007.

Nivre, Joakim, Marneffe, Marie-Catherine de, Ginter, Filip, Goldberg, Yoav, Hajic, Jan, Manning, Christopher D., McDonald, Ryan, Petrov, Slav, Pyysalo, Sampo, Silveira, Natalia, Tsarfaty, Reut, and Zeman, Daniel. Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation*, Paris, France, 2016. European Language Resources Association (ELRA). ISBN 978-2-9517408-9-1.

Nivre et al, Joakim. Universal dependencies 1.3, 2016. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University in Prague.

Palmer, Martha, Gildea, Daniel, and Kingsbury, Paul. The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1):71–106, 2005.

Parikh, Ankur P., Poon, Hoifung, and Toutanova, Kristina. Grounded Semantic Parsing for Complex Knowledge Extraction. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pp. 756–766, 2015.

Parsons, Terence. Some problems concerning the logic of grammatical modifiers. In *Semantics of natural language*, pp. 127–141. Springer, 1972.

Parsons, Terence. *Events in the Semantics of English*. MIT Press, Cambridge, MA, 1990.

Pasupat, Panupong and Liang, Percy. Compositional Semantic Parsing on Semi-Structured Tables. In *Proceedings of Association for Computational Linguistics*, pp. 1470–1480, 2015.

Pavlick, Ellie, Bos, Johan, Nissim, Malvina, Beller, Charley, Van Durme, Benjamin, and Callison-Burch, Chris. Adding Semantics to Data-Driven Paraphrasing. In *Proceedings of Association for Computational Linguistics*, pp. 1512–1522, 2015.

Pelletier, Francis Jeffry. The principle of semantic compositionality. *Topoi*, 13(1): 11–24, 1994.

Pennington, Jeffrey, Socher, Richard, and Manning, Christopher. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

Pereira, Fernando C. N. Categorial Semantics and Scoping. *Computational Linguistics*, 16(1):1–10, 1990. ISSN 0891-2017.

Plank, Barbara, Søgaard, Anders, and Goldberg, Yoav. Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 412–418, Berlin, Germany, 2016.

Pollard, Carl and Sag, Ivan A. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, 1994.

Poon, Hoifung. Grounded Unsupervised Semantic Parsing. In *Proceedings of Association for Computational Linguistics*, pp. 933–943, 2013.

Punyakanok, Vasin, Roth, Dan, and Yih, Wen-tau. Mapping Dependencies Trees: An Application to Question Answering. In *Proceedings of International Symposium on Artificial Intelligence and Mathematics*, pp. 1–10, 2004.

Radev, Dragomir and Mihalcea, Rada. TextGraphs: Graph-based Algorithms for Natural Language Processing. *NAACL HLT Workshop*, 2006.

Reddy, Siva, Lapata, Mirella, and Steedman, Mark. Large-scale Semantic Parsing without Question-Answer Pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392, 2014.

Reddy, Siva, Täckström, Oscar, Collins, Michael, Kwiatkowski, Tom, Das, Dipanjan, Steedman, Mark, and Lapata, Mirella. Transforming Dependency Structures to Logical Forms for Semantic Parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140, 2016. ISSN 2307-387X.

Reddy, Siva, Täckström, Oscar, Petrov, Slav, Steedman, Mark, and Lapata, Mirella. Universal Semantic Parsing. *To appear in EMNLP 2017; available as arXiv:1702.03196*, February 2017.

Riedl, Martin, Somasundaran, Swapna, Glavaš, Goran, and Hovy, Eduard. TextGraphs-11: Graph-based Methods for Natural Language Processing. *ACL Workshop*, 2017.

Rocktäschel, Tim, Singh, Sameer, and Riedel, Sebastian. Injecting Logical Background Knowledge into Embeddings for Relation Extraction. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pp. 1119–1129, 2015.

Roth, Michael and Lapata, Mirella. Neural Semantic Role Labeling with Dependency Path Embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1192–1202, Berlin, Germany, August 2016. Association for Computational Linguistics.

Sachan, Mrinmaya and Xing, Eric. Machine Comprehension using Rich Semantic Representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 486–492, Berlin, Germany, August 2016. Association for Computational Linguistics.

Schein, Barry. *Plurals and events*, volume 23. Mit Press, 1993.

Schuster, Sebastian and Manning, Christopher D. Enhanced English Universal Dependencies: An Improved Representation for Natural Language Understanding Tasks. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation*, Paris, France, 2016. European Language Resources Association (ELRA). ISBN 978-2-9517408-9-1.

Simov, Kiril and Osenova, Petya. Towards Minimal Recursion Semantics over Bulgarian Dependency Parsing. In *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*, pp. 471–478, Hissar, Bulgaria, September 2011. RANLP 2011 Organising Committee.

Spreyer, Kathrin and Frank, Anette. Projecting RMRS from TIGER Dependencies. In *Proceedings of the HPSG 2005 Conference*. CSLI Publications, 2005.

Stanovsky, Gabriel, Ficler, Jessica, Dagan, Ido, and Goldberg, Yoav. Getting More Out Of Syntax with PropS. *ArXiv e-prints*, March 2016.

Steedman, Mark. *Surface Structure and Interpretation*. Linguistic inquiry monographs. MIT Press, Cambridge (Mass.), London, 1996. ISBN 0-262-19379-5.

Steedman, Mark. *The Syntactic Process*. The MIT Press, 2000.

Steedman, Mark. *Taking Scope - The Natural Semantics of Quantifiers*. MIT Press, 2012. ISBN 978-0-262-01707-7.

Steedman, Mark and Baldridge, Jason. Combinatory Categorial Grammar. *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, pp. 181–224, 2011.

Su, Yu, Sun, Huan, Sadler, Brian, Srivatsa, Mudhakar, Gur, Izzeddin, Yan, Zenghui, and Yan, Xifeng. On Generating Characteristic-rich Question Sets for QA Evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 562–572, Austin, Texas, 2016.

Suchanek, Fabian M, Kasneci, Gjergji, and Weikum, Gerhard. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pp. 697–706. ACM, 2007.

Tang, Lappoon R and Mooney, Raymond J. Using multiple clause constructors in inductive logic programming for semantic parsing. In *European Conference on Machine Learning*, pp. 466–477. Springer, 2001.

Tesnière, Lucien. *Eléments de syntaxe structurale*. Librairie C. Klincksieck, 1959.

Thompson, Cynthia A and Mooney, Raymond J. Acquiring Word-Meaning Mappings for Natural Language Interfaces. *Journal of Artificial Intelligence Research*, 18:1–44, 2003.

Türe, Ferhan and Jojic, Oliver. Simple and Effective Question Answering with Recurrent Neural Networks. *CoRR*, abs/1606.05029, 2016.

Vanderwende, Lucy, Menezes, Arul, and Quirk, Chris. An AMR parser for English, French, German, Spanish and Japanese and a new AMR-annotated corpus. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pp. 26–30, Denver, Colorado, June 2015. Association for Computational Linguistics.

Vaswani, Ashish, Bisk, Yonatan, Sagae, Kenji, and Musa, Ryan. Supertagging With LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 232–237, San Diego, California, June 2016. Association for Computational Linguistics.

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

Vlachos, Andreas and Riedel, Sebastian. Fact Checking: Task definition and dataset construction. In *Proceedings of the ACL 2014 Workshop on Language Technologies and Computational Social Science*, pp. 18–22, Baltimore, MD, USA, June 2014. Association for Computational Linguistics.

Vo, Nguyen, Mitra, Arindam, and Baral, Chitta. The NL2kr Platform for building Natural Language Translation Systems. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 899–908, Beijing, China, July 2015. Association for Computational Linguistics.

Wang, Chuan, Xue, Nianwen, and Pradhan, Sameer. A Transition-based Algorithm for AMR Parsing. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pp. 366–375, 2015a.

Wang, Yushi, Berant, Jonathan, and Liang, Percy. Building a Semantic Parser Overnight. In *Proceedings of Association for Computational Linguistics*, pp. 1332–1342, 2015b.

Weischedel, Ralph, Hovy, Eduard, Marcus, Mitchell, Palmer, Martha, Belvin, Robert, Pradhan, Sameer, Ramshaw, Lance, and Xue, Nianwen. OntoNotes: A large training corpus for enhanced processing. *Handbook of Natural Language Processing and Machine Translation. Springer*, 2011.

West, Douglas Brent and others. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

White, Aaron Steven, Reisinger, Drew, Sakaguchi, Keisuke, Vieira, Tim, Zhang, Sheng, Rudinger, Rachel, Rawlins, Kyle, and Van Durme, Benjamin. Universal Decompositional Semantics on Universal Dependencies. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1713–1723, Austin, Texas, November 2016. Association for Computational Linguistics.

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

Wong, Yuk Wah and Mooney, Raymond. Learning Synchronous Grammars for Semantic Parsing with Lambda Calculus. In *Proceedings of Association for Computational Linguistics*, pp. 960–967, 2007.

Wong, Yuk Wah and Mooney, Raymond J. Learning for Semantic Parsing with Statistical Machine Translation. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pp. 439–446, 2006.

Woods, William A, Kaplan, Ronald M, and Nash-Webber, Bonnie. *The Lunar Sciences: Natural Language Information System: Final Report*. Bolt Beranek and Newman, 1972.

Xu, Kun, Feng, Yansong, Huang, Songfang, and Zhao, Dongyan. Question Answering via Phrasal Semantic Parsing. In *Proceedings of Conference and Labs of the Evaluation Forum*, pp. 414–426, 2015.

Xu, Kun, Reddy, Siva, Feng, Yansong, Huang, Songfang, and Zhao, Dongyan. Question Answering on Freebase via Relation Extraction and Textual Evidence. In *Proceedings of the Association for Computational Linguistics*, Berlin, Germany, August 2016. Association for Computational Linguistics.

Yang, Yi and Chang, Ming-Wei. S-MART: Novel Tree-based Structured Learning Algorithms Applied to Tweet Entity Linking. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 504–513, Beijing, China, July 2015. Association for Computational Linguistics.

Yao, Xuchen. Lean Question Answering over Freebase from Scratch. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pp. 66–70, 2015.

Yao, Xuchen and Van Durme, Benjamin. Information Extraction over Structured Data: Question Answering with Freebase. In *Proceedings of Association for Computational Linguistics*, pp. 956–966, 2014.

Yao, Xuchen, Van Durme, Benjamin, Callison-Burch, Chris, and Clark, Peter. Answer Extraction as Sequence Tagging with Tree Edit Distance. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pp. 858–867, 2013.

Yavuz, Semih, Gur, Izzeddin, Su, Yu, Srivatsa, Mudhakar, and Yan, Xifeng. Improving Semantic Parsing via Answer Type Inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 149–159, Austin, Texas, November 2016. Association for Computational Linguistics.

Yih, Wen-tau, Chang, Ming-Wei, He, Xiaodong, and Gao, Jianfeng. Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. In *Proceedings of Association for Computational Linguistics*, pp. 1321–1331, 2015.

Zelle, John M. and Mooney, Raymond J. Learning to Parse Database Queries Using Inductive Logic Programming. In *Proceedings of Association for the Advancement of Artificial Intelligence*, pp. 1050–1055, 1996.

Zettlemoyer, Luke and Collins, Michael. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *Proceedings of Uncertainty in Artificial Intelligence*, pp. 658–666, 2005.

Zettlemoyer, Luke and Collins, Michael. Online Learning of Relaxed CCG Grammars for Parsing to Logical Form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 678–687, Prague, Czech Republic, June 2007.

Zhang, Hao and McDonald, Ryan. Enforcing Structural Diversity in Cube-pruned Dependency Parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 656–661, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

Zhou, Jie and Xu, Wei. End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1127–1137, Beijing, China, July 2015. Association for Computational Linguistics.