THE UNIVERSITY
of EDINBURGH

The Lifecycle of Neural Semantic Parsing

Jianpeng Cheng

Doctor of Philosophy

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

2019

Abstract

Humans are born with the ability to learn to perceive, comprehend and communicate with language. Computing machines, on the other hand, only understand programming languages. To bridge the gap between humans and computers, deep semantic parsers convert natural language utterances into machine-understandable logical forms. The technique has a wide range of applications ranging from spoken dialogue systems and natural language interfaces. This thesis focuses on neural network-based semantic parsing.

Traditional semantic parsers function with a domain-specific grammar that pairs utterances and logical forms, and parse with a CKY-like algorithm in polynomial time. Recent advances in neural semantic parsing reformulate the task as a sequence-to-sequence learning problem. Neural semantic parsers parse a sentence in linear time, and reduce the need for domain-specific assumptions, grammar learning, and extensive feature engineering. But this modeling flexibility comes at a cost since it is no longer possible to interpret how meaning composition is performed, given that logical forms are structured objects (trees or graphs). Such knowledge plays a critical role in understanding modeling limitations so as to build better semantic parsers. Moreover, the sequence-to-sequence learning problem is fairly unconstrained, both in terms of the possible derivations to consider and in terms of the target logical forms which can be ill-formed or unexecutable. The first contribution of this thesis is an improved neural semantic parser, which produces syntactically valid logical forms following a transition system and grammar constrains. The transition system integrates the generation of domain-general (i.e., valid tree-structures and language-specific predicates) and domain-specific aspects (i.e., domain-specific predicates and entities) in a unified way. The model employs various neural attention mechanisms to handle mismatches between natural language and formal language—a central challenge in semantic parsing.

Training data to semantic parsers typically consists of utterances paired with logical forms. Another challenge of semantic parsing concerns the annotation of logical forms, which is labor-intensive. To write down the correct logical form of an utterance, one not only needs to have expertise in the semantic formalism, but also has to ensure the logical form matches the utterance semantics. We tackle this challenge in two ways. On the one hand, we extend the neural semantic parser to a weakly-supervised setting within a parser-ranker framework. The weakly-supervised setup uses training data of utterance-denotation (e.g., question-answer) pairs, which are much easier to obtain and therefore allow to scale semantic parsers to complex domains. Our framework

combines the advantages of conventional weakly-supervised semantic parsers and neural semantic parsing. Candidate logical forms are generated by a neural decoder and subsequently scored by a ranking component. We present methods to efficiently search for candidate logical forms which involve spurious ambiguity—some logical forms do not match utterance semantics but coincidentally execute to the correct denotation. They should be excluded from training.

On the other hand, we focus on how to quickly engineer a practical neural semantic parser for closed domains, by directly reducing the annotation difficulty of utterance-logical form pairs. We develop an interface for efficiently collecting compositional utterance-logical form pairs and then leverage the data collection method to train neural semantic parsers. Our method provides an end-to-end solution for closed-domain semantic parsing given only an ontology. We also extend the end-to-end solution to handle sequential utterances simulating a non-interactive user session. Specifically, the data collection interface is modified to collect utterance sequences which exhibit various co-reference patterns. Then the neural semantic parser is extended to parse context-dependent utterances.

In summary, this thesis covers the lifecycle of designing a neural semantic parser: from model design (i.e., how to model a neural semantic parser with an appropriate inductive bias), training (i.e., how to perform fully supervised and weakly supervised training for a neural semantic parser) to engineering (i.e., how to build a neural semantic parser from a domain ontology).

Acknowledgements

I would like to thank my supervisors Mirella Lapata and Adam Lopez, who shaped my way of thinking, reading, writing and presenting to be a qualified researcher.

I would like to thank my thesis examiners Chris Dyer and Ivan Titov, who denoted time in assessing every piece of my research with critical feedback, intelligent discussions and forward-looking comments.

I would also offer my gratitude to colleagues in ILCC in the University of Edinburgh, for their company in the past three years. Special thanks goes to my brilliant collaborators (in alphabetic order): Li Dong, Bowen Li, Siva Reddy, Vijay Saraswat, Xingxing Zhang; my groupmates: Rui Cai, Yang Liu, Jiangming Liu, Chaoyun Zhang, Yumo Xu; and my teammates in Alexa completion: Ben Krause, Marco Damonte, Mihai Dobre, Daniel Duma, Joachim Fainberg, Federico Fancellu, Emmanuel Kahembwe. Wish you all a success in your research and career.

Many thanks to my parents for mentally supporting me to pursue the PhD degree.

Finally, I would also like to thank Google and AdeptMind for their funding support.

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.

(*Jianpeng Cheng*)

Table of Contents

Chapter 1

Introduction

1.1 Natural Language vs Computer Language

Humans are born with the ability to learn to perceive, comprehend, and manipulate language. They understand utterances in different contexts and are able to produce and use words and sentences to communicate. Computers, on the other hand, are designed to understand only programming language. A longstanding goal in artificial intelligence is thus to develop systems that understand natural language and enable interactions between computers and humans. The task is challenging for several reasons

- **Ambiguity**: Natural languages are full of ambiguity, at various levels. An example of lexical ambiguity is the word *apple* which can mean a fruit or a company, depending on the context it occurs. Another example demonstrates syntactic ambiguity. In the sentence, "*John saw the boy with the telescope*". It is unclear whether John saw the boy by using a telescope, or the boy is carrying a telescope. The meaning is dependent on whether the preposition *with* is attached to *John* or *the boy*. As the examples reveal, one of the major problems in natural language processing is to handle ambiguity. Most ambiguities escape our notice since humans are very good at resolving them using context and knowledge of the world. But computers do not have this knowledge, and consequently are not good at making use of the context. In contrast, computer languages are designed to be nearly or completely unambiguous, which means that any statement has exactly one meaning, regardless of context.

- **Redundancy**: In order to make up for ambiguity and reduce misunderstandings, natural languages employ a lot of redundancy. Utterances can be paraphrased in

many ways, not necessarily using the most concise expressions. For example, the longer sentence "*The attorney Tom was shot by a lawyer called Cheney in Texas*" can be paraphrased with a sequence of short sentences "*Tom is an attorney. He got shot in Texas, and the shooter's name is Cheney. Cheney is a lawyer.*" We see that the short sentences exihibt a certain degree of information overlap. Another example is the user query "*Find the nearest Chinese restaurants*". The same query can be expressed more verbose as "*Please help me find the nearest Chinese restaurants to me*", while the user intention does not change. These kinds of redundancy complicate the structure of natural language. In comparison, computer languages are designed to be less redundant and more concise. It is a challenging problem to resolve lexical and structural mismatches between natural language and computer language.

- **World knowledge**: Natural languages are full of proper nouns and definite noun phrase which represent concepts and entities in the world and the domain of discourse. For example, "*Barack Obama*" refers to the former president of the United States and "*Bastille*" is a name in a restaurant database. Humans gain the knowledge from daily life, dialog context or external resources, and use the knowledge for meaning interpretation. However, computers are not naturally equipped with this knowledge. To understand natural language, computer programs need to interact with the world and the domain, acquire and represent knowledge, and make use of context for interpretation.

To tackle with the challenges in natural language understanding, the thesis adopts a technique known as semantic parsing.

1.2 Definition of Semantic Parsing

Semantic parsing has emerged as a key technology for bridging the gap between humans and computers, by converting natural language utterances into machine-understandable meaning representations. Semantic parsing techniques can be categorized into shallow and deep types. A shallow form of meaning representation is a case-role analysis (a.k.a. semantic role labeling), which identifies entities in an utterance and labels them with roles such as agent, patient, source, and destination (Pradhan et al., 2004; Lang and Lapata, 2010). Shallow semantic parsing is sometimes known as slot-filling or frame semantic parsing, since its theoretical basis comes from frame semantics (Baker

et al., 1998), wherein a word evokes a frame of related concepts and roles. Slot-filling systems are widely used in virtual assistants together with intent classifiers (Mesnil et al., 2015), which are mechanisms for identifying the frame evoked by an utterance.

Deep semantic parsing, which is the focus of this thesis, produces precise meaning representations (a.k.a logical forms) of utterances in predicate logic or other formal language which supports automated reasoning (Zelle, 1996; Zettlemoyer and Collins, 2005a; Berant et al., 2013a). A logical form can be viewed as the formal language specification of a computer program which is executable against a real-world environment such as a knowledge base (KB). The execution result is often called a denotation. Different from shallow approaches, the output of deep semantic parsing can involve a certain degree of compositionality. As an example, shallow semantic parsers can parse utterances like "*Show me the weather of Anfield Stadium tonight*" by classifying the intent as "`display weather`", and filling slots "`target place`" with "*Anfield Stadium*", and "`target time`" with "*tonight*". However, they cannot parse arbitrary compositional utterances like "*Show me weather where Team Liverpool is playing tonight*". Deep semantic parsing attempts to parse such utterances, typically by converting them to a formal meaning representation language. Table 1.1 shows examples of natural language utterances, their corresponding logical forms, and denotations. The first query "*What is the longest river in Ohio?*" is represented by the logical form `longest(and(type.river, location(Ohio)))`, which when executed against a database of US geography returns the answer *Ohio River*. In the second example, the logical form `count(daughterOf(Barack Obama))` corresponds to the query "*How many daughters does Obama have?*" and is executed against the Freebase knowledge base to return the answer *2*.

1.3 Semantic Parsing Applications

Semantic parsing has a wide range of applications centered around human-computer interaction. Published research work applies semantic parsers on question answering (Kwiatkowski et al., 2011; Liang et al., 2011), goal-oriented dialog (Wen et al., 2015), robot control (Matuszek et al., 2012), and interpreting instructions (Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013a), to name just a few.

In practice, semantic parsing has been used as a key component to power commercial personal assistants, such as the Apple Sir, Google Now, Amazon Alexa and Microsoft Cortana. Similar technique has also been applied to focused domains such as customer

Environment: A database of US geography *Utterance:* What is the longest river is in Ohio? *Logical form:* `longest(and(type.river, location(Ohio)))` *Denotation:* Ohio River
Environment: Freebase *Utterance:* How many daughters does Obama have? *Logical form:* `count(daughterOf(Barack Obama))` *Denotation:* 2

Table 1.1: Examples of questions, corresponding logical forms, and their answers.

service, help desk, technical support, navigation and home automation (McTear, 2004).

1.4 Research Questions in Semantic Parsing

Semantic parsing presents a collection of interesting research problems which influence the functionality and design of semantic parsers. This thesis relates to the following topics.

Model Design for Neural Semantic Parsers Conventional semantic parsers (Zelle, 1996; Zettlemoyer and Collins, 2005a; Wong and Mooney, 2006; Kwiatkowksi et al., 2010a; Kwiatkowski et al., 2013; Berant et al., 2013b) are defined by a domain-specific grammar and a machine-learned scoring model. The grammar defines the space of possible derivations from a sentence to a logical form, and the scoring model is used to select the best derivation from these possibilities. A chart parsing algorithm is commonly employed to parse a sentence in polynomial time.

The successful application of recurrent neural networks (Bahdanau et al., 2015; Sutskever et al., 2014) to a variety of NLP tasks has provided strong impetus to treat semantic parsing as a sequence transduction problem where an utterance is mapped to a target meaning representation in string format (Dong and Lapata, 2016; Jia and Liang, 2016; Kočiský et al., 2016). Neural semantic parsers parse a sentence in linear time, while reducing the need for domain-specific assumptions, grammar learning, and more generally extensive feature engineering. But this modeling flexibility comes at a cost since it is no longer possible to interpret how meaning composition is performed (i.e., how to obtain a logical form from an utterance with a sequence of interpretable

steps rather than string concatenation). Such knowledge plays a critical role to build generalizable semantic parsers, which can exploit compositional patterns with training data rather than memorizing it. Moreover, without any task-specific knowledge, the learning problem is fairly unconstrained, both in terms of the possible derivations to consider and in terms of the target output which can be ill-formed (e.g., with extra or missing brackets).

Handling Mismatches between Natural Language and Computer Language A central challenge in semantic parsing is coping with the different ways that logical predicates can be expressed in natural language. For example, both expressions "*which university did x attend*" and "*where did x obtain his degree*" trigger the logical predicate `education`. To tackle with the problem, conventional semantic parsers use a lexicon which is either manually defined (Angeli et al., 2012; Lee et al., 2014; Berant et al., 2013b; Reddy et al., 2014, 2016), or learned from natural language utterances paired with KB (Zettlemoyer and Collins, 2005b, 2007; Kwiatkowksi et al., 2010b; Kwiatkowski et al., 2011; Artzi and Zettlemoyer, 2013a; Krishnamurthy, 2016). A limitation of this approach lies in scalability since textual clues indirectly or not related to KB cannot be exploited.

Some work handles the mismatches between natural language and KB with an intermediate, task-independent representation. The representation can be the output of a syntactic parser (Kwiatkowski et al., 2013; Reddy et al., 2014, 2016), or some form of canonical text representation (Berant and Liang, 2014). While this approach enables the usage of abundant text data to realize cross-domain knowledge transfer, training is by no means easy due to lexical and structural mismatches.

Unifying Domain-general and Specific Information Logical forms developed for downstream applications consist of domain-general and domain-specific information (Wang et al., 2015). Domain-general aspects are language-specific predicates and compositional structures that are applicable across domains. For example, in SQL, the operators SELECT and COUNT are both language-specific and they are applicable no matter if the database stores information about restaurants or geography. On the other hand, domain-specific aspects are predicates and entities in a given domain or grounded to a downstream task. For example, they can be knowledge about an SQL schema which contains indexes of restaurants or names of mountains. One research question is therefore how to handle domain-general and domain-specific information

in semantic parsing a unified way.

Note that there has been work on designing general-purpose logical forms that capture linguistically-motivated semantic structures of utterances. Examples include the Groningen Meaning Bank (Bos et al., 2017), Abstract Meaning Representations (Banarescu et al., 2013), PropBank (Kingsbury and Palmer, 2002), to name just a few. While these logical forms offer broad coverage, solving downstream tasks still requires domain-specific knowledge or grounding. In sum, practical semantic parsers inevitably need to unify domain-general and domain-specific knowledge.

Scaling Neural Semantic Parsing to Larger Domains Early semantic parsers (Zelle, 1996; Zettlemoyer and Collins, 2005a; Wong and Mooney, 2006; Kwiatkowksi et al., 2010a) have for the most part used machine learning techniques to train on a collection of utterances paired with annotated logical forms. More recently, weak supervision has been proposed to alleviate the annotation burden, e.g., training on utterance-denotation pairs (Clarke et al., 2010; Kwiatkowski et al., 2013; Berant et al., 2013b) or user feedback (Iyer et al., 2017). These types of weak signal are much easier to obtain than logical forms and therefore allow to scale semantic parsers to complex domains.

However, weakly-supervised training also introduces challenges since logical forms are latent and need to be searched from an exponentially large space. The search procedure is only guided by clues from the weak signal. One specific challenge is that logical forms can be spurious: a logical form may coincidentally provide the correct response but not actually match the utterance semantics. Such spurious logical forms have a negative impact on the training and should be detected.

Building Neural Semantic Parsers from a Domain Ontology As mentioned above, most existing work on semantic parsing has focused on training with given utterance-logical form pairs. Assume in practice one wants to build a neural semantic parser for a new domain where no training data exists in any form, and the only available resource is a domain ontology (i.e., a collection of entities and relations in the domain). How can the neural semantic parser be engineered quickly starting almost from scratch? Wang et al. (2015) propose a functional-driven approach: they use a synchronous grammar to generate logical forms paired with formal descriptions, which are subsequently paraphrased by crowd workers to obtain natural utterances. It is worth exploring the integration of this data collection approach with neural semantic parsers, in order to develop an end-to-end system for domain service providers.

Neural Semantic Parsing for Context-dependent Utterances The bulk of existing work on semantic parsing has focused on single-turn utterances. However, users typically ask questions or perform tasks in multiple steps, and they often decompose a complex query into a sequence of inter-related sub-utterances (Iyyer et al., 2017). For instance, when searching for a restaurant, a user may first ask "*which restaurants serve thai food*" followed by "*which ones are near me*". Even in cases where users have a well-defined query in mind, it is not uncommon to ask follow-on questions, in an attempt to refine their search or because they wish to compare different results (Moe and Fader, 2001; Asri et al., 2017). A research question is therefore to build a neural semantic parser which handles sequential utterances.

1.5 Thesis Overview

This thesis is about neural semantic parsing, with a focus on the above mentioned research questions. The thesis covers model design, training and engineering: Specifically, we propose a neural semantic parser which, during generation, takes into account the syntactic structure of logical forms, unifies domain-general and specific aspects, and handles mismatches between natural language and KB. The parser (and its extensions) can be trained using various supervision signals, such as utterance-logical form pairs and utterance-denotation pairs. We also look at the more practical side of neural semantic parsing—how does one quickly build a neural semantic parser from a domain ontology? We adopt an interface which enables efficient collection of utterance-logical form pairs, and provide end-to-end solutions for the development of closed-domain neural semantic parsers. Finally, the end-to-end system is extended to handle sequential utterances, which exhibit context dependencies. The thesis is structured as follows.

Chapter 2 presents related work on semantic parsing focusing on three dimensions: semantic formalism, semantic parsing models, and training regimes.

Chapter 3 introduces a novel neural semantic parser that converts natural language utterances to tree-structured logical forms. A neural sequence-to-tree model is proposed to produce syntactically valid logical forms following a transition system and grammar constrains. The transition system integrates the generation of domain-general and specific aspects in a unified way. Moreover, the model employs various neural attention mechanism to handle mismatches between natural language and logical predicates. The neural semantic parser is evaluated on the GEOQUERY dataset, which consists of utterances paired with logical forms.

Chapter 4 extends the neural semantic parser to a weakly-supervised setting within a parser-ranker framework. The framework aims to combine the advantage of conventional weakly-supervised semantic parsers and neural semantic parsers. Candidate logical forms are decoded by a neural decoder and subsequently scored by ranking components. We present methods to efficiently search for candidate logical forms and cope with spurious logical forms. The parser-ranker framework is evaluated on three datasets WEBQUESTIONS, GRAPHQUESTIONS and SPADES.

Chapter 5 focuses on how to build a neural semantic parser for closed domains, from only a domain ontology. Our approach follows and extends the previous work of Wang et al. (2015). Specifically we build an interface for efficiently collecting compositional utterance-logical form pairs as a summarization task, and then leverage the data collection method in training neural semantic parsers. We use this method to crowd-source semantic parsing data for 6 domains covering company management, recommendation engines and health-care applications, and present parsing results.

Chapter 6 extends the previous chapter to handle sequential utterances simulating a non-interactive user session: the user can keep asking questions based on intermediate denotations. First, the data collection method is extended to collect utterance sequences which exhibit different co-reference patterns. Then the neural semantic parser is extended to parse context-dependent utterance sequences in two steps. The first step parses each utterance into an underspecified logical form which may contain placeholders of co-reference, while the second step resolves the co-reference. We evaluate both context-dependent and non context-dependent parsing strategies in the first step. We use our method to crowd-source session-based semantic parsing data on two domains and validate the effectiveness of our neural semantic parser on it.

Finally, **Chapter 7** summarizes the contributions of this thesis and discusses directions for future work.

Chapter 2

Related Work

We introduce related work in semantic parsing from three dimensions: semantic formalism, modeling and training regimes. These are all important aspects in developing semantic parsers and their related work has a certain amount of overlap.

2.1 Semantic Formalism

Logical forms have played a central role in semantic parsing systems since their developments in the 1970s (Winograd, 1972a; Woods et al., 1972). The literature is rife with semantic formalisms which can be used to define logical forms. Previous work has adopted various types of semantic formalisms which can be largely categorized into procedural and declarative semantics. Procedural semantics create computer programs which model actions about how to perform certain activities. Examples of computer programs used in semantic parsing include instructions sent to robots (Winograd, 1972b; Matuszek et al., 2013), and database queries such as domain-specific SQL (Zhong et al., 2017; Iyer et al., 2017) and functional queries (FunQL, Zelle (1995)).

Declarative semantics, on the other hand, define more linguistically motivated broad-coverage representations based on logic which support logical inference for meaning interpretation. Examples include Combinatory Categorial Grammar (CCG, Steedman (2000)), Dependency-based Compositional Semantics (DCS, Liang et al. (2011)) and Abstract Meaning Representations (AMR, Banarescu et al. (2013)).

There has been long been a debate about what is the most appropriate way to represent semantics in AI (Winograd, 1975). In fact, most of the semantic parsers in early eras, such as the LUNAR and the SHRDLU (Winograd, 1972b) involved a heavy use of domain-specialized procedural representations (Barr, 1980). Such representations

have a modeling advantage (trading off completeness) for downstream tasks, since it is natural to describe the manipulation of a simple world as programs. This applies to not only physical process such as robot instructing, but also deductive processes like playing a game or answering a question in steps. The modeling advantage has been revealed in recent advances of neural programming: recurrent neural networks have demonstrated great ability in inducing compositional programs (Reed and De Freitas, 2016; Neelakantan et al., 2016; Cai et al., 2017). For example, they learn to perform grade-school addition, bubble sort and table comprehension in procedures.

2.2 Semantic Parsing Models

Early semantic parsing systems were purely ruled-based and answered questions in constrained domains. The LUNAR system (Winograd, 1972b) is designed to handle questions about moon rocks using a large database. It converts queries into programs by mapping syntactic fragments to semantic units. The programs are subsequently executed with a retrieval component. Another example is the SHRDLU system also designed by Winograd (1972b). The system launches dialogs between the user and the system-simulated robot to manipulate simple objects on a table. Central to these systems is the idea of expressing words and sentences as computer programs, and the execution of programs corresponds to the reasoning procedures. However, the development of these systems is by no means easy since it requires a large deal of domain-specific knowledge and engineering. The systems are not able to function adequately outside the restricted domain for which they are designed.

In reaction to these problems in 1970s, the focus of semantic parsing research shifts from rule-based methods to empirical or statistical methods, where data and machine learning plays an important role. Statistical semantic parsers typically consist of three key components: a grammar, a trainable model, and a parsing algorithm. The grammar defines the space of derivations from utterances to logical forms, and the model together with the parsing algorithm find the highest scoring derivation. An example of early statistical semantic parser is the CHILL system (Zelle, 1996) based on inductive logic programming (ILP). The system uses ILP to learn control rules for a shift-reduce parser. To train and evaluate their system, Zelle (1996) create the GEOQUERY dataset which contains 880 queries to a US geography database. These queries are paired with annotated logical forms in Prolog. Subsequent work improves the CHILL system with refined ILP heuristics (Tang and Mooney, 2001).

Until early 2000, semantic parsing research was still mainly focused on narrow domains. Besides GEOQUERY, some commonly used datasets were ROBOCUP for coaching advice to soccer agents (Kitano et al., 1997), and ATIS for air travel information service (Price, 1990). A that time, statistical approaches for parsing domain-specific context-free grammars have been largely explored. For example, Kate and Mooney (2006) propose KRISP, which induces context-free grammar rules that generate logical forms, and uses kernel SVM to score derivations. Ge and Mooney (2005) propose SCISSOR, which employs an integrated statistical parser to produce a semantically augmented parse tree. Each non-terminal node in the tree has both a syntactic and a semantic label, from which the final meaning representation can be derived. The WASP system proposed by Wong and Mooney (2007) learns synchronous context free grammars that generate utterances and logical forms. Parsing is achieved by finding the most probable derivation that leads to the utterance and recovering the logical form with synchronous rules. Lu et al. (2008) proposes a generative model for utterances and logical forms. Similar to Ge and Mooney (2005), they define hybrid trees whose nodes include both words and logical form tokens. Training is performed with the EM algorithm. Their model, specifically the generative process, is later extend by Kim and Mooney (2010) to learn from ambiguous supervision.

The next breakthrough comes with the work of Zettlemoyer and Collins (2005a), who introduce CCG in semantic parsing. Their probabilistic CCG grammars can deal with long range dependencies and construct non-projective meaning representations. A great deal of work follows Zettlemoyer and Collins (2005a) but focuses on more fine-grained problems such as grammar induction and lexicon learning (Kwiatkowksi et al., 2010b; Kwiatkowski et al., 2011; Krishnamurthy and Mitchell, 2012; Artzi et al., 2014; Krishnamurthy and Mitchell, 2015; Krishnamurthy, 2016; Gardner and Krishnamurthy, 2017) or using less supervision (Artzi and Zettlemoyer, 2013a; Reddy et al., 2014). As a common paradigm, the class of work first generates candidate derivations to logical forms governed by the grammar. These candidates derivations are scored by a trainable model which can take the form of a structured perceptron (Zettlemoyer and Collins, 2007) or a log-linear model (Zettlemoyer and Collins, 2005a). Training updates model parameters such that correct derivations obtain higher scores. During inference, a CKY-style chart parsing algorithm is used to predict the most likely derivation for an utterance. Another class of work follows similar paradigm but use lambda-DCS as the semantic formalism (Berant et al., 2013a; Berant and Liang, 2014, 2015). Other interesting work includes joint semantic parsing and grounding

(Kwiatkowski et al., 2013), parsing context-dependent queries (Artzi and Zettlemoyer, 2013b; Long et al., 2016), and converting dependency trees to logical forms (Reddy et al., 2016, 2017).

With recent advances in neural networks and deep learning, there is a trend of reformulating semantic parsing as a machine translation problem, which converts a natural language sequence into a programming language consequence. The idea is not novel and has been previously studied with statistical machine translation approaches. For example, both Wong and Mooney (2006) and Andreas et al. (2013) develop word-alignment based translation models for parsing queries in the GEOQUERY. However, the task setup is important to be revisited since recurrent neural networks have been shown to be extremely useful in context modeling and sequence generation (Bahdanau et al., 2015). Following this direction, Dong and Lapata (2016) and Jia and Liang (2016) develop neural semantic parsers which treat semantic parsing as a sequence to sequence learning problem. Surprisingly, the approach has been proven effectively on even the small GEOQUERY dataset. Jia and Liang (2016) further introduces a data augmentation approach to feed neural networks which are data hungry. Their approach bootstraps a synchronous grammar from existing data and generates artificial examples as extra training data. State-of-the-art result on the GEOQUERY is obtained with this approach. Subsequent work of Kočiský et al. (2016) attempts to explore the logical form space with a generative autoencoder. They bootstraps a probabilistic monolingual grammar for logical forms, from which unseen logical forms can be sampled. These samples are used as semi-supervised training data to the autoencoder. Other related work extends the vanilla sequence to sequence model in different ways, such as employing two encoder-decoders for coarse-to-fine decoding (Dong and Lapata, 2018), handling multiple tasks with a shared encoder (Fan et al., 2017), parsing cross-domain queries (Herzig and Berant, 2017) and context-dependent queries (Suhr et al., 2018), and applying the model to other formalisms such as AMR (Konstas et al., 2017) and SQL (Zhong et al., 2017; Xu et al., 2017).

The fact that logical forms have a syntactic structure has motivated more recent work on exploring structured neural decoders to generate tree or graph structures, and grammar-constrained decoders to make sure the outputs are meaningful and executable. Our work follows this direction. In Cheng et al. (2017) our parser generates logical forms recursively with a set of transition operations whose sequence is predicted via recurrent neural networks. In Cheng et al. (2018) we compare the top-down or bottom-up generation orders. In both work grammar constrains are used to guide the tree

generation. Other related work includes Yin and Neubig (2017) who generate abstract syntax trees for source code with a grammar constrained neural decoder. Krishnamurthy et al. (2017) introduce a similar neural semantic parser which decodes rules of a grammar to obtain well-typed logical forms. Rabinovich et al. (2017) propose abstract syntax networks with a modular decoder, whose multiple submodels (one per grammar construct) are composed to generate abstract syntax trees in a top-down manner.

2.3 Training Regimes

Various types of supervision have been explored to train semantic parsers, ranging from full supervision with utterance-logical form pairs to unsupervised semantic parsing without given utterances. Early work of statistical semantic parsing has mostly used annotated training data consisting of utterances paired with logical forms (Zelle, 1996; Ge and Mooney, 2005; Kate and Mooney, 2006; Kate et al., 2005; Wong and Mooney, 2006; Lu et al., 2008; Kwiatkowksi et al., 2010a). Same applies to some of the recent work on neural semantic parsing (Dong and Lapata, 2016; Jia and Liang, 2016). This form of supervision is the most effective for training, but is also the most expensive to obtain. In order to write down a correct logical form, the annotator not only needs to have expertise in the semantic formalism, but also has to ensure the logical form matches the utterance semantics and contains no grammatical mistakes. For this reason, fully supervised training applies more to small and closed domain problems, such as querying the US geographical database (Zelle, 1996).

Over the past few years, developments have been made to train semantic parsers with weak supervision from utterance-denotation pairs (Clarke et al., 2010; Liang et al., 2011; Berant et al., 2013b; Kwiatkowski et al., 2013; Pasupat and Liang, 2015). The approach enables more efficient data collection, since denotations (such as answers to a question, responses to a system) are much easier to obtain via crowd-sourcing. For this reason, semantic parsing can be scaled to handle large, complex and open domain problems. Examples include the work that learn semantic parsers from question-answer pairs on Freebase (Liang et al., 2011; Berant et al., 2013b; Berant and Liang, 2014; Liang et al., 2017; Cheng et al., 2017), from system feedback (Clarke et al., 2010; Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013a), from abstract examples (Goldman et al., 2018), and from human feedback (Iyer et al., 2017; Lawrence and Riezler, 2018) or statements (Artzi and Zettlemoyer, 2011). A challenge of weakly supervised setup is that it renders training more difficult since logical forms need to be

searched from a latent space. Various techniques have been proposed to cope with this challenge (Berant et al., 2013b; Pasupat and Liang, 2016; Goldman et al., 2018).

Recent work on semantic parsing starts to seek more clever ways of gathering data or train the semantic parser with even weaker forms of supervision. In a class of distant supervision methods, the input is solely a knowledge base and a corpus of unlabeled sentences. Artificial training data is generated from the given resources. For example, Cai and Yates (2013) generate utterance paired with logical forms. Their approach searches for sentences containing certain entity pairs, and assume (with some pruning technique) the sentences express a certain relation from the KB. In Krishnamurthy and Mitchell (2012) and Krishnamurthy and Mitchell (2014) whose authors work with CCG formalism, an extra source of supervision is added. The semantic parser is trained to produce parses that syntactically agree with dependency structures. Reddy et al. (2014) generates utterance-denotation pairs by masking entity mentions in declarative sentences from a large corpus. A semantic parser is then trained to predict the denotations corresponding to the masked entities. Finally, Poon and Domingos (2010) and Goldwasser et al. (2011) develop purely unsupervised semantic parsers trained with only a corpus of sentences (or their dependency structures), using EM-style algorithms.

In summary, semantic parsing research includes a wide range of topics from formalism to modeling and training. From the next chapter onwards, we will introduce our neural semantic parsing framework covering these related topics.

Chapter 3

Fully-supervised Neural Semantic Parsing

In this chapter, we present a novel neural semantic parser that learns to map an utterance into a logical form, using annotated utterance-logical form pairs as the training data. Compared to traditional semantic parsers, the neural semantic parser reduces the amount of manually engineered features and domain-specific rules, as well as the complexity of the inference step. Compare to neural sequence-to-sequence models, our semantic parser generates structured objects with syntactic guarantees. Our presentation starts with the introduction of Recurrent Neural Networks (RNNs) and Long Short-Term Memory Networks (LSTMs), which have emerged to become the basic building block for sequential natural language processing (Section 3.1.1). We then introduce a variant of the LSTM, namely the stack-LSTM, which enables us to encode and generate more structured data (e.g., trees and graphs, Section 3.1.2). We also introduce encoder-decoder models (Section 3.1.3), a popular neural framework for sequence-to-sequence transduction. Next, we present our semantic parsing method. We start with a discussion on the choice of logical formulation (Section 3.2.1). We present a generic, transition-based tree generation algorithm (Section 3.2.2), and then extend the algorithm to suit the semantic parsing task (Sections 3.2.3). We then provide architectural details of the neural network that implements the generation algorithm (Section 3.2.4). We cover how to handle the loose alignments and mismatches between natural language and logical tokens through various attention mechanisms (Section 3.2.6). Finally, we present the experiments (Section 3.3) and discussion (Section 3.4).

3.1 Preliminaries

3.1.1 Recurrent Neural Networks and Long Short-Term Memory: Modeling Sequential Data

A recurrent neural network (RNN) is a class of artificial neural network where connections between neurons form a directed graph along a sequence. Such a network exhibits dynamic temporal behavior; the internal state, or the short term memory of the RNN, makes it suitable to process sequential data. When processing a variable-length sequence $x = (x_1, x_2, \cdots, x_n)$, the parameters of the RNN are repeatedly used as

$$h_t = \sigma(W_x x_t + W_h h_{t-1}) \tag{3.1}$$

where h denotes the hidden state of the RNN; W_x and W_h are weight matrices; and σ is the logistic sigmoid function.

Although RNNs can in principle model long-range dependencies, training them is difficult in practice since the repeated application of a squashing nonlinearity at each step results in an exponential decay in the error signal through time. A Long Short-Term Memory (LSTM, Hochreiter and Schmidhuber (1997)) aims to address this issue with an extra long-term memory "cell" (c_t) that is constructed as a linear combination of the previous hidden state and the input.

Formally, an LSTM processes a variable-length sequence $x = (x_1, x_2, \cdots, x_n)$ by incrementally adding new content into the memory cell, with gates controlling the extent to which new content should be memorized, old content should be erased, and current content should be exposed. At time step t, the memory c_t and the hidden state h_t are updated with the following equations:

$$\begin{bmatrix} i_t \\ f_t \\ o_t \\ \hat{c}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W \cdot [h_{t-1}, x_t] \tag{3.2}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t \tag{3.3}$$

$$h_t = o_t \odot \tanh(c_t) \tag{3.4}$$

where i, f, and o are gate activations; σ is the component-wise logistic sigmoid function; and $\odot$ is the component-wise product.

3.1.2 The Stack-LSTM: Modeling Structured Data

The standard LSTM introduced above is a powerful tool for modeling sequential data, such as the word sequence. However, in some domains data exhibits structures beyond the sequence level. An example is semantic parsing where the output logical forms are trees or graphs. In this section, we introduce a variant of the LSTM, called stack-LSTMs (Dyer et al., 2015), which allows us to model more structured data.

A stack-LSTM augments an LSTM with a "stack pointer". Similar to a conventional LSTM, the stack-LSTM always adds new inputs in the right-most position. But differently, the current location of the stack pointer determines which cell in the stack-LSTM provides c_{t-1} and h_{t-1} when computing the new memory cell contents. In addition to adding elements to the end of the sequence, the stack-LSTM adopts a pop action which moves the stack pointer to the previous element (not necessarily the right-most element). Thus, contents in the stack-LSTM are never overwritten. The push action always adds new inputs at the end, and pop only updates the stack pointer.

So far we have not yet described how the stack-LSTM enables us to model structured data. However, intuitions can be gained from the application of stack in general computer science. The push and pop actions that a stack defines allows it to model the traversal of trees and graphs. An example is the call stack which interprets and executes the abstract syntax tree of a computer program. We will describe how we adopt the stack-LSTM to generate tree-structured objects in Section 3.2.2.

3.1.3 Neural Encoder-Decoders

Many natural language generation tasks deal with the transduction from a source sequence to a target sequence. Examples include machine translation, abstractive sentence summarization, text paraphrasing, to name just as few. To tackle with such problems, previous work proposes a class of encoder-decoder models (Bahdanau et al., 2015) based on RNNs or LSTMs, which generate the target sequence conditioned on the source.

The encoder converts a variable-length source sequence $x = (x_1, x_2, \cdots, x_n)$ into a list of context-sensitive embeddings $[h_1, \cdots, h_n]$. As a common practice, an LSTM is used for the conversion and each embedding h_t is taken as the hidden state of the LSTM:

$$h_t = \text{LSTM}(h_{t-1}, x_t) \tag{3.5}$$

where the LSTM symbol in the above equation denotes the inherent transformation function that an LSTM adopts, as described in Section 3.1.1.

After the source in encoded, the decoder predicts or generates a variable-length target sequence $y = (y_1, y_2, \cdots, y_m)$ in a sequential order. It is modeled as another LSTM, which is trained to predict the next token y_t given the source x and all the previously predicted target tokens $y = (y_1, y_2, \cdots, y_{t-1})$. These target tokens are assumed to be memorized by the current state of the decoder LSTM, denoted by g_t. In other words, the decoder defines a probability over the target sequence y by decomposing the joint probability into the product of the order conditional probabilities:

$$p(y|x) = \prod_{t=1}^{m} p(y_t|y_1, \cdots, y_{t-1}, x) \tag{3.6}$$

and each conditional probability is modeled as

$$p(y_t|y_1, \cdots, y_{t-1}, x) = f(h_1, \cdots, h_n, g_t) \tag{3.7}$$

where f is a non-linear function that uses features from both the encoder (i.e., source sequence) and decoder (i.e., generated target sequence) to predict the probability of the next target token.

Encoder-decoder models have demonstrated their effectiveness in many sequence transduction problems, since neural networks are good at context encoding and conditional generation. However, as we mentioned earlier, the task of semantic parsing involves the transduction from a text sequence into a tree or graph. To solve the problem, we will argument the vanilla encoder-decoder model with a stack-LSTM decoder. Before we introduce the model details, we first discuss the semantic formalism since it determines how the output is structured.

3.2 Methodology

3.2.1 Recursive Logical Forms

In this chapter, we adopt the functional queries (FunQL) as our semantic formalism. FunQL is a LISP-style, recursive meaning representation language which maps simple first order logical forms to functional operators that abstract away from variables and quantifiers (Kate and Mooney, 2006). The language is also closely related to lambda-DCS (Liang, 2013), which makes existential quantifiers implicit. Lambda-DCS is more compact in the sense that it can use variables in cases to handle anaphora and build composite binary predicates.

The FunQL logical forms we define contain the following primitive functional operators. They overlap with simple lambda-DCS (Berant et al., 2013b) but differ slightly in syntax to ease recursive generation of logical forms. Let l denote a logical form, $[\![l]\!]_{\mathcal{K}}$ represent its denotation, and $\mathcal{K}$ refers to a knowledge base. We write the denotation as $[\![l]\!]$ for simplicity.

- Unary base case: An `entity` e (e.g., `Barack Obama`) is a unary logical form whose denotation is a singleton set containing that entity:

$$[\![e]\!] = \{e\} \tag{3.8}$$

- Binary base case: A `relation` r (e.g., `daughterOf`) is a binary logical form with denotation:

$$[\![r]\!] = \{(e_1, e_2) : (e_1, r, e_2) \in \mathcal{K}\} \tag{3.9}$$

- A relation r can be applied to an entity e_1 (written as $r(e_1)$) and returns as denotation the unary satisfying the relation:

$$[\![r(e_1)]\!] = \{e : (e_1, e) \in [\![r]\!]\} \tag{3.10}$$

  For example, the expression `daughterOf(Barack Obama)` corresponds to the question "*Who are Barack Obama's daughters?*".

- `count` returns the cardinality of the unary set u:

$$[\![\texttt{count}(u)]\!] = \{|[\![u]\!]|\} \tag{3.11}$$

  For example, `count(daughterOf(Barack Obama))` represents the question "*How many daughters does Barack Obama have?*".

- `argmax` or `argmin` return a subset of the unary set u whose specific relation r is maximum or minimum:

$$[\![\texttt{argmax}(u, r)]\!] = \{e : e \in u \cap \forall e' \in u, r(e) \geq r(e')\} \tag{3.12}$$

  For example, the expression `argmax(daughterOf(Barack Obama), age)` corresponds to the utterance "*Who is Barack Obama's eldest daughter?*".

- `filter` returns a subset of the unary set u where a comparative constraint ($=$, $!=$, $>$, $<$, $\geq$, $\leq$) acting on the relation r is satisfied:

$$[\![\texttt{filter}_{>}(u, r, v)]\!] = \{e : e \in u \cap r(e) > v\} \tag{3.13}$$

For example, the query `filter`$_{>}$ `(daughterOf(Barack Obama), age, 5)` returns the daughters of Barack Obama who are older than five years.

- `and` takes the intersection of two urinary sets u_1 and u_2:

$$[\![\texttt{and}(u_1,u_2)]\!] = [\![u_1]\!] \cap [\![u_2]\!] \tag{3.14}$$

while `or` takes their union:

$$[\![\texttt{or}(u_1,u_2)]\!] = [\![u_1]\!] \cup [\![u_2]\!] \tag{3.15}$$

For example, the expression `and(daughterOf(Barack Obama), InfluentialTeensByYear( 2014))` would correspond to the query "*Which daughter of Barack Obama was named Most Influential Teens in the year 2014?*".

The operators just defined give rise to highly compositional logical forms (e.g., `count( and(daughterOf(Barack Obama), InfluentialTeensByYear( 2014 ) )`.

The reason for using FunQL in our framework lies in its recursive nature which allows us to model the process of generating logical form as a sequence of transition actions, which can be decoded by powerful recurrent neural networks. We extend the parser to handle non-recursive meaning representations in Chapter 5. We next describe how our semantic representation is integrated with a transition-based tree-generation algorithm to produce tree-structured logical forms.

3.2.2 Tree Generation Algorithm

We now describe a generic tree generation algorithm which recursively generates tree constituents with a set of transition actions. The key insight underlying our algorithm is to define a canonical traversal or generation order, which generates a tree as a transition sequence. A transition sequence for a tree is a sequence of configuration-transition pairs $[(c_0,t_0),(c_1,t_1),\cdots,(c_m,t_m)]$. In this work, we consider two commonly used generation orders, namely top-down pre-order and bottom-up post-order.

The **top-down** system is specified by the tuple $c = (\sum, \pi, \sigma, N, P)$ where $\sum$ is a stack used to store partially complete tree fragments, π is non-terminal token to be generated, σ is the terminal token to be generated, N is a stack of open non-terminals, and P is a function indexing the position of a non-terminal pointer. The pointer indicates where subsequent children nodes should be attached (e.g., $P(X)$ means that the pointer is pointing to the non-terminal X). The initial configuration of the transition system is

Top-down Transitions	
NT(X)	$([\sigma\|X'], X, \varepsilon, [\beta\|X'], P(X')) \Rightarrow ([\sigma\|X', X], \varepsilon, \varepsilon, [\beta\|X', X], P(X))$
TER(x)	$([\sigma\|X'], \varepsilon, x, [\beta\|X'], P(X')) \Rightarrow ([\sigma\|X', x], \varepsilon, \varepsilon, [\beta\|X', x], P(X'))$
RED	$([\sigma\|X', X, x], \varepsilon, \varepsilon, [\beta\|X', X], P(X)) \Rightarrow ([\sigma\|X', X(x)], \varepsilon, \varepsilon, [\beta\|X'], P(X'))$

Bottom-up Transitions	
TER(x)	$(\sigma, \varepsilon, x) \Rightarrow ([\sigma\|x], \varepsilon, \varepsilon)$
NT-RED(X)	$([\sigma\|x], X, \varepsilon) \Rightarrow ([\sigma\|X(x)], \varepsilon, \varepsilon)$

Table 3.1: Transitions for top-down and bottom-up generation system. Stack $\sum$ is represented as a list with its head to the right (with tail σ), same for stack N (with tail β).

$c_0 = ([], TOP, \varepsilon, [], \perp)$, where TOP stands for the root node of the tree, ε represents an empty string, and $\perp$ represents an unspecified function. The top-down system employs three transition actions defined in Table 3.1:

- NT(X) creates a new subtree non-terminal node denoted by X. The non-terminal X is pushed on top of the stack and written as X(. Subsequent tree nodes are generated as children underneath X.
- TER(x) creates a new child node denoted by x. The terminal x is pushed on top of the stack, written as x.
- RED is the reduce action which indicates that the current subtree being generated is complete. The non-terminal root of the current subtree is closed and subsequent children nodes will be attached to the predecessor open non-terminal. Stack-wise, RED recursively pops children (which can be either terminals or completed subtrees) on top until an open non-terminal is encountered. The non-terminal is popped as well, after which a completed subtree is pushed back to the stack as a single closed constituent, written for example as X1(X2, X3).

We define the **bottom-up** system by tuple $c = (\sum, \pi, \sigma)$ where $\sum$ is a stack used to store partially complete tree fragments, π is the non-terminal to be generated, and σ is the terminal to be generated. We take the initial parser configuration to be $c_0 = ([], x_l, \varepsilon)$, where x_l stands for the leftmost terminal node of the tree, and ε represents

an empty string. The bottom-up generation uses two transition actions defined in Table 3.1:

- `TER(x)` creates a new terminal node denoted by `x`. The terminal `x` is pushed on top of the stack, written as `x`.
- `NT-RED(X)` builds a new subtree by attaching a parent node (denoted by `X`) to children nodes on top of the stack. The children nodes can be either terminals or smaller subtrees. Similarly to `RED` in the top-down case, children nodes are first popped from the stack, and subsequently combined with the parent `X` to form a subtree. The subtree is pushed back to the stack as a single constituent, written for example as `X1(X2, X3)`. A challenge with `NT-RED(X)` is to decide how many children should be popped and included in the new subtree. In this work, the number of children is dictated by the number of arguments expected by `X` which is in turn constrained by the logical language. For example, from the FunQL grammar it is clear that `count` takes one argument and `argmax` takes two. The language we use does not contain non-terminal functions with a variable number of arguments.

Note that although top-down generation is defined by three transition actions, while bottom-up order applies two transition actions only (since it combines reduce with non-terminal generation), the amount of action predictions required are the same for the two systems. The reason is that the reduce action in the top-down system is deterministic when the FunQL grammar is used as a constraint, which will be described below.

3.2.3 Generating Tree-structured Logical Forms

To generate tree-structured logical forms, we integrate the generic transition actions with FunQL, whose grammar determines the space of allowed terminal and non-terminal symbols:

- `NT(X)` includes transition actions that generates relations `NT(relation)`, and other domain-general operators in FunQL: `NT(and)`, `NT(or)`, `NT(count)`, `NT(argmax)`, `NT(argmin)` and `NT(filter)`. Note that `NT(relation)` creates a placeholder for a relation. The relation will be predicted by a domain-specific classifier.
- `TER(X)` includes two transition actions: `TER(relation)` for generating relations and `TER(entity)` for generating entities. Both transition actions create a placeholder

Operation	Logical form token	Stack
NT(count)	count	count(
NT(and)	and	count(|| and(
NT(relation)	daughterOf	count(|| and(|| daughterOf
TER(entity)	Barack Obama	count(|| and(|| daughterOf(|| Barack Obama
RED		count(|| and(|| daughterOf(Barack Obama)
NT(relation)	InfluentialTeensByYear	count(|| and(|| daughterOf(Barack Obama) || InfluentialTeensByYear(
TER(entity)	2014	count(|| and(|| daughterOf(Barack Obama) || InfluentialTeensByYear(|| 2014
RED		count(|| and(|| daughterOf(Barack Obama) || InfluentialTeensByYear(2014)
RED		count(|| and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))
RED		count(and(daughterOf(Barack Obama), InfluentialTeensByYear(2014)))

Table 3.2: Top-down generation of the logical form `count(and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))`. Elements on the stack are separated by || and the top of the stack is on the right.

for a relation or an entity. The relation or entity will be predicted by domain-specific classifiers.

- `NT-RED(X)` includes `NT-RED(relation)`, `NT-RED(and)`, `NT-RED(or)`, `NT-RED(count)`, `NT-RED(argmax)`, `NT-RED(argmin)` and `NT-RED(filter)`. Again, `NT-RED(relation)` creates a placeholder for a relation, which is subsequently predicted by a domain-specific classifier.

Table 3.2 illustrates the sequence of transition actions employed by our parser in order to generate the logical form `count(and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))` top-down. Table 3.3 shows how the same logical form is generated bottom-up. Note that the examples are simplified for illustration purposes; the logical form is generated conditioned on an input utterance, such as "*How many daughters of Barack Obama were named Most Influential Teens in the year 2014?*".

Given enough training data, we expect that neural networks are capable of figuring out the valid action and token sequence that constitutes a valid logical form. However, a challenge is to ensure the outputs are always well-formed and meaningful, to facilitate downstream applications. To this end, we incorporate two types of constraints in our system. The first ones are structural constraints to ensure that the outputs are syntactically valid logical forms. For the top-down system these constraints include:

- The first transition action must be `NT`;
- `RED` cannot directly follow `NT`;

Operation	Logical form token	Stack
TER(entity)	Barack Obama	Barack Obama
NT-RED(relation)	daughterOf	daughterOf(Barack Obama)
TER(entity)	2014	daughterOf(Barack Obama) || 2014
NT-RED(relation)	InfluentialTeensByYear	daughterOf(Barack Obama) || InfluentialTeensByYear(2014)
NT-RED(and)	and	and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))
NT-RED(count)	count	count(and(daughterOf(Barack Obama), InfluentialTeensByYear(2014)))

Table 3.3: Bottom-up generation of the logical form `count(and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))`. Elements on the stack are separated by || and the top of the stack is on the right.

- The maximum number of open non-terminal symbols allowed on the stack is 10. NT is disabled when the maximum number is reached;
- The maximum number of (open and closed) non-terminal symbols allowed on the stack is 10. NT is disabled when the maximum number is reached.

Tree constraints for the bottom-up system are:

- The first transition action must be TER;
- The maximum number of consecutive TERs allowed is 5;
- The maximum number of terminal symbols allowed on the stack is the number of words in the sentence. TER is disallowed when the maximum number is reached.

The second type of constraints relate to the FunQL-grammar itself, ensuring that the generated logical forms are meaningful for execution:

- The type of argument expected by each non-terminal symbol must follow the FunQL grammar;
- The number of arguments expected by each non-terminal symbol must follow the FunQL grammar;
- When the expected number of arguments for a non-terminal symbol is reached, a RED action must be called for the top-down system; for the bottom-up system this constrain is built within the NT-RED action, since it reduces the expected number of arguments based on a specific non-terminal symbol.

3.2.4 Neural Network Realizer

Next we explain how the logical form generation algorithm is realized with a neural network. The model can be viewed as an encoder-decoder model, where the encoder is a bidirectional LSTM to encode an utterance sequence, and the decoder is a stack-LSTM to generate a tree-structured logical form.

Bidirectional LSTM Encoder Utterance x is encoded with a bidirectional LSTM architecture. A bidirectional LSTM is comprised of a forward LSTM and a backward LSTM. The forward LSTM processes a variable-length sequence $x = (x_1, x_2, \cdots, x_n)$ from left to right. For simplicity, we denote the recurrent computation of the forward LSTM as:

$$\overrightarrow{h_t} = \overrightarrow{\text{LSTM}}(x_t, \overrightarrow{h_{t-1}}) \tag{3.16}$$

After encoding, a list of forward representations $[\overrightarrow{h_1}, \overrightarrow{h_2}, \cdots, \overrightarrow{h_n}]$ within the forward context is obtained. Similarly, the backward LSTM processes the sequence from right to left, computing a list of token representations $[\overleftarrow{h_1}, \overleftarrow{h_2}, \cdots, \overleftarrow{h_n}]$ within the backward context as:

$$\overleftarrow{h_t} = \overleftarrow{\text{LSTM}}(x_t, \overleftarrow{h_{t+1}}) \tag{3.17}$$

Finally, each input token x_i is represented by the concatenation of its forward and backward LSTM state vectors, denoted by $h_i = \overrightarrow{h_i} : \overleftarrow{h_i}$. The list storing token vectors for the entire utterance x can be considered as a buffer, in analogy to syntactic parsing. A notable difference is that tokens in the buffer will not be removed since its alignment to logical form tokens is not pre-determined in the general semantic parsing scenario. We denote the buffer b as $b = [h_1, \cdots, h_k]$, where k denotes the length of the utterance.

Stack-LSTM Decoder The generation history, aka partially completed subtrees, is encoded with a variant of stack-LSTM (Dyer et al., 2015), which also serves as the decoder for logical forms. Predictions will be subsequently made based on the generation history and also the utterance. The stack-LSTM captures not only previously generated tree tokens but also tree structures. We first discuss the stack-LSTM in the top-down transition system and then present modifications to account for the bottom-up system.

In **top-down** transitions, actions `NT` and `TER` change the stack-LSTM representation s_t which we compute as:

$$s_t = \text{LSTM}(y_t, s_{t-1}) \tag{3.18}$$

where y_t denotes the newly generated non-terminal or terminal token. A `RED` action recursively pops the stack-LSTM states as well as corresponding tree tokens on the

output stack. The popping stops when a non-terminal state is reached and popped, after which the stack LSTM reaches an intermediate state $s_{t-1:t}$[1]. The representation of the completed subtree u is then computed as:

$$u = W_u \cdot [p_u : c_u] \tag{3.19}$$

where p_u denotes the parent (non-terminal) embedding of the subtree, c_u denotes the average of the children (terminal or completed subtree) embeddings, and W_u denotes the weight matrix. Finally, the subtree embedding u serves as the input to the LSTM and updates $s_{t-1:t}$ to s_t as:

$$s_t = \text{LSTM}(u, s_{t-1:t}) \tag{3.20}$$

Figure 3.1 provides a graphical view on how the three transition actions change the configuration of a stack-LSTM.

In comparison, the **bottom-up** transition system uses the same `TER` action to update the stack LSTM representation s_t when a terminal y_t is newly generated:

$$s_t = \text{LSTM}(y_t, s_{t-1}) \tag{3.21}$$

Differently, the effects of `NT` and `RED` are merged into a `NT-RED(X)` action. When `NT-RED(X)` is invoked, a non-terminal y_t is first predicted and then the stack LSTM starts popping its states on the stack. The number of pops is decided by the amount of argument expected by y_t. After that, a subtree can be obtained by combining the non-terminal y_t and the newly popped terminal tokens, while the stack LSTM reaches an intermediate state $s_{t-1:t}$. Similar to the top-down system, we compute the representation of the newly combined subtree u as:

$$u = W_u \cdot [p_u : c_u] \tag{3.22}$$

where p_u denotes the parent (non-terminal) embedding of the subtree, c_u denotes the average of the children (terminal or completed subtree) embeddings, and W_u denotes the weight matrix. Finally, the subtree embedding u serves as the input to the LSTM and updates $s_{t-1:t}$ to s_t as:

$$s_t = \text{LSTM}(u, s_{t-1:t}) \tag{3.23}$$

The key difference here is that a non-terminal tree token is never pushed alone to update the stack-LSTM, but rather as part of a completed subtree that does the update.

[1] We use $s_{t-1:t}$ to denote the intermediate transit state from time step $t-1$ to t, after terminal tokens are popped from the stack; s_t denotes the final LSTM state after the subtree representation is pushed back to the stack (as explained in the following).

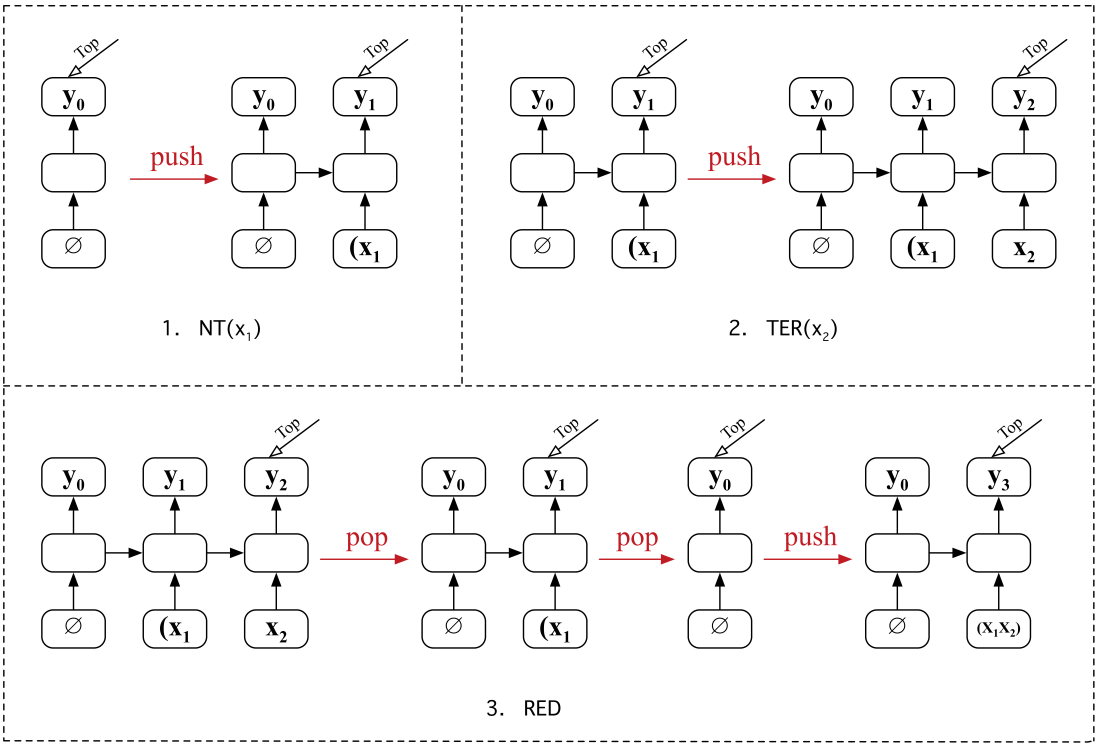


Figure 3.1: A stack-LSTM extends a standard LSTM with the addition of a stack pointer (shown as Top in the figure). The example shows how the configuration of the stack changes when the transition actions `NT`, `TER`, and `RED` are applied in sequence. The initial stack is presumed empty for illustration purposes. We only show how the stack-LSTM updates its states, not how subsequent predictions are made which depend not only on the hidden state of the stack-LSTM, but also on the natural language utterance.

Predictions Given representations of the utterance and generation history, our model makes two types of predictions pertaining to transition actions and logical form tokens (see Table 3.2). First, at every time step, the next transition action a_{t+1} is predicted based on the utterance encoding b and the generation history s_t:

$$a_{t+1} \sim f(b, s_t) \tag{3.24}$$

where f is a neural network that computes the parameters of a categorical distribution over the action space which is restricted by the constraints discussed in Section 3.2.3.

Next, the logical form token underlying each transition action must be emitted. When the transition action contains one of the domain-general non-terminals `count`, `argmax`, `argmin`, `and`, `or`, and `filter` (e.g., `NT(count)`), the logical form token is the corresponding non-terminal (e.g., `count`). When the transition action involves one of the placeholders `entity` or `relation` (e.g., `NT(relation)`, `TER(relation)` and

`TER(entity)`), a domain-specific logical form token y_{t+1} (i.e., an entity or a relation) is predicted in a fashion similar to action prediction:

$$y_{t+1} \sim g(b, s_t) \tag{3.25}$$

where g is a neural network that computes the parameters of a categorical distribution over the token space.

A remaining challenge lies in designing predictive functions f (for the next action) and g (for the next logical form token) in the context of semantic parsing. We explore various attention mechanisms which we discuss in the next sections.

3.2.5 Transition Action Prediction

This section explains the exploration of predictive functions f for the next action. To predict the next action, we compute at each time step t a single adaptive utterance or buffer representation $\bar{b}_t$ with a soft attention mechanism:

$$u_t^i = V^T \tanh(W_b b_i + W_s s_t) \tag{3.26}$$

$$\alpha_t^i = \text{softmax}(u_t^i) \tag{3.27}$$

$$\bar{b}_t = \sum_i \alpha_t^i b_i \tag{3.28}$$

where W_b and W_s are weight matrices and V is a weight vector. We then combine the representation of the buffer (i.e. the utterance) and the stack (i.e. the generation history) with a feed-forward neural network (Equation (3.29)) to yield a feature vector for the generation system. Finally, softmax is taken to obtain the parameters of the categorical distribution over actions:

$$a_{t+1} \sim \text{softmax}(W_{oa} \tanh(W_f[\bar{b}_t, s_t])) \tag{3.29}$$

where W_{oa} and W_f are weight matrices.

3.2.6 Logical Token Prediction

This section presents various functions g for predicting the next logical form token (i.e., a specific `entity` or `relation`). A hurdle in semantic parsing concerns handling mismatches between natural language and logical predicates corresponding to the knowledge base of interest. For example, both utterances "*Where did X graduate from*" and "*Where did X get his PhD*" would trigger the same predicate `education` in Freebase.

Traditional semantic parsers map utterances directly to domain-specific logical forms relying exclusively on a set of lexicons either predefined or learned for the target domain with only limited coverage. Recent approaches alleviate this issue by firstly mapping the utterance to a domain-general logical form which aims to capture language-specific semantic aspects, after which ontology matching is performed to handle mismatches (Kwiatkowski et al., 2013; Reddy et al., 2014, 2016). Beyond efficiency considerations, it remains unclear which domain-general representation is best suited to domain-specific semantic parsing.

Neural networks provide an alternative solution: the matching between natural language and domain-specific predicates is accomplished via an attention layer, which encodes a context-sensitive probabilistic lexicon. This is analogous to the application of the attention mechanism in machine translation (Bahdanau et al., 2015), which is used as an alternative to conventional phrase tables. In this work, we consider a practical domain-specific semantic parsing scenario where we are given no lexicon. We first introduce the basic form of attention used to predict logical form tokens and then discuss various extensions as shown in Figure 3.3.

Soft Attention In the case where no lexicon is provided, we use a soft attention layer similar to action prediction. The parameters of the soft attention layer prior to softmax are shared with those used in action prediction:

$$u_t^i = V \tanh(W_b b_i + W_s s_t) \tag{3.30}$$

$$\alpha_t^i = \text{softmax}(u_t^i) \tag{3.31}$$

$$\bar{b}_t = \sum_i \alpha_t^i b_i \tag{3.32}$$

$$y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f[\bar{b}_t, s_t])) \tag{3.33}$$

which outputs the parameters of the categorical distribution over logical form tokens (either predicates or entities). When dealing with extremely large knowledge bases with many predicates or entities, the output space can be pruned and restricted with an entity lexicon. This method requires us to identity potential entity candidates in the sentence, and then generate only entities belonging to this subset and the relations linking them.

Structured Soft Attention Next we explored a structured attention layer (Kim et al., 2017; Liu and Lapata, 2017) to encourage the model to attend to contiguous natural language phrases when generating a logical token, while being differentiable.

The structured attention layer we adopt is a linear-chain conditional random field (CRF; Lafferty et al. (2001). Assume that at time step t each word in the buffer (e.g., the ith word) is assigned an attention label $A_t^i \in \{0,1\}$. The CRF defines $p(A_t)$, the probability of the sequence of attention labels at time step t as:

$$p(A_t) = \frac{\exp \sum_i W_f \cdot \psi(A_t^{i-1}, A_t^i, b_i, s_t)}{\sum_{A_t^1,\cdots,A_t^n} \exp \sum_i W_f \cdot \psi(A_t^{i-1}, A_t^i, b_i, s_t)} \tag{3.34}$$

where $\sum_i$ sums over all words and $\sum_{A_t^1,\cdots,A_t^n}$ sums over all possible sequences of attention labels. W_f is a weight vector and $\psi(A_t^{i-1}, A_t^i, b_i, s_t)$ a feature vector. In this work the feature vector is defined with three dimensions: the state feature for each word:

$$u_t^i \cdot A_t^i \tag{3.35}$$

where u_t^i is the word-specific attention score computed in Equation (3.30); the transition feature:

$$A_t^{i-1} \cdot A_t^i \tag{3.36}$$

and the context-dependent state feature

$$u_t^i \cdot A_t^{i-1} \cdot A_t^i \tag{3.37}$$

The outcome of the CRF layer is a list of marginal probabilities (denoted by α_t^i) computed by the forward-backward message passing algorithm (Lafferty et al., 2001):

$$\alpha_t^i = \text{forward-backward}(u_t^i) \tag{3.38}$$

Two choices exist regarding how these marginal probabilities (αs) should be used subsequently: since each of the αs denotes a word-specific bernoulli distribution, the first choice is to directly use αs to compute an adaptive buffer representation as follows:

$$\bar{b}_t = \sum_i \alpha_t^i b_i \tag{3.39}$$

The other choice is to renormalize the word-specific marginal probabilities to yield a categorial distribution over all words—which is more comparable to the standard soft attention:

$$\bar{\alpha}_t^i = \text{softmax}(\alpha_t^i) \tag{3.40}$$

and compute the adaptive buffer representation with the renormalized probablities:

$$\bar{b}_t = \sum_i \bar{\alpha}_t^i b_i \tag{3.41}$$

Objective: Predict the next logical form token given the current stack representation s_t and n input word representations in the buffer $b_1 \cdots b_n$.
Steps:

1. Compute the logit u_t^i of each input word b_i as $u_t^i = V \tanh(W_b b_i + W_s s_t)$. The logit will be used to compute the first and third feature in ψ.
2. Forward algorithm: Initialize $\beta(A_t^1) = 1$.
   For $i \in \{2 \cdots n\}$, $A_t^i \in \{0,1\}$: $\beta(A_t^i) = \sum_{A_t^{i-1} \in \{0,1\}} \beta(A_t^{i-1}) \times \psi(A_t^{i-1}, A_t^i, b_i, s_t)$, where ψ is the context-dependent feature vector.
3. Backward algorithm: Initialize $\gamma(A_t^n) = 1$.
   For $i \in \{1 \cdots (n-1)\}$, $A_t^i \in \{0,1\}$: $\gamma(A_t^i) = \sum_{A_t^{i+1} \in \{0,1\}} \gamma(A_t^{i+1}) \times \psi(A_t^i, A_t^{i+1}, b_i, s_t)$, where ψ is the context-dependent feature vector.
4. Compute the marginal probability α_t^i of each input word b_i: $\alpha_t^i = \beta(A_t^i) \times \gamma(A_t^i)$
5. Apply soft attention to compute an adaptive buffer representation: $\bar{b}_t = \sum_i \alpha_t^i b_i$
6. Predict the next logical token: $y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f[\bar{b}_t, s_t]))$
7. Compute the error and backpropagate.

Figure 3.2: The structured attention model for logical token prediction.

As an empirical choice, we adopted the first choice and ignored the renormalization step in this work. The adaptive buffer representation is used to compute a distribution of output logical form tokens:

$$y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f[\bar{b}_t, s_t])) \tag{3.42}$$

Hard Attention Soft attention learns a probablistic mapping between natural language and logical tokens. At every time step, every word in the utterance is assigned the probability of triggering every logical predicate. In order to render the inner workings of the model more transparent we explore the use of a hard attention mechanism as a means of rationalizing neural predictions by inducing a lexicon.

At each time step, hard attention computes the attention probability of using a word x_t to predict y_{t+1}:

$$u_t^i = V \tanh(W_b b_i + W_s s_t) \tag{3.43}$$

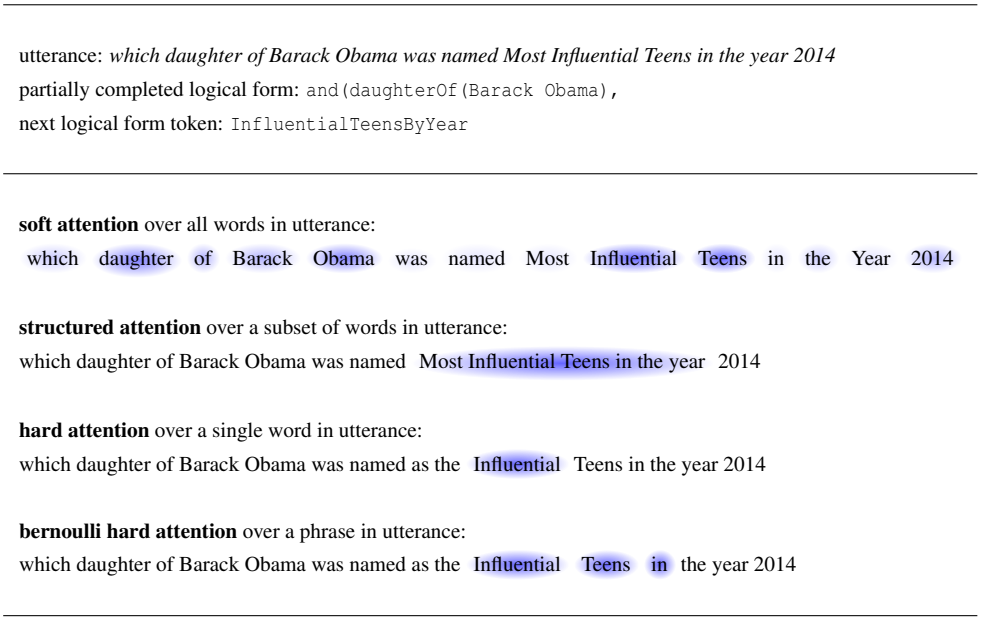


Figure 3.3: Different attention mechanisms for predicting the next logical form token. The example utterance is *which daughter of Barack Obama was named Most Influential Teens in the year 2014?* and the corresponding logical form to be generated is `and(daughterOf(Barack Obama), InfluentialTeensByYear(2014))`. The figure shows attention for predicting `InfluentialTeensByYear`. Darker shading indicates higher values.

$$x_t \sim \text{softmax}(u_t^i) \tag{3.44}$$

The representation of x_t denoted by b_t is then used to predict the logical token:

$$y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f[b_t, s_t])) \tag{3.45}$$

As will become clearer in Section 3.2.7, hard attention maximizes a lower bound of the marginal likelihood of the logical form token (and also the entire logical form). The objective is differentiable, but it is not tractable without inconvenient independence assumptions: the computation requires us to sum over all the possible intermediate "natural language structures". In Section 3.2.7, we present a sampling-based algorithm and a baseline method to reduce the variance of the predictor.

Bernoulli Hard Attention A limitation of hard attention lies in selecting a single word to attend to at each time step. In practice, a logical form predicate is often triggered by a natural language phrase or a multi-word expression. A way to overcome this limitation is to compute a bernoulli distribution for every word separately, indicating the probability of the word being selected. Then an attention label is assigned to each

word based on this probability (e.g., with threshold 0.5). Let $A_t^i \in \{0,1\}$ denote the attention label of the ith word at time step t. Using the unnormalized attention score u_t^i computed in Equation (3.30), we obtain the probability $p(A_t^i = 1)$ as:

$$p(A_t^i = 1) = \text{logistic}(u_t^i) \tag{3.46}$$

where logistic denotes a sigmoid function.

We compute the adaptive buffer representation as an average of the selected word embeddings:

$$\bar{b}_t = \frac{1}{\sum_i A_t^i} \sum_i A_t^i b_i \tag{3.47}$$

which is then used to compute a distribution of the output logical form tokens:

$$y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f[\bar{b}_t, s_t])) \tag{3.48}$$

3.2.7 Training

In the fully supervised setup, the training data consists of utterance-logical form pairs. Consider utterance x with logical form l whose structure is determined by a sequence of transition actions a and a sequence of logical form tokens y. Our ultimate goal is to maximize the conditional likelihood of the logical form given the utterance for all training data:

$$\mathcal{L} = \sum_{(x,l)\in\mathcal{T}} \log p(l|x) \tag{3.49}$$

which can be decomposed into the action likelihood and the token likelihood:

$$\log p(l|x) = \log p(a|x) + \log p(y|x,a) \tag{3.50}$$

Soft attention The above objective consists of two terms, one for the action sequence:

$$\mathcal{L}_a = \sum_{(x,l)\in\mathcal{T}} \log p(a|x) = \sum_{(x,l)\in\mathcal{T}} \sum_{t=1}^{n} \log p(a_t|x) \tag{3.51}$$

and one for the logical form token sequence:

$$\mathcal{L}_y = \sum_{(x,l)\in\mathcal{T}} \log p(y|x,a) = \sum_{(x,l)\in\mathcal{T}} \sum_{t=1}^{n} \log p(y_t|x,a_t) \tag{3.52}$$

These constitute the training objective for fully differentiable neural semantic parsers, when (basic or structured) soft attention is used.

Hard attention In the case when hard attention is used for token prediction, the objective $\mathcal{L}_a$ remains the same but $\mathcal{L}_y$ differs. This is because the attention layer is non-differentiable for errors to backpropagate through. We use the alternative REINFORCE algorithm (Williams, 1992) for backpropagation. In this scenario, the neural attention layer is used as a policy predictor to emit attention choice, while subsequent neural layers are used as the value function to compute a reward—a lower bound of the log likelihood $\log p(y|x,a)$. Let u_t denote the (latent) attention choice at each time step t; we maximize the expected log likelihood of the logical form token given the overall attention choice for all examples, which by Jensen's Inequality is the lower bound on the log likelihood $\log p(y|x,a)$:

$$\begin{aligned}\mathcal{L}_y &= \sum_{(x,l)\in\mathcal{T}} \sum_u [p(u|x,a)\log p(y|u,x,a)] \\ &\leq \sum_{(x,l)\in\mathcal{T}} \log \sum_u [p(u|x,a)p(y|u,x,a)] \\ &= \sum_{(x,l)\in\mathcal{T}} \log p(y|x,a)\end{aligned} \tag{3.53}$$

The gradient of $\mathcal{L}_y$ with respect to model parameters θ is given by:

$$\begin{aligned}\frac{\partial \mathcal{L}_y}{\partial \theta} &= \sum_{(x,l)\in\mathcal{T}} \sum_u p(u|x,a)\frac{\partial \log p(y|u,x,a)}{\partial \theta} + \log p(y|u,x,a)\frac{\partial p(u|x,a)}{\partial \theta} \\ &= \sum_{(x,l)\in\mathcal{T}} \sum_u p(u|x,a)\frac{\partial \log p(y|u,x,a)}{\partial \theta} + \log p(y|u,x,a)\frac{\partial \log p(u|x,a)}{\partial \theta}p(u|x,a) \\ &= \sum_{(x,l)\in\mathcal{T}} \sum_u p(u|x,a)\left[\frac{\partial \log p(y|u,x,a)}{\partial \theta} + \log p(y|u,x,a)\frac{\partial \log p(u|x,a)}{\partial \theta}\right] \\ &\approx \sum_{(x,l)\in\mathcal{T}} \frac{1}{N}\sum_{k=1}^{K}\left[\frac{\partial \log p(y|u^k,x,a)}{\partial \theta} + \log p(y|u^k,x,a)\frac{\partial \log p(u^k|x,a)}{\partial \theta}\right]\end{aligned} \tag{3.54}$$

which is estimated by the Monte Carlo estimator with K samples. This gradient estimator incurs high variance because the reward term $\log p(y|u^k,x,a)$ is dependent on samples of u^k. An input-dependent *baseline* is used to reduce the variance, which adjusts the gradient update as:

$$\frac{\partial \mathcal{L}_y}{\partial \theta} = \sum_{(x,l)\in\mathcal{T}} \frac{1}{N}\sum_{k=1}^{K}\left[\frac{\partial \log p(y|u^k,x,a)}{\partial \theta} + (\log p(y|u^k,x,a) - b)\frac{\partial \log p(u^k|x,a)}{\partial \theta}\right] \tag{3.55}$$

As baseline, we use the soft attention token predictor described earlier. The effect is to encourage attention samples that return a higher reward than standard soft attention,

while discouraging those resulting in a lower reward. For each training case, we approximate the expected gradient with a single sample of u^k.

3.3 Experiments

We evaluate the fully-supervised training setup on the GEOQUERY (Zelle, 1996) dataset, which contains 880 questions and database queries about US geography. The utterances are compositional, but the language is simple and vocabulary size small (698 entities and 24 relations).

Across training regimes, the dimensions of word vector, logical form token vector, and LSTM hidden state are 50, 50, and 150 respectively. Word embeddings were initialized with Glove embeddings (Pennington et al., 2014). All other embeddings were randomly initialized. We used one LSTM layer in forward and backward directions. Dropout was used on the combined feature representation of the buffer and the stack (Equation (3.29)), which computes the softmax activation of the next action or token. The dropout rate was set to 0.5. Finally, momentum SGD (Sutskever et al., 2013) was used as the optimization method to update the parameters of the model.

Experimental results on GEOQUERY dataset are shown in Table 3.4. The first block contains conventional statistical semantic parsers, previously proposed neural models are presented in the second block, whereas variants of TNSP are shown in the third block. Specifically we build various top-down and bottom-up TNSP using the three types of attention (soft, hard, and structured attention). We report accuracy which is defined as the proportion of the utterances that are correctly parsed to their gold standard logical forms. Amongst various TNSP models, a top-down system with structured attention performs best. Overall, we observe that differences between top down and bottom up systems are small; it is mostly the attention mechanism that affects performance, with hard attention performing worst and structured attention performing best for both top-down and bottom-up systems. TNSP outperforms previously proposed neural semantic parsers which treat semantic parsing as a sequence transduction problem and use LSTMs to map utterances to logical forms (Dong and Lapata, 2016; Jia and Liang, 2016). TNSP yields performance improvements over these systems when using comparable data sources for training. Jia and Liang (2016) achieve better results with synthetic data that expands GEOQUERY; we could adopt their approach to improve model performance, however, we leave this to future work. Our system is on the same par with model of Rabinovich et al. (2017) who also ensure their model outputs well-

Models	Accuracy
Zettlemoyer and Collins (2005a)	79.3
Zettlemoyer and Collins (2007)	86.1
Kwiatkowksi et al. (2010a)	87.9
Kwiatkowski et al. (2011)	88.6
Kwiatkowski et al. (2013)	88.0
Zhao and Huang (2015)	88.9
Liang et al. (2011)	91.1
Dong and Lapata (2016)	84.6
Jia and Liang (2016)	85.0 (89.1)
Rabinovich et al. (2017)	87.1
TNSP, soft attention, top-down	86.8
TNSP, soft structured attention, top-down	87.1
TNSP, hard attention, top-down	85.3
TNSP, bernoulli hard attention, top-down	85.5
TNSP, soft attention, bottom-up	86.1
TNSP, soft structured attention, bottom-up	86.8
TNSP, hard attention, bottom-up	85.3
TNSP, bernoulli hard attention, bottom-up	85.3

Table 3.4: Fully supervised experimental results on the GEOQUERY dataset. For Jia and Liang (2016), we include two of their results: one is a standard neural sequence to sequence model; and the other is the same model trained with a data augmentation algorithm on the labeled data (reported in the parenthesis).

formed trees in a top-down manner using a decoder built of many submodels, each associated with a specific construct in the underlying grammar.

3.4 Summary

In this chapter we propose a fully-supervised neural semantic parser (TNSP) that generates logical forms from given utterances. Across experiments we observe TNSP performs competitively while producing meaningful and well-formed logical forms. One characteristic of the neural semantic parser is that it generates tree-structured representations in an arbitrarily canonical order, as a sequence of transition actions. We investigated

two such orders, top-down pre-order and bottom-up post-order. Experimentally, we observed that pre-order generation provides marginal benefits over post-order generation. One reason for this is that compared to sibling information which the bottom-up system uses, parent information used by the top-down system is more important for subtree prediction.

We explored various attention mechanisms, including soft attention, structured attention and two variants of hard attention. Quantitatively, we observe that soft attention outperforms hard attention. Although hard attention offers interpretability of the induced attention structures but renders training and optimization difficult. The structured attention layer lies in between the two; it is also differentiable since it computes the marginal probability of each token being selected with a dynamic programming procedure. During test, the induced attention structures can be observed with the Viterbi algorithm. We observe that on GEOQUERY which represents the fully supervised setting, structured attention only offers marginal gains over soft attention. However, it should be noted that the structured attention layer at each decoding step requires the forward-backward algorithm, which has time complexity $O(n^2)$ (where n denotes the utterance length) and therefore slower than soft attention which has linear ($O(n)$) complexity.

Overall, the neural semantic parser presented in this chapter contributes to a promising research direction. However, a challenge in fully supervised training setup is to obtain annotated training data at a large scale, which does not easily scale up to complex domain. This motivates us to explore weakly supervised alternatives which we introduce in the next chapter.

Chapter 4

Weakly-supervised Neural Semantic Parsing

In this chapter, we extend the neural semantic parser described in Chapter 3 to a weakly supervised setting, which allows us to train with utterance-denotation pairs. The latter are easier to obtain via crowd-sourcing. The reason is that labeling utterance-denotations does not require any knowledge about the logical language. For example, it is easy to provide an answer to the question "*How many daughters does Barack Obama have*", than writing down the corresponding logical form `count(daughterOf(Barack Obama))`.

However, training a neural semantic parser in a weakly supervised setup is more challenging since we do not have access to gold standard logical forms for backpropagation. Instead, the logical form is modeled as a latent variable and needs to be searched from a large space. To this end, we propose to integrate the neural semantic parser (presented in Chapter 3) with a global ranker, formulating a parser-ranker framework (in Section 4.1). The role of the neural parser is to generate a list of *candidate* logical forms, and the role of the ranker is to select which candidate to use. This modeling approach is widely used in conventional semantic parsers trained with weak supervision (Berant et al., 2013b). To further ease the data collection process and scale semantic parsing to open domains, we explore an even weaker form of supervision—distant supervision where denotations are not provided but artificial training data is generated automatically from declarative sentences crawled from the web. The artificial data is in the form of utterance-denotation pairs, which can be trained with the parser-ranker framework. We evaluate and analyze if such artificial data is helpful for a practical question-answering system (Section 4.2). Finally, we focus on a specific learning challenge

of our approach—logical forms may involve a certain degree of spuriousness—some logical forms coincidentally execute to the correct denotation but they do not match the utterance semantics. We incorporate a generative neural network for detecting spurious logical forms by scoring how a logical form represents the utterance semantics (Section 4.3).

4.1 The Parser-Ranker Framework

4.1.1 Motivation

Conventional weakly-supervised semantic parsing systems separate the parser from the learner (Liang, 2016). The parser, which is often chart-based and non-parameterized, recursively builds derivations for each span of the utterance. The learner, typically a log-linear model, defines features useful for scoring and ranking the set of candidate derivations, based on denotations. As mentioned in Liang (2016), the chart-based parser brings a disadvantage since the system does not support incremental contextual interpretation, because features of a span can only depend on the sub-derivations in that span, as a requirement of dynamic programming.

As a departure from chart-based parsers, a neural semantic parser is itself a parametrized model and is able to leverage global utterance features for decoding. However, training the neural parser directly with utterance-denotation pairs is challenging since the decoder does not have access to gold standard logical forms for backpropagation (in other words, the gold decision at each decision step is not provided). Moreover, it should be noted that the neural decoder suffers from the label bias problem: it generates logical forms with local decoding.

The above facts motivate us to extend the neural semantic parser (presented in 3) with a global ranker to cope with the weak training signal. As mentioned earlier, the role of the neural parser is to generate a list of candidate logical forms, while the ranker aims to leverage global features of utterance-logical form-denotation triples to select which candidate to use for execution during test.

4.1.2 Methodology

Parser We briefly summarize the neural semantic parser described in the previous chapter. The parser consists of a bidirectional LSTM utterance encoder and a stack-LSTM decoder that generates tree-structured logical forms. At each time step of the

decoding, the parser primarily applies soft attention to predict a transition operation that generates a tree following a canonical order. If the transition operation is one that generates a relation or an entity placeholder, the parser further predicts a specific token with either soft, hard, or structured attention, In this way, the most likely logical form l for the utterance x is generated incrementally as:

$$\log p(l|x) = \sum_t \log p(a_t|x, a_{1,\cdots,t-1}, y_{1,\cdots,t-1}) + \log p(y_t|x, a_{1,\cdots,t}, y_{1,\cdots,t-1}) \tag{4.1}$$

where as denote the sequence of transition operations and ys denote the sequence of logical form tokens.

Note that in the weakly supervised setting, the parser decodes a list of candidate logical forms L with beam search, instead of outputting the most likely logical form. During training, the candidate logical forms are executed to find the set of consistent ones (denoted by $L_c(x)$) which lead to the correct denotation. As the training objective, we could maximize the log-likelihood of the correct denotation z by treating logical forms as a latent variable:

$$\log p(z|x) = \log \sum_{l \in L} p(l|x) p(z|x,l) \tag{4.2}$$

where L denotes the set of candidate logical forms generated by the neural parser and l' denotes any logical form in L. Note that $p(z|x,l)$ equates to 1 if the logical form executes to the correct denotation and 0 otherwise. For this reason, we can also write the above equation as

$$\log \sum_{l \in L(c)} p(l|x) \tag{4.3}$$

where $L(c)$ is the set of consistent logical forms which execute to the correct denotation. In our experiment, we adopt a variant of the above training objective by directly maximizing the total log likelihood of consistent logical forms:

$$\sum_{l \in L_c(x)} \log p(l|x) \tag{4.4}$$

This is an empirical choice.

Ranker As mentioned previously, it is impractical to rely solely on a neural decoder to find the most likely logical form at run time in the weakly-supervised setting. One reason is that the decoder incrementally generates logical forms with a sequence of local decisions. This problem can be alleviate with the introduction of a discriminative

ranker, which scores logical forms based on global features of utterance-logical form-denotation triplets.

Specifically we adopt a log linear ranker which computes $p(l|x)$ as

$$p(l|x) = \frac{\exp(f(x,l)\theta)}{\sum_{l' \in L} \exp(f(x,l')\theta)} \tag{4.5}$$

where L is the entire set of candidate logical forms; f is the feature function that maps an utterance-logical form pair (and also the corresponding denotation) into a feature vector; and θ denotes the weight parameter of the model.

The training objective of the ranker is similar to the parser (Equation 4.3). However, the difference lies in how the likelihood $p(l|x)$ is computed. In the parser, the likelihood (denoted by $p_\phi(l|x)$) is factorized into a sequence of conditional probablities (Equaition 4.1) and computed with neural parameters ϕ. These conditional probablities are used to generate logical forms. In comparison, the ranker computes the likelihood (denoted by $p_\theta(l|x)$) in a discriminative fashion—the computation is based on a global scoring function (Equation 4.5) and parameters θ.

Training such a system involves the following steps. Given an input utterance, the neural parser first generates a list of candidate logical forms via beam search. Then these candidate logical forms are executed and those which yield the correct denotation are marked as *consistent* logical forms. The neural parser is then trained to maximize the likelihood of these consistent logical forms $\sum_{l \in L_c} \log p(l|x)$. Meanwhile, the ranker is trained to maximize the marginal likelihood of denotations $\log p(z|x)$.

Clearly, if the parser does not generate any consistent logical forms, no model parameters will be updated. A challenge in this training paradigm is the fact that we rely exclusively on beam search to find good logical forms from an exponential search space. In the beginning of training, neural parameters are far from optimal, and as a result good logical forms are likely to fall outside the beam. We alleviate this problem by performing entity linking (Reddy et al., 2014) which greatly reduces the search space. We determine the identity of the entities mentioned in the utterance according to the knowledge base and restrict the neural parser to generating logical forms containing only those entities.

4.1.3 Experiments

We evaluate our weakly-supervised semantic parser on two datasets: WEBQUESTIONS (Berant et al., 2013a) and GRAPHQUESTIONS (Su et al., 2016). WEBQUESTIONS

contains 5,810 question-answer pairs. It is based on Freebase (Bollacker et al., 2008), a large scale knowledge base created by community members. The questions are not very compositional (i.e., most questions involve a single relational predicate). However, they are real questions asked by people on the web. GRAPHQUESTIONS contains 5,166 question-answer pairs which were created by showing 500 Freebase graph queries to Amazon Mechanical Turk workers and asking them to paraphrase them into natural language. The questions there are more compositional than WEBQUESTIONS. Training and test partitions of the datasets are 0.8 and 0.2, respectively.

We adopt the same training setup as in the fully supervised scenario. The dimensions of word vector, logical form token vector, and LSTM hidden state are 50, 50, and 150 respectively. Word embeddings were initialized with Glove embeddings (Pennington et al., 2014). All other embeddings were randomly initialized. We used one LSTM layer in forward and backward directions. Dropout was used before the prediction layer with a rate set to 0.5. Momentum SGD (Sutskever et al., 2013) was used as the optimization method to update the parameters of the model.

As mentioned earlier, we use entity linking to obtain surrogate logical forms which dramatically reduces the beam search space. For both datasets we perform entity linking following the procedure described in Reddy et al. (2016). We identify potential entity spans using seven handcrafted part-of-speech patterns and associate them with Freebase entities obtained from the Freebase/KG API.[1] We use a structured perceptron trained on the entities found in WEBQUESTIONS and GRAPHQUESTIONS, respectively, to select the top 10 non-overlapping entity disambiguation possibilities. We treat each possibility as a candidate entity to construct candidate logical forms with beam search of size 500, among which we look for the consistent ones.

Recall that in the weakly supervised scenario, our parsing system additionally includes a discriminative ranker, whose role is to select the final logical form to execute from a list of candidates generated by the neural semantic parser. As features of the ranker, we consider the log likelihood outputted by the neural parser, the embedding cosine similarity between the utterance (excluding stop-words) and the logical form, the token overlap count between the two, and also similar features between the lemmatized utterance and the logical form. In addition, we include as features the embedding cosine similarity between the question words and the logical form, the similarity between the question words (e.g., *what, who, where, whose, date, which, how many, count*) and relations (e.g., `president_of_us`, `date_of_birth`) in the logical form, and the

[1] http://developers.google.com/freebase/

similarity between the question words and answer type as indicated by the last word in the Freebase relation (Xu et al., 2016). Finally, we add as a feature the length of the denotation given by the logical form (Berant et al., 2013b). We normalize all features before feeding them into the ranker.

We include results of different model variants (e.g., different attention, generation order) as described in the previous chapter. For all Freebase related datasets, we follow Berant et al. (2013b) and use F1 as the evaluation metric. We report results on WEBQUESTIONS and GRAPHQUESTIONS in Tables 4.1 and 4.2, respectively. The first block in the tables groups conventional semantic parsers, the second block presents related neural models, and the third block variants of our model—the Transition-based Neural Semantic Parser (TNSP). For fair comparison, we also built a baseline sequence-to-sequence model enhanced with an attention mechanism (Dong and Lapata, 2016).

On WEBQUESTIONS the best performing TNSP system generates logical forms based on a top down pre-order while employing soft attention. The same top-down system with structured attention performs closely. In general we observe that our semantic parser obtains performance on par with the best symbolic systems (see the first block in Table4.1). TNSP performs competitively despite not having access to any linguistically-informed syntactic structure. Regarding neural systems (see second block in Table 4.1), our model outperforms the sequence-to-sequence baseline and other related neural architectures using similar resources. Xu et al. (2016) represent the state of the art on WEBQUESTIONS. Their system uses an end-to-end neural network for question answering. In addition, they use Wikipedia to prune out erroneous candidate answers extracted from Freebase. Our model would also benefit from a similar post-processing.

With respect to GRAPHQUESTIONS, we report F1 for various TNSP models (third block in Table 4.2), and conventional statistical semantic parsers (first block in Table 4.2). The first three systems are conventional statistical semantic parsers. The numbers are reported in Su et al. (2016). Again, we observe that a top-down variant of TNSP with soft attention performs the best. It is superior to the sequence-to-sequence baseline and obtains performance comparable to Reddy et al. (2017), which make use of an external syntactic parser. The model of Dong et al. (2017) is state of the art on GRAPHQUESTIONS. Their method is trained end-to-end using questions-answer pairs as a supervision signal together with question paraphrases as a means of capturing different ways of expressing the same content. Importantly, their system is optimized with question-answering in mind, and does not produce logical forms.

Models	F1
SEMPRE (Berant et al., 2013a)	35.7
JACANA (Yao and Van Durme, 2014)	33.0
PARASEMPRE(Berant and Liang, 2014)	39.9
AQQU (Bast and Haussmann, 2015)	49.4
AGENDAIL (Berant and Liang, 2015)	49.7
DEPLAMBDA (Reddy et al., 2016)	50.3
SUBGRAPH (Bordes et al., 2014)	39.2
MCCNN (Dong et al., 2015)	40.8
STAGG (Yih et al., 2015)	48.4 (52.5)
MCNN (Xu et al., 2016)	47.0 (53.3)
Sequence-to-sequence	48.3
TNSP, soft attention, top down	50.1
TNSP, hard attention, top down	49.4
TNSP with structured attention, top down	49.8
TNSP, soft attention, bottom up	49.6
TNSP, hard attention, bottom up	48.4
TNSP, structured attention, bottom up	49.5

Table 4.1: Weakly supervised experimental results on the WEBQUESTIONS datasets. Results with additional resources are shown in parentheses.

When learning from denotations a challenge concerns the handling of an exponentially large set of logical forms. In our approach, we rely on the neural semantic parser to generate a list of candidate logical forms by beam search. Ideally, we hope the beam size is large enough to include good logical forms which will be subsequently selected by the discriminative ranker. Figure 4.1 shows the effect of varying beam size on GRAPHQUESTIONS[2] (dev set) when training executes for two epochs using the TNSP soft attention model with top-down generation order is used. We report the number of utterances that are answerable (i.e., an utterance is considered answerable if the beam includes one or more consistent logical forms leading to the correct denotation) and the number of utterances that are correctly answered eventually. As the beam size

[2]The dataset is chosen as a testbed since it involves more compositional, and difference types of questions.

Models	F1
SEMPRE (Berant et al., 2013a)	10.8
PARASEMPRE (Berant and Liang, 2014)	12.8
JACANA (Yao and Van Durme, 2014)	5.1
SIMPLEGRAPH (Reddy et al., 2016)	15.9
UDEPLAMBDA (Reddy et al., 2017)	17.6
Sequence-to-sequence	16.2
PARA4QA (Dong et al., 2017)	20.4
TNSP with soft attention, top down	17.3
TNSP with hard attention, top down	16.2
TNSP with structured attention, top down	17.1
TNSP with soft attention, bottom up	16.9
TNSP with hard attention, bottom up	16.8
TNSP with structured attention, bottom up	17.1

Table 4.2: Weakly supervised experimental results on the GRAPHQUESTIONS dataset. Results with additional resources are shown in parentheses.

increases, the gap between utterances that are answerable and those that are correctly answered becomes larger. And the curve for correctly answered utterances gradually plateaus and drops slightly. This indicates a trade-off between generating candidates that cover good logical forms and picking the best logical form for execution: when the beam size is large, there is a higher chance for good logical forms to be included but also for the discriminative ranker to make mistakes.

GRAPHQUESTIONS consists of four types of questions. As shown in Table 4.3, the first type are relational questions (denoted by `relation`). An example of a relational question is *what periodic table block contains oxygen*; the second type contains count questions (denoted by `count`). An example is *how many firefighters does the new york city fire department have*; the third type includes aggregation questions requiring argmax or argmin (denoted by `aggregation`). An example is *what human stampede injured the most people*; the last type are filter questions which requires comparisons by $>$, $\geq$, $<$ and *leq* (denoted by `filter`). An example is *which presidents of the united states weigh not less than 80.0 kg*. Table 4.3 shows the number of questions broken down by type, as well as the proportion of answerable and correctly answered

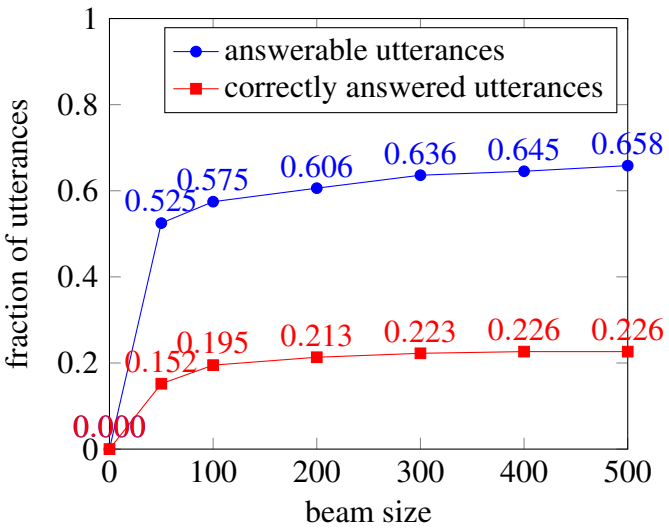


Figure 4.1: Fraction of utterances that are answerable versus those correctly predicted with varying beam size used by TNSP on the GRAPHQUESTIONS dev set.

Utterance type	Number	% Answerable	% Correctly answered
`relation`	1938	0.499	0.213
`count`	309	0.421	0.032
`aggregation`	226	0.363	0.075
`filter`	135	0.459	0.096
All	2,608	0.476	0.173

Table 4.3: Breakdown of questions answered by type for the GRAPHQUESTIONS.

questions. As the results reveal, `relation` questions are the simplest to answer which is expected since `relation` questions are non-compositional and their logical forms are easy to find with beam search. The remaining types of questions are rather difficult to answer: although the system is able to discover logical forms that lead to correct denotations during beam search, the ranker is not able to identify the right logical forms to execute. Aside from the compositional nature of these questions which makes them hard to answer, another difficulty is that such questions are a minority in the dataset posing a learning challenge for the ranker to identify them. As future work, we plan to train separate rankers for different types of questions.

4.2 Distant Supervision

4.2.1 Motivation

Although weakly supervised training allows us to scale semantic parsing to large open-domain problems (Kwiatkowski et al., 2013; Berant et al., 2013b; Yao and Van Durme, 2014), the collection of utterance-denotation pairs still relies on crowd-sourcing. A promising alternative is to train a semantic parser with distant supervision without any annotated logical forms or denotations (Reddy et al., 2014). In this setting, the training data is a collection of unlabeled sentences and a knowledge base. Utterance-denotation pairs are artificially created by replacing entity mentions (which are grounded to KB) in sentences with variables. Then, the semantic parser is trained to predict the denotation for the variable that includes the mentioned entity. For example, given the declarative sentence *NVIDIA was founded by Jen-Hsun Huang and Chris Malachowsky*, the distant supervision approach creates the utterance *NVIDIA was founded by Jen-Hsun_Huang and _blank_* paired with the corresponding denotation *Chris Malachowsky*. In some cases, even stronger constraints can be applied. For example, if the mention is preceded by the word *the*, then the correct denotation includes exactly one entity. In general, the approach converts a corpus of entity-recognized sentences into utterance-denotation pairs on which a weakly supervised training method can be applied. In this section, we test our semantic parser in the distant supervision setting; and more importantly, we evaluate if this approach is useful for practical question answering.

4.2.2 Experiments

To start with, we evaluate our neural semantic parser on the SPADES dataset (Bisk et al., 2016), which contains 93,319 questions derived from CLUEWEB09 (Gabrilovich et al., 2013) sentences. Entity mentions in CLUEWEB09 have been automatically annotated with Freebase entities. Specifically, the questions were created by randomly replacing an entity with a blank symbol, thus producing sentence-denotation pairs (Reddy et al., 2014). The sentences include two or more entities and although they are not very compositional, they constitute a large-scale dataset for neural network training with distant supervision.

Table 4.4 presents experimental results on SPADES. Previous work on this dataset has used a semantic parsing framework where natural language is converted to an intermediate syntactic representation and then grounded to Freebase. Specifically, Bisk

Models	F1
Unsupervised CCG (Bisk et al., 2016)	24.8
Semi-supervised CCG (Bisk et al., 2016)	28.4
Supervised CCG (Bisk et al., 2016)	30.9
Rule-based system (Bisk et al., 2016)	31.4
Sequence-to-sequence	28.6
TNSP with soft attention, top down	32.4
TNSP with hard attention, top down	31.5
TNSP with structured attention, top down	32.1
TNSP with soft attention, bottom up	32.1
TNSP with hard attention, bottom up	30.7
TNSP with structured attention, bottom up	31.4

Table 4.4: Distantly supervised experimental results on the SPADES dataset.

et al. (2016) evaluate the effectiveness of four different CCG parsers on the semantic parsing task when varying the amount of supervision required. As can be seen, TNSP outperforms all CCG variants (from unsupervised to fully supervised) without having access to any manually annotated derivations or lexicons. Again, we observe that a top-down TNSP system with soft attention performs best and is superior to the sequence-to-sequence baseline.

The results on SPADES hold promise for scaling semantic parsing by using distant supervision. In fact, artificial data could potentially help improve weakly supervised question answering models trained on utterance-denotation pairs. To this end, we use the entity-masked declarative sentences paired with their denotations in SPADES as additional training data for GRAPHQUESTIONS. We train the neural semantic parser with the combined training data and evaluate on the GRAPHQUESTIONS. We use the top-down, soft-attention TNSP model with a beam search size of 300. During each epoch of training, the model was first trained with a mixture of the additional SPADES data and the original training data. Figure 4.2 shows the fraction of answerable and correctly answered questions generated by the neural semantic parser on GRAPHQUESTIONS. Note that the original GRAPHQUESTIONS training set consists of 1,794 training examples and we report numbers when different amounts of SPADES training data are used.

As Figure 4.2 shows, using artificial training data is able to improve the neural

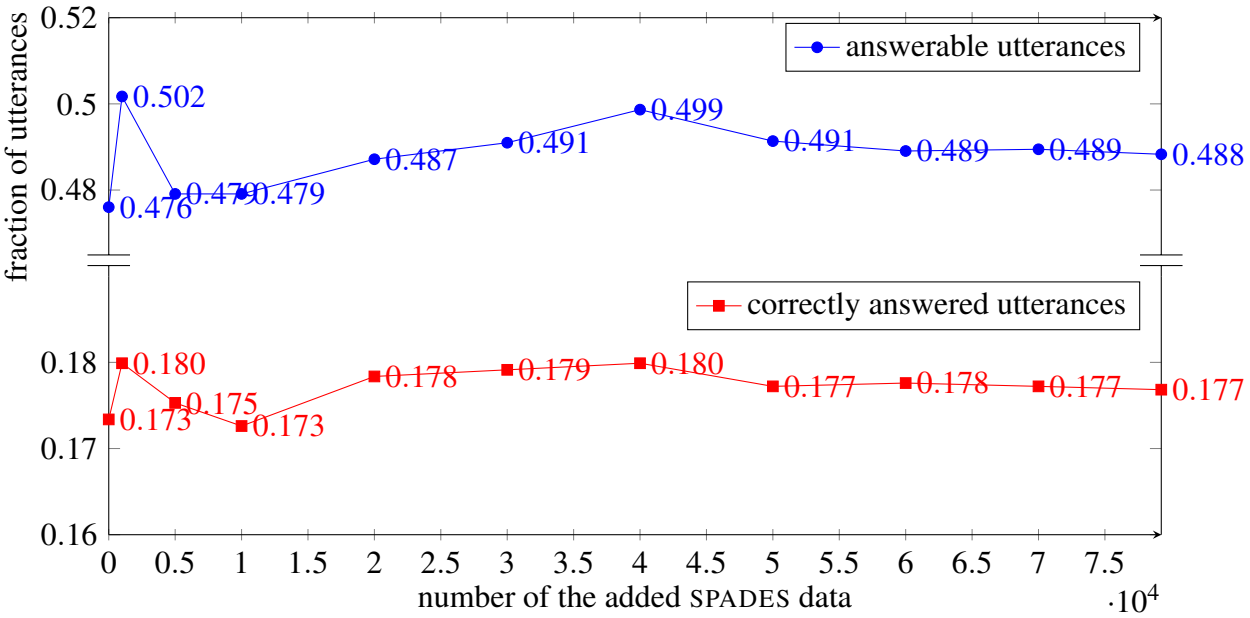


Figure 4.2: Fraction of answerable and correctly answered utterances in the GRAPHQUESTIONS when different amounts of the SPADES data are used.

semantic parser on a question answering task to some extent. This suggests that distant supervision is a promising direction for building practical semantic parsing systems. Since artificial training data can be abundantly generated to fit a neural parser, the approach can be used for data argumentation when question-answer pairs are limited.

However, we observe that the maximum gain occurs when 1,000 extra training examples are used, a size comparable to the original training set. After that no further improvements are made when more training examples are used. We hypothesize this is due to the disparities between utterance-denotation pairs created in distant supervision and utterance-denotation pairs gathered from real users. For example, given the declarative sentence *NVIDIA was founded by Jen-Hsun Huang and Chris Malachowsky*, the distant supervision approach creates the utterance *NVIDIA was founded by Jen-Hsun_Huang and _blank_* and the corresponding denotation *Chris Malachowsky*. However, the actual question users may ask is *Who founded NVIDIA together with Jen-Hsun_Huang*. This poses a challenge if the neural network is trained on one type of utterance and tested on another. We observe that the distribution mismatch outweighs the addition of artificial data quickly. Future work will focus on how to alleviate this problem by generating more realistic data with an advanced question generation module.

Another factor limiting performance is that SPADES mainly consists of relational questions without high-level predicates but GRAPHQUESTIONS does. Examples include

`count`, `filter` and `aggregation` related questions such as *how many firefighters are employed with the new york city*.

4.3 Handling Spurious Logical Forms

4.3.1 Motivation

Weakly-supervised training of the neural parser-ranker system relies on beam search to find consistent logical forms that execute to the correct answer. In this section we focus on a specific learning challenge—handling the spurious ambiguity of logical forms. Typically there are two types of ambiguities. One type comes from the ambiguity of derivations: two different logical forms can be semantically equaivalent, due to the commutative property of certain compositional operations. The other type is the so called spurious ambiguity which we study in this work: some logical forms coincidentally execute to the correct answer but do not match the utterance semantics. For example, both logical forms `former_president(United States)` and `husband_of(Michelle Obama)` gives the same denotation `Barack Obama`, but only the former corresponds to the utterance "*Who is the former president of the USA*". The spurious logical forms act as misleading training signals for the neural semantic parser.

In this section we propose a method for removing spurious logical forms by validating how well a logical form matches the utterance meaning. The intuition is that, a meaning-preserving logical form should be able to provide a high likelihood of reconstructing the original utterance, denoted by $p(x|l)$. However, since correct logical forms are not annotated either, we can not directly compute and maximize $p(x|l)$. To this end, we propose a generative model to measure this reconstruction likelihood.

4.3.2 Methodology

The Generative Neural Network The model assumes utterance x is generated from corresponding logical form l, and only the utterance is observable. The objective is therefore to maximize the log marginal likelihood of the utterance:

$$\log p(x) = \log \sum_{l} p(x,l) \tag{4.6}$$

We adopt neural variational inference Mnih and Gregor (2014) to solve the above objective, which is equivalent to maximizing an evidence lower bound:

$$\log p(x) = \log \frac{q(l|x)p(x|l)p(l)}{q(l|x)} \geq \mathbb{E}_{q(l|x)} \log p(x|l) + \mathbb{E}_{q(l|x)} \log \frac{p(l)}{q(l|x)} \tag{4.7}$$

Since our semantic parser performs constrained decoding and always outputs well-formed logical forms, we assume a uniform prior and the probability $p(l)$ to be a constant. Thus the above objective can be reduced into:

$$\mathbb{E}_{q(l|x)} \log p(x|l) - \mathbb{E}_{q(l|x)} \log q(l|x) = \mathcal{L}(x) \tag{4.8}$$

where the first term computes the reconstruction likelihood $p(x|l)$; and the second term is the entropy of the approximated posterior $q(l|x)$ for regularization. Specifically, $q(l|x)$ is computed with an inference module. In our task-specific context, we consider our semantic parser as the inference module [3]. The reconstruction likelihood $p(x|l)$ is computed with an inverse parser that recovers the utterance x from its logical form l. We use $p(x|l)$ to measure how well the logical form reflects the utterance meaning; details of the inverse parser are described as follows.

Inverse Parser: Stack-LSTM Encoder To reconstruct utterance x, logical form l is first encoded with a stack-LSTM encoder. To do that we deterministically convert the logical form into a sequence of bottom-up transition operations (see Section 3.2.3), which correspond to the creation of tree nodes and completion of subtrees. For example, the transition operations for the logical form `count (and (daughterOf (Barack Obama), InfluentialTeensByYear (2014))` are shown in Table 3.3 as `TER(entity)`, `NT-RED(relation)`, `TER(entity)`, `NT-RED(relation)`, `NT-RED(and)`, `NT-RED(count)`.

The stack-LSTM sequentially processes the transition operations and updates its states accordingly. Similar to the bottom-up generation generation, the bottom-up encoding uses the following two transition operations:

- `TER(X)` encodes the terminal node denoted by `X`. The terminal `X` is pushed on top of the stack, written as `X`.
- `NT-RED(X)` completes the subtree being generated by attaching a parent node (denoted by `X`) to children nodes on top of the stack. Children are first popped from the stack, and subsequently combined with the parent `X` to form a subtree. The subtree is pushed back to the stack as a single constituent.

[3] In the previous section, the output distribution of the semantic parser is denoted by $p(l|x)$.

and the states of the stack-LSTM changes as follows.

When `TER(X)` is called, the stack LSTM state g_t is updated with the terminal y_t as:

$$g_t = \text{LSTM}(y_t, g_{t-1}) \tag{4.9}$$

When `NT-RED(X)` is called, the non-terminal y_t will first be combined with the terminals or subtrees underneath to compute a subtree encoding:

$$u = W_u \cdot [y_t : c] \tag{4.10}$$

where y_t is the parent (non-terminal) embedding of the subtree, c denotes the average of the children (terminal or completed subtree) embeddings, and W_u denotes the weight matrix. After that, the subtree embedding u serves as the input to the LSTM and updates $g_{t-1:t}$ to s_t as:

$$g_t = \text{LSTM}(u, g_{t-1:t}) \tag{4.11}$$

Finally, we save a list of terminal, non-terminal and subtree representations $[g_1, \cdots, g_s]$, where each representation is the stack-LSTM state at the corresponding time step of encoding. The list essentially contains the representation of every tree node and the representation of every subtree (the total number of representations is denoted by s).

Inverse Parser: LSTM Decoder Utterance x is reconstructed with a standard LSTM decoder attending to the tree nodes and subtree representations. At each time step of the decoding, attention is performed over decoder state r_t and tree node representations $[g_1, \cdots, g_s]$:

$$v_t^i = V'^T \tanh(W_{g'} g_i + W_r r_t) \tag{4.12}$$

$$b_t^i = \text{softmax}(v_t^i) \tag{4.13}$$

$$\bar{g}_t = \sum_{i=1}^{s} b_t^i g_i \tag{4.14}$$

and the probability of the next word is predicted as:

$$x'_{t+1} \sim \text{softmax}(W_{x'} \tanh(W_{f'}[\bar{g}_t, r_t])) \tag{4.15}$$

where Ws and V' are all weight parameters.

Optimization The training objective of the generative neural network (i.e. maximizing the reconstruction likelihood for meaning-preserving logical forms) is given in Equation (4.8). Parameters of the neural network include those of the semantic parser (denoted

by θ) and the inverse parser (denoted by ϕ). In order to compute Equation (4.8), the semantic parser first computes $q(l|x)$, where logical forms l are beam searched from this distribution. The logical form is fed as the input to the inverse parser to compute $p(x|l)$.

Differentiating Equation (4.8) with respect to ϕ, we obtain the gradient as:

$$\frac{\partial \mathcal{L}(x)}{\partial \phi} = \mathbb{E}_{q(l|x)} \frac{\partial \log p(x|l)}{\partial \phi} \tag{4.16}$$

and the gradient with respect to θ is computed as:

$$\frac{\partial \mathcal{L}(x)}{\partial \theta} = \mathbb{E}_{q(l|x)}[(\log p(x|l) - \log q(l|x)) \times \frac{\partial \log q(l|x)}{\partial \theta}] \tag{4.17}$$

Both gradients involve expectations $\mathbb{E}_{q(l|x)}$ which we estimate with beam-searched logical forms l. They are from the distribution $q(l|x)$ computed by the semantic parser.

4.3.3 Scheduled Training

The addition of the inverse parser for removing spurious logical forms, the entire semantic parsing system consists of three components: a parser that generates a logical forms from an utterance, a ranker that measures the likelihood of the logical form executing to the *correct result*, and an inverse parser that scores how the logical forms are *meaning-preserving* (i.e., by reconstruction likelihood). We propose to a scheduled training scheme to balance the two objectives.

Stage 1: At the beginning of training when all model parameters are far from optimal, we train only the parser and the ranker: the parser generates a list of candidate logical forms, from which we find consistent ones and update both the parser and the ranker.

Stage 2: We turn on the inverse parser and update all three components in one epoch. However, the reconstruction loss is only used to update the inverse parser and we prevent it from back-propagating to the semantic parser. The reason is that at this point, the parameters of the inverse parser are sub-optimal and we cannot obtain an accurate approximation of the reconstruction loss.

Stage 3: We allow the reconstruction loss to back-propagate to the parser, and all three components are updated normally. At this point, the two training objectives are both enabled completely: maximizing the likelihood of consistent logical forms and maximizing the reconstruction likelihood.

4.3.4 Neural Lexicon Encoding

In this section we propose to further improve the parsing performance by encoding a lexicon. A lexicon, which covers mappings from knowledge base relations and entities to natural language phrases, is widely used in conventional chart-based parsers (Berant et al., 2013b; Reddy et al., 2014). The lexicon is either hard-coded or learned Krishnamurthy (2016). For example, from a lexicon we know that *the president of* is mapped to the predicate `president_of`. We show here how a lexicon can be used to benefit a neural semantic parser.

The central idea is that each relation or entity can be considered as a single-node tree-structured logical form. We can therefore pretrain the semantic parser (and the inverse parser) with these basic logical form-natural language phrase pairs. The synthetic data act as important prior knowledge to initialize the distribution $q(y|x)$ and $p(x|y)$. With pre-trained word embeddings capturing linguistic regularities on the natural language side, we also expect the approach to help the neural model generalize to unseen natural language phrases quickly. We will experimentally validate the effectiveness of this approach.

4.3.5 Experiments

In this section we present a series of experiments conducted to evaluate the augmented weakly-supervised neural semantic parser. We evaluated our model on three Freebase datasets used earlier: WEBQUESTIONS, GRAPHQUESTIONS and SPADES.

We follow exactly the same setup used in Section 4.1.3. Across training regimes, the dimensions of word vector, logical form token vector, and LSTM hidden states (for the semantic parser and the inverse parser) are 50, 50, and 150, respectively. Word embeddings were initialized with Glove embeddings (Pennington et al., 2014). All other embeddings were randomly initialized. We used one LSTM layer in forward and backward directions. Dropout was used before the softmax activation. The dropout rate was set to 0.5. Finally, momentum SGD (Sutskever et al., 2013) was used as the optimization method to update the parameters of the model. Entity linking has been performed following the same procedures described previously.

We list a few variants of our model to study the impact of each component or method in a controlled experimental setup. We primarily consider the vanilla neural parser-ranker system (see Section 4.1, denoted by TNSP). The parser is trained to maximize the likelihood of consistent logical forms. We then compare it with the

Models	F1
Berant et al. (2013a)	35.7
Berant and Liang (2014)	39.9
Berant and Liang (2015)	49.7
Reddy et al. (2016)	50.3
Yao and Van Durme (2014)	33.0
Bast and Haussmann (2015)	49.4
Bordes et al. (2014)	39.2
Dong et al. (2015)	40.8
Yih et al. (2015)	52.5
Xu et al. (2016)	53.3
TNSP	50.1
+ GRANKER	50.2
+ lexicon encoding on GRANKER	51.7
+ lexicon encoding on parser and GRANKER	52.5

Table 4.5: WEBQUESTIONS results for the augmented TNSP.

system augmented with a generative ranker (denoted by GRANKER), introducing a second objective to maximize the reconstruction likelihood. Finally, we study the impact of neural lexicon encoding when it is used for the generative ranker, and also when it is used for the entire system.

Experimental results on WEBQUESTIONS are shown in Table 4.5. We compare the performance of the TNSP with previous work, including conventional chart-based semantic parsing models (e.g., Berant et al. (2013a)), information extraction models (e.g., Yao and Van Durme (2014)), and more recent neural question-answering models (e.g., Dong et al. (2015)). The neural models do not generate logical forms but instead build a differentiable network to solve the question-answering task.

As the results reveal, the vanilla TNSP outperforms the majority of previous work, especially those with chart-based semantic parsers. The results suggest that neural networks are powerful tools of generating candidate logical forms in the weakly-supervised setting, due to its ability of encoding and utilizing sentential context and generation history. Comparing among various TNSP variants, the addition of a inverse parser only results in marginal gains. However, with the neural lexicon encoding

Models	F1
SEMPRE (Berant et al., 2013a)	10.80
PARASEMPRE (Berant and Liang, 2014)	12.79
JACANA (Yao and Van Durme, 2014)	5.08
UDEPLAMBDA (Reddy et al., 2017)	17.70
TNSP	17.30
+ GRANKER	17.38
+ lexicon encoding on GRANKER	17.67
+ lexicon encoding on parser and GRANKER	18.22

Table 4.6: GRAPHQUESTIONS results for the augmented TNSP.

Models	F1
Unsupervised CCG (Bisk et al., 2016)	24.8
Semi-supervised CCG (Bisk et al., 2016)	28.4
Supervised CCG (Bisk et al., 2016)	30.9
Rule-based system (Bisk et al., 2016)	31.4
Memory networks (Das et al., 2017)	39.9
TNSP	32.4
+ GRANKER	33.1
+ lexicon encoding on GRANKER	35.5
+ lexicon encoding on parser and GRANKER	37.6

Table 4.7: SPADES results for the augmented TNSP.

applied to the inverse parser, we are able to improve the results considerably. We hypothesize the reason is because the inverse parser adopts an unsupervised training objective, which could benefit a lot from prior domain-specific knowledge used to initialize its parameters. When neural lexicon encoding is applied to the semantic parser as well, the system performance can be further improved. In fact, the only work we cannot beat is that of Xu et al. (2016), who use external Wikipedia resources to prune out erroneous candidate answers.

Tables 4.6 and 4.7 present our results on the GRAPHQUESTIONS and SPADES datasets, respectively. Comparison systems for GRAPHQUESTIONS include two chart-based semantic parsers (Berant et al., 2013a; Berant and Liang, 2014), an information

Utterance: *which baseball teams were coached by dave eiland* Logical forms preference before and after augmentation: baseball.batting_statistics.player:baseball.batting_statistics.team(ent.m.0c0x6v) baseball.historical_coaching_tenure.baseball_coach:baseball.historical_coaching_tenure.baseball_team(ent.m.0c0x6v)
Utterance: *who are coca-cola's endorsers* Logical forms for comparison: food.nutrition_fact.food:food.nutrition_fact.nutrient(ent.m.01yvs) business.product_endorsement.product:business.product_endorsement.endorser(ent.m.01yvs)
Utterance: *what are the aircraft models that are comparable to airbus 380* Logical forms for comparison: aviation.aviation_incident_aircraft_relationship.flight_destination:aviation.aviation_incident_aircraft_relatio -nship.aircraft_model(ent.m.0qn2v) aviation.comparable_aircraft_relationship(ent.m.018rl2)

Table 4.8: Comparison between logical forms preferred by the TNSP before and after augmentation. Spurious logical forms (red color) receive higher scores than the semantically-correct counterparts (blue color). The scores of these spurious logical forms become lower than the counterparts in the augmented TNSP.

extraction model (Yao and Van Durme, 2014), and finally a model based on universal dependency to logical form conversion (Reddy et al., 2017). For SPADES, we compare with the method of Bisk et al. (2016) which parses an utterance into a syntactic representation and then grounded to Freebase; and also Das et al. (2017) who employ memory networks and external text resources. On both datasets, we observe similar trends as in the WEBQUESTIONS dataset. The best performing TNSP variant outperforms almost all previous work.

One claim we made about the extended TNSP is that it reduces the impact of spurious logical forms during training. Table 4.8 highlights examples of spurious logical form compared to more semantically-correct counterparts. These spurious logical forms are assigned higher scores in the vanilla TNSP, but lower scores in the extended TNSP.

4.3.6 Discussion

The vanilla neural parser-ranker introduced in Section 4.1 is optimized with consistent logical forms which lead to correct denotations. Although it achieved competitive results compared to chart-based parsers, training of the model can be misled by spurious logical forms. The introduction of the inverse parser aims to alleviate the problem by scoring how a logical form reflects the utterance semantics. Although the inverse parser is not directly used to rank logical forms at test time, the training objective it adopts encourages the parser to generate meaning-preserving logical forms with higher probability. These probabilities are used as features in the log-linear ranker, and therefore the inverse parser implicitly affects ranking results. However, it should be noted that the unsupervised training objective is relatively difficult to optimize, since there are no constraints to regularize the latent logical forms. This motivates us to propose a schedule training regime. We see from the results that when trained properly, the inverse parser and the unsupervised objective indeed bring gains. Moreover, the neural lexicon encoding method we applied essentially produces synthetic data to further regularize the latent space. With pretrained word embeddings capturing phrase similarities, the method also helps the parser to generalize to unseen phrases quickly. For example, by encoding the mapping between the natural language phrase *locate in* and the Freebase predicate `fb:location.location.containedby`, the parser can potentially link a new phrase *located at* to the same predicate.

4.4 Summary

In this chapter we presented a weakly-supervised neural semantic parsing framework. The framework primarily consists of a neural parser and a log linear ranker. The neural parser is essentially the model presented in Chapter 3 and the only difference here is that it generates a list of candidate logical forms, instead of the most likely logical form. These candidates are subsequently ranked by the log linear model, which can leverage global features defined over utterance-logical form pairs. Experiments on question answering datasets reveal the effectiveness of the framework, compared to conventional weakly-supervised semantic parsers.

To further enhance the scalability of the approach, we move on to distant supervision, where utterance-denotation pairs are artificially generated from entity-recognized declarative sentences and a knowledge base. The approach shows promise in improving a practical

question-answering system, but also reveals a performance bottleneck. This is possibly due to the distributional mismatch between artificial data and real questions asked by humans. Future work could focus on better question generation modules to reduce the mismatch.

Finally, we tackled a particular challenge within weakly supervised semantic parsing—the search of candidate logical forms which involve a certain degree of spuriousness. Our solution is based on the fact that correct logical forms should have higher likelihood of reconstructing their utterances, compared to spurious logical forms which coincidentally execute to the correct denotation. We introduces another ranking component (i.e., an inverse parser) to the system for scoring the reconstruction likelihood. Furthermore, we propose a schedule training regime that balances the impact of the new ranking component against the old one. We also introduced a neural lexicon encoding approach to inject prior domain-specific knowledge into neural parameters. We adopted a controlled experimental setup and presented improved results on the question answering datasets.

Compared to the fully-supervised method described in Chapter 3, the weakly-supervised setting is more scalable since training data is easier to collect. As a trade-off, training and optimization under weak signals become more challenging. In the next chapter, we look at a more practical side of neural semantic parsing and attempt to directly reduce the annotation burden of quickly eliciting utterance-logical pairs. This enables us to quickly engineer a neural semantic parser without challenges in training and optimization.

Chapter 5

Building a Neural Semantic Parser from a Domain Ontology

In the previous chapter, we explored a weakly-supervised approach which allows us to train the neural semantic parser with utterance-denotation pairs. While this setting can potentially scale neural semantic parsing to large and open domains, it introduces new challenges regarding training and optimization, since logical forms are latent and incur a large search space. In this chapter, we investigate another approach which directly reduces the data collection burden for utterance-logical form pairs. It offers a more practical solution to quickly building a neural semantic parser for close domains.

As mentioned earlier, the primary challenge of labeling an utterance with logical form is that it requires expert knowledge of the underlying semantic formalism. Annotators need to be careful enough to ensure the logical form is syntactically and semantically valid. Moreover, a general challenge is collecting semantic parsing data in any format is to ensure the utterances have a broad coverage. These utterances are usually human generated, and could have a limited coverage for domains with a large or dynamic ontology.

To overcome these challenges, we build on the work of Wang et al. (2015) an interface for collecting utterance-logical forms, which sidesteps the difficulty of creating utterances and annotating logical forms. The approach inversely starts from the logical form space, which computers can understand and explore. We use a grammar to generate logical forms paired with formal meaning descriptions, which human can understand and computers can process. These formal descriptions are in the form of natural language templates, where each template corresponds to a decomposed fragment of the logical form. The templates can be instantiated and combined to

describe complex human intentions. After the templates are generated, we ask annotators to summarize them into natural utterances, thereby obtaining utterance-logical form pairs. The method is discussed in Section 5.1.

A modeling advantage of the proposed data collection method is that it caches the sequence of grammar rules applied to derive a logical form, in the form of templates. This enables us to extend the neural semantic parser described in Chapter 3 to handle arbitrary non-recursive meaning representations: the parser is now fully-supervised and trained to predict the derivation tree (i.e., rules applied in an order) of the logical form, which has recursive structures. We explain the modeling details in Section 5.2.

We adopt our data elicitation framework to collect a dataset of 6 domains consisting of 7,708 utterance-logical form pairs and test our semantic parser on it (Section 5.3). We provide discussions of our approach in Section 5.4.

5.1 Data Collection Method

5.1.1 Overview

Our data collection method builds on Wang et al. (2015) who advocate crowd-sourcing as a means of mitigating the paucity of training data for new domains. The basic idea is to use a synchronous grammar to generate logical forms paired with *artificial utterances* which crowdworkers are asked to paraphrase. Their method sidesteps the difficulty of annotating logical forms directly, while being able to obtain semantic parsing data with a broad coverage. For example, the logical form `argmin(food_type(Thai food), distance)` is deterministically mapped to "*restaurants with minimum distance with food type thai*", which will be later paraphrased by crowdworkers into more naturally sounding English (e.g., "*nearest restaurants serving thai food*"). Since paraphrasing is much easier than annotating logical forms, the method allows us to obtain semantic parsing data accurately and efficiently. However, the readability of these artificial utterances decreases when the complexity of the querying task and its logical form increases. Take the following artificial sentence as an example: "*find the restaurants that serve food type of kfc which has minimum price*". The artificial sentence is mapped from a logical form with compositional depth 3 and it is rather difficult to interpret due to the ambiguity caused by propositional attachment: it is unclear if *"has minimum price"* is used to modify *kfc*, *food* or *restaurants*. Therefore, the approach primarily targets at utterances exhibiting shallow compositionality often with two predicates and

entities (Wang et al., 2015).

Our first contribution is to extend the approach of Wang et al. (2015) to handle more complicated querying tasks which involve complex human intentions. Instead of representing a logical form with a single artificial sentence, our key insight is to represent it as a sequence of templates where each template corresponds to a fragment of the logical form. So the utterance `argmin(food_type(Thai food), distance)` would be represented with the templates "*Result*$_1$ = *find the [restaurants] where [food type] is [thai]*" and "*Result*$_2$ = *find [Result*$_1$*] with smallest [distance]*". Iyyer et al. (2017) show that when expressing a complex querying task, users often employ a set of inter-related short sentences; such decomposed utterances can potentially handle higher levels of compositionality. Since templates correspond to specific aspects of the logical form, they are easier to understand by crowdworkers than a more elaborate auto-generated utterance representing the entire logical form. Thus the data collection task is formulated as a task of summarizing templates into natural utterances.

Our method supports two modes of data elicitation, striking a balance between efficiency and flexibility (see Figure 5.1). In the *static* mode, our framework first generates a logical form in the backend, and then looks up and fills in natural language templates corresponding to the rules used to derive the logical form. Annotators are then shown these filled templates and are asked to write down a natural language utterance that summarizes the querying task. In the *dynamic* mode annotators are given more flexibility: they can select the templates, fill them, and produce a valid utterance. This mode is designed for domain-specific labelers to use as an annotation tool. It gives them freedom to create utterances that can be anticipated in a given domain.

5.1.2 Logical Form Components

Our approach follows Wang et al. (2015) in that it decomposes a logical form into various constructs. The decomposition allows us to builds a program which generates meaning representations with broad coverage; and explicitly model the generation process during parsing. Throughout this chapter, we exemplify our approach with a database querying task, similar to the previous example (e.g., "*nearest restaurants serving thai food*"). All meaning representations are constructed with lambda calculus representing rules and variables in a computer program that queries a database.

The first construct of logical forms is a class of domain-general rules which stems from the semantic formalism (e.g., lambda calculus) used by the semantic parser.

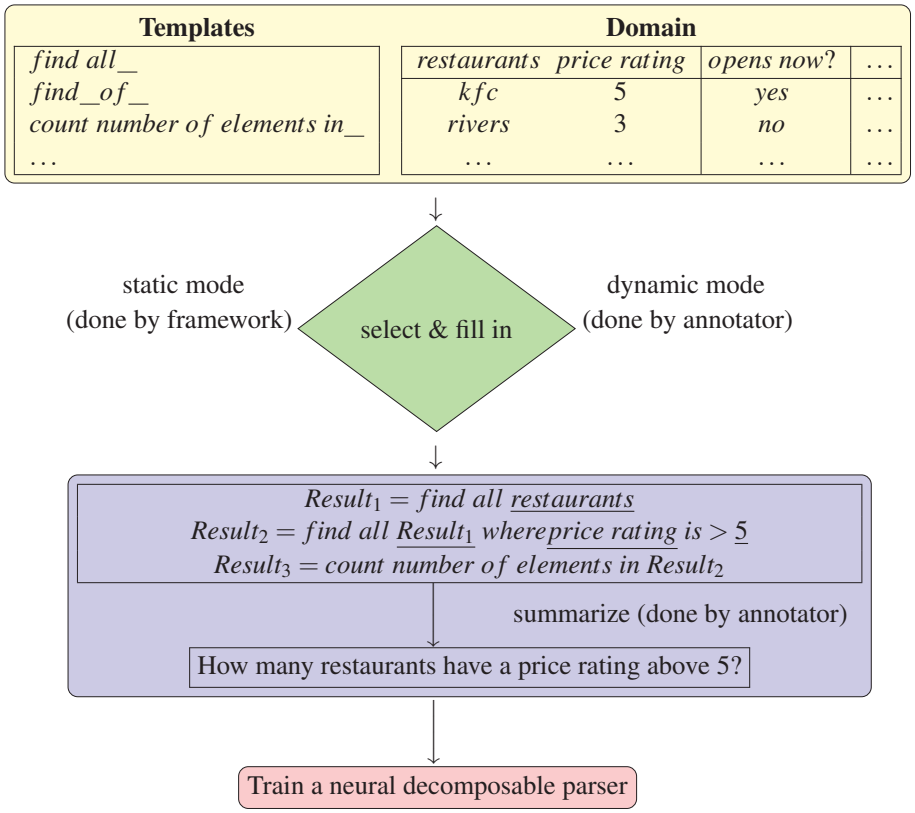


Figure 5.1: Building a single-turn semantic parser from scratch on new domains.

Table 5.1 shows the domain-general rules represented with lambda expressions in the language. They specify various functionalities such as looking up a column in the database, counting, aggregation, and filtering by condition. These rules are generic across domains, in the scope of querying a database.

The second construct of logical forms is the class of domain-specific rules which generate domain-specific predicates or entities. Table 5.2 shows a list of predicates and entities represented as variables in the language. They are examples from a restaurant domain, which covers binary predicates for properties (e.g., `custom_rating`), unary predicates for assertions (e.g., `open_now`), and entities (e.g., `restaurant.kfc`). These aspects pertain to a domain-specific ontology.

5.1.3 Annotation Modes

The central idea of the data elicitation framework is to map each meaning representation to an artificial description, which is incrementally accomplished by mapping derivation rules of the meaning representation to natural language.

Category	Domain-general rules	Description and evaluation
`LookupKey`	`λs:(lookupKey (var s))`	Looks for the entire set of *s*
`LookupValue`	`λpλs:(lookupValue (var s) (var p))`	Looks for specific property *p* of entity *s*
`Filter(property)`	`λsλpλv:(filter (var s) (var p) = (var v))`	Looks for subset of *s* whose property *p* equates to some value *v*
`Filter(assertion)`	`λsλp:(filter (var s) (var p) = true)`	Looks for subset of *s* which satisfies condition *p*
`Count`	`λs:(size (var s))`	Computes the total number of elements in the set *s*
`Sum`	`λs:(sum (var s))`	Computes the total sum of numeric elements in the set *s*
`Comparative` (<)	`λsλpλv:(filter (var s) (var p) < (var v))`	Looks for subset of *s* whose numeric property *p* is smaller than some numeric value *v*
`Comparative` (≤)	`λsλpλv:(filter (var s) (var p) ≤ (var v))`	Looks for subset of *s* whose numeric property *p* is smaller than or equal to some numeric value *v*
`Comparative` (>)	`λsλpλv:(filter (var s) (var p) > (var v))`	Looks for subset of *s* whose numeric property *p* is larger than some numeric value *v*
`Comparative` (≥)	`λsλpλv:(filter (var s) (var p) ≥ (var v))`	Looks for subset of *s* whose numeric property *p* is larger than or equal to some numeric value *v*
`CountComparative` (<)	`λsλpλv:((var s) (size (var p)) < (var v))`	Looks for subset of *s* where the cardinality of property *p* is smaller than some numeric value *v*
`CountComparative` (≤)	`λsλpλv:((var s) (size (var p)) ≤ (var v))`	Looks for subset of *s* where cardinality of property *p* is smaller than or equal to some numeric value *v*
`CountComparative` (>)	`λsλpλv:((var s) (size (var p)) > (var v))`	Looks for subset of *s* where cardinality of property *p* is larger than some numeric value *v*
`CountComparative` (≥)	`λsλpλv:((var s) (size (var p)) ≥ (var v))`	Looks for subset of *s* where cardinality of property *p* is larger than or equal to some numeric value *v*
`Superlative` (min)	`λsλp:((var s) argmin (var p))`	Looks for subset of *s* whose numeric property *p* is the smallest
`Superlative` (max)	`λsλp:((var s) argmax (var p))`	Looks for subset of *s* whose numeric property *p* is the largest
`CountSuperlative` (min)	`λsλp:((var s) argmin (size (var p)))`	Looks for subset of *s* where cardinality of property *p* is the smallest
`CountSuperlative` (max)	`λsλp:((var s) argmax (size (var p)))`	Looks for subset of *s* where cardinality of property *p* is the largest

Table 5.1: Domain-general rules and their descriptions used to define meaning representations in our experiments

For domain-general rules, our framework maps them into natural language templates with missing entries. Each template specifies the functionality of a rule, while missing entries specify the variables which the rule expects. Table 5.3 displays the full collection of templates for the domain-general rules we used to query a database. These templates are described in such a way that can be understood by annotators who have no knowledge of the logical language. Different from more natural descriptions shown in Table 5.1, templates are precise formal descriptions in a fixed format which is human readable and also computer processable.

Domain-specific variables are mapped to natural language phrases with a lexicon specified by the domain manager. This lexicon is the only resource we ask domain

Category	Domain-specific predicates and entities	Description
BinaryPredicate	custom_rating	Overall rating from customers
	price_rating	Price rating from customers
	distance	Distance of the restaurant
	num_reviews	Number of reviews from customers
	location	Location of the restaurant
	cuisine	Type of food served by the restaurant
	open_time	Open time of the restaurant
UnaryPredicate	open_now	Is the restaurant is open now?
	take_away	Does the restaurant offer take-away?
	reservation	Does the restaurant accept reservations?
	credit_card	Does the restaurant accept credit cards for payment?
	waiter	Does the restaurant have waiter service?
	delivery	Does the restaurant offer delivery?
	kids	Is the restaurant suitable for kids?
	groups	Is the restaurant suitable for groups?
Entity	restaurant.kfc	KFC
	location.oxford_street	Oxford Street

Table 5.2: Domain-specific predicates and entities from a restaurant domain, covering binary predicates (properties), unary predicates (assertions) and entities.

managers to provide, for the purpose of describing the domain ontology. Note that natural language descriptions are important in cases where domain-specific predicates or entities are not verbalized (for example a predicate may be simply represented as an index m.001 in the database). Such descriptions must be provided to annotators to allow for basic understanding, to enable the paraphrasing task. However, we do not use the lexicon for building a neural semantic parser in this chapter. Table 5.4 displays a lexicon for the restaurant domain. The natural language descriptions of predicates or entities are used to instantiate templates.

As shown in Figure 5.1 our framework supports two modes of data collection. Examples of the two modes are shown in Tables 5.5 and 5.6, respectively.

Static Mode Once a domain manager specifies the necessary domain-specific information (e.g., restaurant names and properties), our framework obtains utterance-logical form pairs with the following method:

1. We use a context-free grammar to generate logical forms by sampling domain-

Category	Natural language templates
`LookupKey`	*find all of $s*
`LookupValue`	*find $p of $s*
`Filter(property)`	*find $s where $p is $v*
`Filter(assertion)`	*find $s which satisfies $p*
`Count`	*count number of elements in $s*
`Sum`	*sum all elements in $s*
`Comparative(<)`	*find $s with $p < $v*
`Comparative(>)`	*find $s with $p > $v*
`Comparative(≤)`	*find $s with $p ≤ $v*
`Comparative(≥)`	*find $s with $p ≥ $v*
`CountComparative(<)`	*find $s with number of $p < $v*
`CountComparative(>)`	*find $s with number of $p > $v*
`CountComparative(≤)`	*find $s with number of $p ≤ $v*
`CountComparative(≥)`	*find $s with number of $p ≥ $v*
`Superlative(min)`	*find $s with smallest $p*
`Superlative(max)`	*find $s with largest $p*
`CountSuperlative(min)`	*find $s with smallest number of $p*
`CountSuperlative(max)`	*find $s with largest number of $p*

Table 5.3: Domain-general rules are associated with natural language templates specified by our framework.

Predicates/entities from database	Natural language phrase
custom_rating	customer rating
price_rating	price rating
distance	distance
num_reviews	number of customer reviews
location	location
cuisine	cuisine
open_time	opening time
open_now	opens now
take_away	offers take away
reservation	takes reservation
credit_card	accepts credit card
waiter	has waiter service
delivery	offers delivery
kids	suitable for kids
groups	suitable for groups
restaurant.kfc	KFC
location.oxford_street	Oxford Street

Table 5.4: Examples of the lexicon for the restaurant domain.

general and specific rules. The generation is performed in a bottom-up manner, so that larger pieces of logical forms can be constructed from smaller ones. The application of each domain-general rule takes as arguments an expected amount of domain-specific variables or smaller logical forms, and results in a new piece of meaning.

Table 5.5 shows an example of the generation process. We use grammar constrains to ensure that all logical forms constructed bottom-up are always syntactically valid (i.e. variables are type-checked). and semantically correct (i.e., no rules logically entail or contradict with each other). For example, if the previous logical form applies a `Count` rule which returns a number, the next rule cannot be `LookupKey` since the expected argument is a database column instead of a number. If a `Filter` rule is applied to include only restaurants within 500 meters, it does not make sense to use a subsequent `Filter` rule to look for restaurants which are more than 1 kilometers away.

2. For each logical form fragment in the bottom-up derivation, we look up the corresponding template underlying the domain-general rule that is used to construct

it. The template has missing entries, which are instantiated with domain-specific predicates, entities or referent to other templates. In the end, we obtain a sequence of instantiated templates that describe the task underlying the final logical form.

3. The instantiated templates are displayed to crowdworkers, who are asked to summarize the task with an utterance. The utterance does not need to be a single sentence; it can consist of a few sentences or clauses. We set no restriction to the format of expression. As a result, we collect utterances paired with logical forms.

Dynamic Mode The dynamic mode offers annotators the flexibility to create logical forms on their own. We envisage the primary users of our framework in this mode being professional domain annotators who have a better understanding of (or can devote more time to) the task. Query-logical form pairs are collected as follows:

1. The framework displays all available templates and a table containing domain information to the annotator. The table includes natural language descriptions of all available predicates. However, the list of entities need not be exhaustive, since it is straightforward to apply entity replacement (for entities of the same type) to construct more utterance-logical forms from a set of base utterance-logical forms.

2. The annotator manually selects and fills in a sequence of templates to come up with a querying task. In this process, logical rules are applied recursively at the back-end to construct the final logical form.

3. The annotator writes down a natural summary of the querying task and obtains an utterance-logical form pair.

The two modes of annotation allow to trade efficiency with flexibility. The static mode only requires annotators to summarize a collection of mature templates to create datasets quickly. The dynamic mode is an annotation tool which offers annotators the flexibility to generate their own querying tasks.

5.2 Decomposable Neural Semantic Parsing

Another advantage of our data collection method stems from the fact that it caches the sequence of logical rules used to obtain a logical form; each rule corresponds to

1. Bottom-up construction of meaning representations (done by framework)

The first piece of meaning representation is constructed with the domain-general rule:

`λs:(lookupKey (var s))`

and the domain-specific variable:

`type.restaurant`

By applying the domain-general rule to the variable, we get the first piece of meaning representation

$\texttt{Result}_1$`=(lookupKey (type.restaurant))`

The second piece of meaning representation is constructed with the domain-general rule:

`λsλpλv:(filter (var s) (var p) =` `(var v))`

and the domain-specific and coreferential variables:

$\texttt{Result}_1$, `rel.cuisine`, `cuisine.thai`

By applying the domain-general rule to the variables, we get the second piece of meaning representation

$\texttt{Result}_2$`=(filter (`$\texttt{Result}_1$`) (rel.cuisine) = (cuisine.thai))`

The third piece of meaning representation is constructed with the domain-general rule:

`λs:(lookupValue (var s) (var p))`

and the domain-specific variables:

`restaurant.kfc`, `rel.distance`

By applying the domain-general rule to the variables, we get the third piece of meaning representation

$\texttt{Result}_3$`=(lookupValue (restaurant.kfc) (rel.distance))`

The final piece of meaning representation is constructed with the domain-general rule:

`λsλpλv:(filter (var s) (var p) <` `(var v))`

and the domain-specific variables:

$\texttt{Result}_2$, `rel.distance`, $\texttt{Result}_3$

By applying the domain-general rule to the variables, we get the final piece of meaning representation

$\texttt{Result}_4$`=(filter (`$\texttt{Result}_2$`) (rel.distance) <` `(`$\texttt{Result}_3$`))`

2. Each piece of meaning representation is converted to a canonical representation described by templates. Templates associated with domain-general rules are instantiated with with domain-specific and coreferential variables (shown in brackets):

$Result_1$ = *find all* [*restaurants*]

$Result_2$ = *find* [$Result_1$] *where* [*cuisine*] *is* [*Thai*]

$Result_3$ = *find* [*distance*] of [*KFC*]

$Result_4$ = *find* [$Result_2$] *with* [*distance*] < [Result_3]

3. The templates are displayed to the annotator, whose job is to summarize them into a pre-defined utterance:

Which restaurant has Thai food and is closer to me than KFC?

Table 5.5: An example of the static mode data collection process for a single-turn utterance paired with meaning representations.

a decomposable fragment of the logical form. This enables us to extend the neural semantic parser described in Chapter 3 to handle both recursive and non-recursive meaning representations: the parser is trained fully-supervised to predict the derivation[1] of the logical form, which is recursive and tree(or graph)-structured. See Figure 5.2

[1] A derivation refers to the sequence of logical rules applied to obtain a logical form.

1. Instructions, templates (unfilled) from Table 5.3 and domain information (see Table 5.2) are displayed to the annotator.
2. The annotator manually selects and fills in a few templates to come up with a querying task: *Result*$_1$ = *find all* [*restaurants*] *Result*$_2$ = *find* [*Result*$_1$] *where* [*cuisine*] *is* [*Thai*] *Result*$_3$ = *find* [*distance*] of [*KFC*] *Result*$_4$ = *find* [*Result*$_2$] *with* [*distance*] $<$ [Result$_3$]
3. While templates are inputted, bottom-up construction of logical forms are performed at the back-end: The first piece of meaning representation is constructed with the domain-general rule: λs:(lookupKey (var s)) and the domain-specific variable: type.restaurant By applying the domain-general rule to the variable, we get the first piece of meaning representation Result$_1$=(lookupKey (type.restaurant)) The second piece of meaning representation is constructed with the domain-general rule: λsλpλv:(filter (var s) (var p) = (var v)) and the domain-specific and coreferential variables: Result$_1$, rel.cuisine, cuisine.thai By applying the domain-general rule to the variables, we get the second piece of meaning representation Result$_2$=(filter (Result$_1$) (rel.cuisine) = (cuisine.thai)) The third piece of meaning representation is constructed with the domain-general rule: λs:(lookupValue (var s) (var p)) and the domain-specific variables: restaurant.kfc, rel.distance By applying the domain-general rule to the variables, we get the third piece of meaning representation Result$_3$=(lookupValue (restaurant.kfc) (rel.distance)) The final piece of meaning representation is constructed with the domain-general rule: λsλpλv:(filter (var s) (var p) < (var v)) and the domain-specific variables: Result$_2$, rel.distance, Result$_3$ By applying the domain-general rule to the variables, we get the final piece of meaning representation Result$_4$=(filter (Result$_2$) (rel.distance) < (Result$_3$))
4. The annotator writes down an utterance that summarizes the querying task they have come up with: *Which restaurant has Thai food and is closer to me than KFC?*

Table 5.6: An example of the dynamic mode data collection process for a single-turn utterance paired with meaning representations.

for an example of the derivation tree.

Overview The decomposable neural semantic parser for derivation trees has a same architecture as the parser present in Chapter 3.

Similar to Chapter 3, we encode the input utterance with a bidirectional LSTM and generate the derivation tree of the logical form with a stack-LSTM. In the generation process, a sequence of logical rules is predicted following a canonical order. Two specific modeling choices we adopt in the chapter is top-down, pre-order generation and soft attention-based prediction due to their superior performance in Chapter 3. Other variants of the semantic parser presented in Chapter 3 can be applied too.

We reply on the `reduce` mechanism of stack-LSTM to build larger logical forms from smaller ones. `reduce` essentially shows when a subtree generation is completed. When `reduce` is called, we apply a beta reduction to the domain general rule represented by the root of the subtree. In this way, the final logical form can be built recursively. As an example of the restaurant domain, the space of all logical rules consists of the general rules in Table 5.1 and the domain-specific ones in Table 5.2. For simplicity, we denote each rule with a category name as shown in the tables. For domain-specific rules, we first predict their general category (binary predicate, unary predicate, or entity) and then the specific predicate or entity choice (with different neural network parameters). The resulting space of rule predictions is [`LookupKey`, `LookupValue`, `Filter`, `Count`, `Sum`, `Comparative`, `CountComparative`, `Superlative`, `CountSuperlative`, `BinaryPredicate`, `UnaryPredicate`, `Entity`]. The derivation tree for the logical form (rendered in red) in Table 5.5 is shown in Figure 5.2. This structure is similar to a FunQL logical form. For completion of the presentation, we provide details of the neural semantic parser as follows.

Encoder We encode utterance $x = (x_1, \cdots, x_n)$ with a bidirectional LSTM encoder, into a list of token representations $[h_1, \cdots, h_n]$. Each representation is the concatenation of the corresponding forward and backward LSTM states.

Decoder The derivation tree of the logical form is generated with a stack-LSTM decoder simulating a transition sytem. As shown in Figure 5.2, non-terminal nodes of the derivation tree are domain-general rules, while terminal nodes are domain-specific ones. When a non-terminal or terminal rule y_t is newly predicted, the stack-LSTM state, denoted by g_t, is updated from its older state g_{t-1} as an ordinary LSTM:

$$g_t = \text{LSTM}(y_t, g_{t-1}) \tag{5.1}$$

The new state is additionally pushed onto the stack marking whether it corresponds to a non-terminal or terminal.

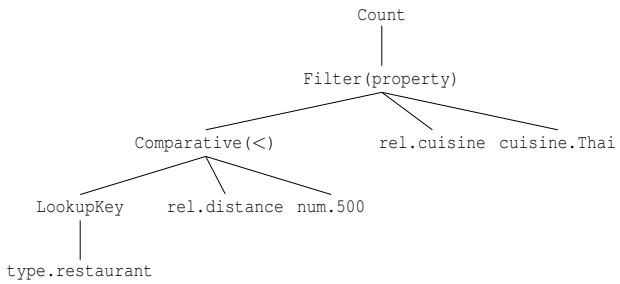


Figure 5.2: Derivation tree for the logical form (shown in red) from Table 5.5. Domain-general rules are represented as non-terminal nodes, using the abbreviations shown in Table 5.1. For example, `Count` refers to λs:`(call size (var s))`. Domain-specific aspects are represent as terminal nodes.

The generation of the tree nodes is performed in top-down, pre-order and the model needs to identify when the prediction of a subtree branch is completed—then a beta reduction step should be applied to the corresponding logical rules. To achieve that, we rely on the built-in "reduce" mechanism of stack-LSTM and incorporate `Reduce` as an additional rule in the rule prediction space: When a subtree branch is "reduced" (or completely predicted), the states of the stack-LSTM are recursively popped from the stack until a non-terminal is encountered. This non-terminal state is popped as well, after which the stack-LSTM reaches an intermediate state denoted by $g_{t-1:t}$. At this point, we compute the representation of the completed subtree z_t as:

$$z_t = W_z \cdot [p_z : c_z] \tag{5.2}$$

where p_z denotes the parent (non-terminal) embedding of the subtree, and c_z denotes the average embedding of the children (terminals or already-completed subtrees). W_z is the weight matrix. Finally, z_t serves as input for updating $g_{t-1:t}$ to g_t:

$$g_t = \text{LSTM}(z_t, g_{t-1:t}) \tag{5.3}$$

Prediction At each time step of the decoding, the parser first predicts a rule conditioned on the decoder state g_t and the encoder states $h_1 \cdots h_n$. We apply standard soft attention between g_t and the encoder states $h_1 \cdots h_n$ to compute a feature representation $\bar{h}_t$:

$$u_t^i = V \tanh(W_h h_i + W_g g_t) \tag{5.4}$$

$$a_t^i = \text{softmax}(u_t^i) \tag{5.5}$$

$$\bar{h}_t = \sum_i a_t^i h_i \tag{5.6}$$

where V, W_h and W_g are all weight parameters. The prediction of the next rule y_{t+1} is computed with a softmax classifier, which takes the concatenated features $\bar{h}_t$ and g_t as input:

$$y_{t+1} \sim \text{softmax}(W_y \tanh(W_f[\bar{h}_t, g_t])) \tag{5.7}$$

Recall that for domain-specific or terminal rules, our parser first predicts their high-level category (binary predicate, unary predicate or entity). Only when y_{t+1} matches one of the terminal categories, we further predict a fine-grained predicate or entity, with another set of neural parameters:

$$y_{t+1} \sim \text{softmax}(W_{y'} \tanh(W_{f'}[\bar{h}_t, g_t])) \tag{5.8}$$

where W_y and $W_{y'}$ are different weight matrices.

Summary The decomposable neural semantic parser presented above generates the derivation tree of a logical form, and builds the logical form recursively. This extends the parser in Chapter 3 to handle both recursive and non-recursive logical forms, as long as derivations exist. Model-wise, the decomposable neural semantic parser is a specific variant of the parser in Chapter 3: the tree is generated in top-down and pre-order; and soft attention is used for all predictions. Other model variants (e.g., bottom-up generation, hard and structured attention) described in Chapter 3 can be transfered here to generate derivation trees, too.

5.3 Experiments

In this section, we first describe how we used our data elicitation framework to obtain annotations for six domains in the static mode. We then discuss experiments on this dataset with the semantic parser just described. We finally compare our framework to Wang et al. (2015), and provide evaluation of the dynamic mode.

5.3.1 Data Collection in Static Mode

Our static-mode data collection focused on six domains, two of which relate to company management (meeting and employees databases), two concern recommendation engines (hotel and restaurant databases), and two target healthcare applications (disease and medication databases).

#rules	1				2				3				4				All			
Domain	Q	Tp	Tk	WO	Q	Tp	Tk	WO	Q	Tp	Tk	WO	Q	Tp	Tk	WO	Q	Tp	Tk	WO
meeting	378	191	9.1	1.41	570	537	16.20	2.21	254	254	16.2	2.81	46	46	21.37	3.19	1,248	1,028	12.59	2.21
employees	322	240	8.96	1.34	320	319	13.47	2.95	486	486	18.2	3.66	268	268	22.37	3.12	1,396	1,313	15.79	3.12
hotel	170	146	8.99	1.91	358	542	16.86	3.64	542	542	16.86	4.11	433	433	19.89	4.05	1,503	1,479	15.87	4.05
restaurant	132	98	8.14	1.40	301	295	12.74	3.41	495	495	16.74	3.74	311	311	20.02	3.66	1,239	1,199	15.68	3.66
disease	283	212	9.3	1.29	301	455	14.23	3.52	455	455	19.49	5.65	213	213	23.95	4.48	1,252	1,176	16.68	4.48
medication	136	102	8.53	1.31	252	246	11.89	2.35	435	435	16.09	3.17	247	247	20.04	2.91	1,070	1,030	15.05	2.91

Table 5.7: Number of utterances (Q), number of templates (Tp), average number of tokens (Tk) and token overlap between utterances and templates (WO) per domain and overall (All).

The dataset was collected with Amazon Mechanical Turk (AMT). Figure A.1 shows the instructions and web interface of the static mode data collection. Across domains, the total number of querying tasks (described by templates) that the framework generated was 7,225. These tasks were sampled randomly (without replacement) to show to crowdworkers. Each worker saw three tasks per HIT and was paid 0.3$. After removing repeated utterance-logical form pairs, we collected a semantic parsing dataset of 7,708 examples. The average amount of time annotators spent on each domain was three hours. We evaluated the correctness of 100 randomly chosen utterances, and the accuracy was 81%. We did not conduct any manual post-processing as our aim was to simulate a real-world scenario that handles noisy data.

Table 5.7 shows the number of utterances (Q) we obtained for each domain broken down according to the depth of compositionality. The table also provides statistics on the number of templates (Tp) per domain, the average number of tokens (Tk) per utterance in each domain, and the token overlap (Ov) between the utterances and the corresponding templates. Overall, we observe that the dataset reveals a high degree of compositionality across domains. The number of utterances collected at each compositional level is dependent on the space of predicates in each domain. We also see that utterance length does not vary drastically among domains even though the average number of tokens is affected by the verbosity of entities and predicates in each domain. We use word overlap as a measure of the amount of paraphrasing. We see that utterances in the disease domain deviate least from their corresponding templates while utterances in the meeting domain deviate most. This number is affected by the degree of expert knowledge required for paraphrasing in each domain.

Table 5.8 presents examples of the utterances we elicited for the six domains

together with their corresponding templates.

Domain	Templates	Query
meeting	R_1 = *find* [*location*] *of* [*annual review*]	*what location will the annual review take place*
meeting	R_1 = *find* [*location*] *of* [*annual review*] R_2 = *find all* [*meetings*] R_3 = *find* [R_2] *where* [*location*] is not [R_1] R_4 = *find* [R_3] *with smallest number of* [*attendee*]	*which meeting is not held in the same venue as annual review, and attracts the least amount of attendance*
employees	R_1 = *find all* [*employees*] R_2 = *find* [R_1] *with smallest number of* [*projects*]	*which employee is assigned minimum projects*
employees	R_1 = *find all* [*employees*] R_2 = *find* [R_1] *with* [*salary*] $<$ [5000] R_3 = *find* [R_1] *with* [*salary*] $>$ [15000] R_4 = *find* [R_2 *or* R_3] *where* [*division*] *is* [IT]	*for those employees in the IT division, who are paid less than 5000 or more than 15000*
hotel	R_1 = *find all* [*hotels*] R_2 = *find* [R_1] *where* [*distance*] is [*300*] R_3 = *find* [R_2] *which satisfies* [*free cancellation*]	*which hotel is at 300 metres and doesn't charge a cancellation fee*
hotel	R_1 = *find all* [*hotels*] R_2 = *find* [R_1] *where* [*location*] *is* [*oxford street*] R_3 = *find* [R_2] *where* [*room type*] *is* [*single* or *double*] R_4 = *find* [R_3] *which satisfies* [*has free wifi*]	*which hotel in oxford has single or double room? the hotel should has free wifi too*
restaurant	R_1 = *find all* [*restaurants*] R_2 = *find* [R_1] *with* [*number of reviews*] $>$ [*100*] R_3 = *count elements in* [R_2]	*how many restaurants have more than 100 reviews*
restaurant	R_1 = *find all* [*restaurants*] R_2 = *find* [R_1] *with* [*number of reviews*] $>$ [*500*] R_3 = *find* [R_2] *with largest number of* [*cuisine*] R_4 = *find* [R_3] *which satisfies* [*has outdoor seatings*]	*for the restaurants with more than 500 reviews, look for those with the largest variety of food, and then those with seats outside*
disease	R_1 = *find all* [*diseases*] R_2 = *find* [R_1] *with smallest* [*incubation period*] R_3 = *find* [R_2] *where* [*symptom*] *is not* [*bleeding*] R_4 = *count elements in* [R_3]	*how many diseases having the smallest incubation period don't result in bleeding*
disease	R_1 = *find* [*symptom*] *of* [*fever*] R_2 = *find all* [*diseases*] R_3 = *find* [R_2] *where* [*symptom*] *is* [R_1] R_4 = *find* [R_3] *with largest* [*incubation period*]	*which disease has the same symptom as fever, and has the longest incubation period*
medication	R_1 = *find all* [*medications*] R_2 = *find* [R_1] *which satisfies* [*for adult only*] R_3 = *find* [R_2] *where* [*target symptom*] *is* [*bleeding*] R_4 = *find* [R_3] *where* [*side effect*] *is* [*headache*]	*what adult medications treat bleeding in exchange for a headache*
medication	R_1 = *find all* [*medications*] R_2 = *find* [R_1] *where* [*target symptom*] *is* [*headache*] R_3 = *find* [R_2] *where* [*category*] *is* [*physician or pharmacist*] R_4 = *find* [R_3] *which satisfies* [*requires prescription*]	*find the medicine for headache, with a category of physician or pharmacist; and the medicine requires prescription*

Table 5.8: Examples of templates (filled values shown within brackets), and elicited utterances across six domains. R is a shorthand for $Result$.

5.3.2 Semantic Parsing Results

In the process of building the above dataset, our framework cached the derivations of logical forms (rules or templates used in sequence). This allowed us to train a neural semantic parser that generates the logical form by following its derivation.

In our experiments, all LSTMs had one layer with 150 dimensions. The word embedding size and the rule embedding size were set to 50. A dropout of 0.5 was used on the input features of the softmax classifiers. Momentum SGD (Sutskever et al., 2013) was used to update the parameters of the model. We implemented two baselines, the first is the sequence-to-sequence (S2S) parser (Dong and Lapata, 2016; Jia and Liang, 2016), while the second is the vanilla sequence-to-tree (S2T) parser we presented in Chapter 3. Note that the vanilla sequence-to-tree parser generates peripheral tree-structured outputs with transition actions (`NT`, `TER` and `RED`). This generation process is interpretable for recursive logical forms whose peripheral tree structures reveal corresponding derivations, However, it is non-interpretable for non-recursive logical forms whose peripheral tree structures do not reveal how the logical forms are obtained.

Model	meeting		employees		hotel		restaurant		disease		medication	
	ExM	SeM	ExM	SeM	ExM	SeM	ExM	SeM	ExM	SeM	ExM	SeM
S2S	37.2	43.2	14.3	17.5	24.5	31.2	21.3	29.0	16.7	23.2	15.5	16.7
S2T	41.2	46.8	21.4	28.2	31.5	43.5	25.4	35.9	22.4	34.4	28.2	33.9
S2D	45.6	54.0	27.8	35.5	39.2	52.6	47.2	49.5	26.9	44.6	35.8	46.2

Table 5.9: Performance on various domains (test set) using exact match (ExM) and semantic match (SeM).

Table 5.9 shows results on the test set of each domain using exact match as the evaluation metric (ExM). Our sequence-to-derivation tree model (S2D) yields substantial gains over the baselines (S2S and S2T) across domains. However, a limitation of exact match is that different logical forms may be equivalent to the commutativity and associativity of rule applications (Xu et al., 2017). For example, two subsequent `Filter` rules in a logical form are interchangeable. For this reason, we additionally compute the number of logical forms that match the gold standard at the denotation level (see SeM in Table 5.9). Again, we find that the sequence to derivation tree model outperforms related baselines by a wide margin.

We conducted further experiments by training a single model on data from all domains, and testing it on the test set of each individual domain. As the results in Table 5.10 reveal, we obtain gains for most domains on both metrics of exact and semantic match. Our results agree with previous work (Herzig and Berant, 2017) which improves semantic parsing accuracy by training a single sequence to sequence model over multiple knowledge bases. Since domain-general aspects are shared, when training across domains, the parser receives more supervision cues on discovering these domain-general aspects from their natural language descriptions.

Domain	ExM	SeM
meeting	(45.6) 48.8	(54.0) 56.8
employees	(27.8) 31.7	(35.5) 41.4
hotel	(39.2) 41.8	(52.6) 56.1
restaurant	(47.2) 33.1	(49.5) 48.8
disease	(26.9) 30.7	(44.6) 48.3
medication	(35.8) 37.4	(46.2) 51.8

Table 5.10: Sequence-to-derivation tree model (S2D) trained on *six* domains and evaluated on the test set of each domain. Results of S2D when trained and tested on a *single* domain are shown in parens.

5.3.3 Comparison to Wang et al. (2015)

Our static mode is similar to Wang et al. (2015) in that we also ask crowdworkers to paraphrase artificial expressions into natural ones. In their approach crowdworkers paraphrase a *single* artificial sentence, whereas in our case they are asked to paraphrase a *sequence* of templates. We directly evaluated how crowdworkers perceive our templates compared to single sentences. For each domain we randomly sampled 24 querying tasks described by templates (144 tasks in total) and derived the corresponding artificial language using Wang et al. (2015)'s grammar. Artificial sentences and templates were also paired with a natural language description (generated by us) which explained the task (see Table 5.13 for examples). Workers were asked to rate how well the sentence and templates corresponded to the natural language description according to two criteria: (a) intelligibility (how easy is the artificial language to understand?) and (b) accuracy (does it match the intention of the task?). Participants used a 1–5 rating

scale where 1 is worst and 5 is best. We elicited 5 responses per task. Figure A.2 shows the instructions and web interface of the comparative study.

Domain	Intelligibility		Accuracy		Combined	
	S	T	S	T	S	T
meeting	3.54	3.81	3.86	4.02	3.70	3.91
employee	3.58	4.13	3.48	4.43	3.53	4.28
hotel	3.81	3.96	3.79	4.29	4.12	3.80
restaurant	3.93	4.23	3.80	3.75	3.86	4.13
disease	3.41	3.69	3.68	4.02	3.55	3.86
medication	3.83	3.97	3.83	4.40	3.83	4.19
All	3.96	3.67	3.55	4.03	3.70	4.06

Table 5.11: Comparison between artificial sentences (S; Wang et al., 2015) and template-based approach (T). Mean ratings are shown per domain and overall. Combined is the average of Intelligibility and Accuracy. Means are underlined if their difference is statistically significant at $p < 0.05$ using a post-hoc Turk test.

Table 5.11 summarizes the mean ratings for each domain and overall. As can be seen, our approach generally receives higher ratings for Intelligibility and Accuracy. When both types of ratings are combined, the templates significantly outperform the individual sentences for all domains but meetings. Table 5.12 shows a breakdown of our results according to the depth of compositionality. Queries of depth 1 and 2 can be easily described by one sentence, and our template-based approach has no clear advantage over Wang et al. (2015). However, when the compositional depth increases to 3 and 4, templates are perceived as more intelligible and accurate across domains; all means differences for depths 3 and 4 are statistically significant ($p < 0.01$). Further qualitative analysis suggests that our approach receives higher ratings in cases where the output of the grammar from Wang et al. (2015) involves various propositional attachment ambiguities. The ambiguities are common when the compositional depth increases (Example 1 in Table 5.13), when the utterance contains conjunction and disjunction (Example 2 in Table 5.13), and when a sub-utterance acts as object of comparison in a longer utterance (Example 3 in Table 5.13).

Depth	Intelligibility		Accuracy		Combined	
	S	T	S	T	S	T
1	4.00	4.29	4.05	4.29	4.03	4.26
2	4.11	4.18	4.03	4.18	4.07	4.02
3	3.61	4.09	3.60	4.01	3.58	4.01
4	3.56	4.28	3.35	4.25	3.60	4.10

Table 5.12: Comparison between artificial sentences (S; Wang et al., 2015) and template-based approach (T) for varying compositionality depths. Mean ratings are aggregated across domains. Combined is the average of Intelligibility and Accuracy. Means are underlined if their difference is statistically significant at $p < 0.01$ using a post-hoc Turk test.

5.3.4 Data Collection in Dynamic Mode

Next, we provide evaluation of our framework in the dynamic mode which we view as an annotation tool for domain managers. Because it is not realistic to recruit a large number of domain managers to participate in our evaluation, we ran this experiment on AMT with crowdworkers who were paid more to compensate for the increased workload. For each domain, workers were given unfilled templates and a table describing naturalized predicates of that domain. They were also given instructions and examples explaining how to use them. Workers were then asked to chose the templates, fill them, and write down an utterance summarizing the task. We recruited 50 workers for each domain (300 in total) and each was asked to complete two tasks in one HIT worth 1$. Figure A.3 shows the instructions and web interface of the dynamic mode data collection.

Table 5.14 shows the statistics of the data we obtained. Workers tend to use more than 3 templates (on average) per task, and the degree of paraphrasing is increased compared to the static mode which suggests that the dynamic mode provides more flexibility for annotators to create data. Moreover, across domains, 90% of the logical forms generated by AMT workers are executable. Although we envisage domain managers as the main users of the dynamic mode, the result indicates that (with proper instructions and examples) naive AMT workers can generate meaningful logical forms to some extent. Inspection of the output failures revealed two common reasons. Firstly, annotators filled in templates with wrong types. For example, one worker filled the

Task description	Our templates	Wang et al. (2015)
We have a database of diseases and would like to find diseases which have fever as their symptom. These diseases should be treatable with antibiotics. Their incubation period is longer than a day. If you have such a disease you should see a doctor.	R_1 = *find the diseases whose symptom is fever* R_2= *find* R_1 *whose treatment is antibiotics* R_3= *find* R_2 *whose incubation period is longer than a day* *QR*= *find* R_3 *which require to see a doctor*	diseases whose symptom is fever whose treatment is aspirin whose incubation period is larger than a day which require to see a doctor
We have a database of diseases and would like to find diseases which have fever as their symptom; amongst them, we would like to find those with heart disease as complication. Finally, we want to find all diseases that can be treated with antibiotics.	R_1= *find the diseases whose symptom is fever* R_2= *find the diseases whose complication is heart disease* *QR*= *find* R_1 *and* R_2 *whose treatment is antibiotics*	disease whose symptom is fever and disease whose complication is heart disease whose treatment is antibiotics
We have a database of diseases. We would like to first find the incubation period of fever; and then find the diseases which have incubation period longer than fever; these diseases can be also treated with antibiotics.	R_1= *find incubation period of fever* R_2= *find diseases whose incubation period is larger than* R_1 *QR*= *find* R_2 *whose treatment is antibiotics*	diseases whose incubation period is larger than incubation period of fever whose treatment is antibiotics

Table 5.13: Templates and artificial sentences shown to AMT crowdworkers together with task description. Examples are taken from the disease domain. *R* and *QR* are shorthands for *Result* and *Query Result*, respectively.

`LookupKey` template with an entity (e.g., *find all* [*annual review*]) instead of a database key; and another worker used `Count` as the first template followed by `Filter`, however type constraints require `Filter` to take a set of entities as argument instead of a number (as returned by `Count`). Secondly, annotators ignored information in the table and created tasks using non-existing predicates. Since the logical forms can be tested for their validity automatically, providing feedback on the fly will yield a higher percentage of executable utterances.

Domain	Tk	WO	Tp	Acc
meeting	16.71	2.83	3.46	0.925
employees	18.64	3.71	3.33	0.938
hotel	16.97	4.08	3.37	0.969
restaurant	17.33	3.72	3.36	0.943
disease	18.95	5.12	3.12	0.925
medication	17.25	3.26	3.15	0.989

Table 5.14: Average number of tokens (Tk), token overlap between utterances and templates (WO), average number of templates (Tp) and proportion of parsable templates (Acc) per domain on dynamic mode.

5.4 Summary

In this chapter we provide an end-to-end solution for building a practical neural semantic parser for closed domains. We developed a template-based data collection framework for utterance-logical form pairs, which starts from logical form space and formulates the annotation task as a summarization problem. The framework supports two modes (static vs. dynamic) that strike a balance between efficiency and flexibility. Compared to the previous work of Wang et al. (2015), our approach can potentially handle more complex querying tasks. Quantitative and qualitative analysis is provided to support this argument.

The data collection framework fits seamlessly with the neural semantic parser described in Chapter 3. More importantly, it enables us to extend the parser to handle non-recursive logical forms, by generating their derivation trees. The final logical forms can be recursively obtained by composing rules in the derivation. With the data collection framework and the neural semantic parser, one can quickly build a system to parse domain-specific utterances, involving complex human intentions, and starting with only a domain ontology. A limitation of the approach is that it does not support sequential input utterances, which are also common for expressing complex human intentions. We extend our framework in handling such utterances in the next chapter.

Chapter 6

Neural Semantic Parsing for Sequential Utterances

The bulk of existing work, including our neural semantic parser presented so far, has focused on single-turn utterances, ignoring the fact that most natural language interfaces receive inputs in streams. With complex intentions, users typically ask questions or perform tasks in multiple steps, and decompose a complex utterance into a sequence of inter-related sub-utterances (Iyyer et al., 2017). For instance, when searching for a restaurant, a user may first ask "*which restaurants serve thai food*" followed by "*which ones are near me*". Even in cases where users have a well-defined utterance in mind, it is not uncommon to ask follow-on questions, in an attempt to refine their search or because they wish to compare different results (Moe and Fader, 2001; Asri et al., 2017). For example, a request following the utterance "*which restaurants serve thai food*" could be "*of those in oxford street*" and "*which ones are in bond street*". A feature of these utterances is that they exhibit context dependencies. For example, both pronouns *those* and *ones* refer to *the resultants serve thai food*.

In this chapter, we extend the data collection-modeling framework described in Chapter 5 to handle sequential utterances, which simulate a user session. We consider a non-interactive setting which the system can present denotations to the user but cannot generate optimum responses for user interaction. We formulate the sequential semantic parsing task as parsing each utterance correctly and resolving co-reference between utterances. The task is challenging due to the many different ways natural language expresses the same information within the same utterance and across utterances. For instance, a long utterance can be paraphrased by two short ones; and a single utterance can exhibit arbitrary degrees of compositionality. A second challenge concerns the

availability of training data.

To this end, we analyze the types of co-reference patterns commonly attested in sequential utterances (Section 6.1), and extend the data collection method described in Chapter 5 to crowdsource sequential utterance-logical form pairs which represent these co-reference patterns. A context-free grammar generates co-referring logical forms paired with sequential, artificial templates, which are then paraphrased by crowd workers to obtain natural language utterances (Section 6.2). On the modeling side, we decompose semantic parsing for sequential utterances into the subtasks of single utterance parsing and co-reference prediction, each of which is accomplished by a neural network. The model is described in Section 6.3.

We perform experiments on restaurant and hotel domains, and elicit 15,000 sessions of utterances for each domain. We show that our neural model is able to accurately parse sequential utterances. The overall approach allows to quickly build a neural semantic parser for sequential utterances starting from a domain ontology.

6.1 Patterns of Sequential Utterances

Our first task is to gather sequential semantic parsing data that facilitates modeling. Sequential utterances can potentially exhibit a rich class of co-reference phenomenon, either explicit or implicit. However, co-reference in meaning representations is always explicit with fixed format. We start by analyzing the co-reference patterns in meaning representations and then proceed to collect examples reflecting these patterns.

We use an ad-hoc, inductive approach which enumerates co-reference patterns in three consecutive meaning representations as the base cases. Complex co-reference patterns can be constructed from the base cases. When the interpretation of meaning representation M2 (with corresponding utterance Q2 and template R2) depends on meaning representation M1 (with corresponding utterance Q1 and template R1), we call M2 (and Q2, R2) the consequent, and M1 (and Q1, R1) the antecedent. Coreference patterns in three consecutive queries are shown in Figure 6.1 and described in more details below.

Exploitation refers to chain-structured co-reference, where the consequent of the previous meaning representation is the antecedent of the next utterance. The user exploits a querying task by incrementally adding constrains. As a result, the next utterance consistently uses the result of the previous one.

Exploration refers to branch-structured co-reference, where one antecedent has

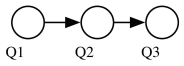
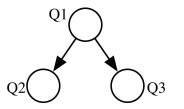
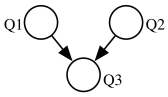
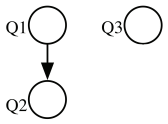
	Exploitation	
	Q1: *restaurants in oxford street?*	R_1 = *find restaurants where location is oxford street*
	Q2: *which cost less than 50?*	R_2 = *find* $[R_1]$ *where price* < 50
	Q3: *with car parking?*	R_3 = *find* $[R_2]$*which has car parking*
	Exploration	
	Q1: *restaurants in oxford street*	$Result_1$ = *find restaurants where location is oxford street*
	Q2: *with chinese food?*	R_2 = *find* $[R_1]$ *where food type is chinese*
	Q3: *oxford street restaurants with thai food?*	R_3 = *find* $[R_1]$ *where food type is thai*
	Merging	
	Q1: *find chinese restaurants.*	R_1 = *find restaurants where food type is chinese*
	Q2 *find thai restaurants.*	R_2 = *find restaurants where food type is thai*
	Q3 *chinese or thai restaurants with car parking?*	R_2 = *find* $[R_1]$ *or* $[R_2]$ *which has car parking*
	Unrelated	
	Q1: *which restaurants serve chinese food?*	R_1 = *find restaurants where food type is chinese*
	Q2: *those in oxford street?*	R_2 = *find* $[R_1]$ *where location is oxford street*
	Q3: *which restaurants serve thai food?*	R_3 = *find restaurants where food type is thai*

Figure 6.1: Coreference patterns in three consecutive meaning representations or corresponding utterances. They are represented as nodes; edges are drawn between co-referring utterances; antecedents are nodes with outgoing edges while consequents are nodes with incoming edges.

two consequents. The user explores different options or constrains related to the same antecedent utterance.

Merging is the inverse of exploration, where a consequent utterance has two antecedents. The user combines two or more previous utterance results with union or intersection. And the combined result is then used in a subsequent utterance.

Unrelated refers to two unconnected co-reference structures observed in a session. The user specifies two different querying tasks in the same turn which are not mutually co-referent.

We expect the four categories mentioned above to cover the majority of co-reference patterns in sequential meaning representations. However, there exist meanings which they fail to represent or construct. An example is the following sequential utterances: "*find restaurants in oxford street*", "*find restaurants in bond street*", "*which of them serve chinese food*", and finally "*how about those in oxford street*". Note that the scope

of this chapter is to handle the "head" co-reference patterns of sequential meaning representations, but not solving the long tails. The chapter does not aim to provide a formalization of co-reference patterns either.

6.2 Data Collection

We extend the data collection method described in Chapter 5 to collect sequential semantic parsing data which exhibits the above co-reference patterns. Following the previous chapter, we assume the grammar consists of domain-general rules which specify functionalities to query a database (Table 5.1). Each domain-general rule is associated with a template description. Besides, the grammar includes domain-specific rules to create domain-specific predicates or entities. We expect that a lexicon is specified to map each predicate or entity to a natural language description (Table 5.2).

In the case of sequential semantic parsing, an additional ingredient of the grammar includes rules generating co-reference placeholders, which establish an anaphoric link between two pieces of logical forms (antecedents and consequents). To model co-reference, we adopt the notion of discourse referents (DRs) and discourse entities (DEs) from the Discourse Representation Theory (Webber, 1978; Kamp and Reyle, 2013). DRs are referential expressions appearing in utterances which denote DEs, which are mental entities in the speaker's model of discourse. The co-referential placeholders imply the DEs in the antecedent and consequent refer to the same real-world entity: the entity (or entities) obtained from the execution of the antecedent. In corresponding utterances, the co-reference is justified by explicit or implicit occurrence of a pronoun or the DR (e.g., a definite noun phrase) of the consequent. The main principle in determining whether DRs co-refer is that it must be possible to infer their relation from the dialog context alone, without using world knowledge. An example of co-reference indicated by an explicit pronoun is: *which restaurant serve thai food* followed by *of those which are nearest to me*; while for implicit pronoun it is: *which restaurant serve thai food* followed by *nearest to me*; and the co-reference example with a definite noun phrase is given by: *which restaurant serve thai food* followed by *thai restaurants nearest to me*. The co-referential placeholders in meaning representations indicate any of the above co-reference phenomenon happening at the corresponding utterances. In this work, we consider three types of co-reference placeholders. as shown in Table 6.1. They can be used in place of (type-agreed) domain-specific entities to construct logical forms.

ID	Description
`Coref`	Refers to a single antecedent
`Union_coref`	Refers to union of two antecedents
`Intersection_coref`	Refers to intersection of two antecedents

Table 6.1: Co-reference placeholders.

Same as in the single-turn case, at the beginning of sequential data collection, we request a domain manager (e.g., the person maintaining a restaurant database) to provide domain-specific details such as restaurant names and properties, and the lexicon. After that, our method obtains sequential semantic parsing data in the following steps (also exemplified in Table 6.2):

- As in the single-turn case, we use a context-free grammar to generate logical forms by sampling domain-general and specific rules. The generation is performed in a bottom-up manner, so that larger pieces of logical forms can be constructed from smaller ones. The application of each domain-general rule takes an expected amount of domain-specific or co-referential variables, and results in a new piece of meaning. Different from the single-turn case, we allow each meaning representation to be constructed with one to three domain-general rules, so as to have some basic level of compositionality. Besides, we consider a rich class of co-reference patterns in between logical forms.

  We introduce three rules which generate different co-referential variables shown in Table 6.1. For each co-referential variable, we additionally generate its value by sampling from the list of previously generated logical forms. As mentioned earlier, an index has been assigned to each logical form (e.g., $\texttt{Result}_1$, $\texttt{Result}_2$) and this index is used to denote the value of a co-referential variable.

  Similar to the single-turn case, we use grammar constrains to ensure that all logical forms constructed bottom-up are always syntactically valid (i.e. variables are type-checked). and semantically correct (i.e., no rules logically entail or contradict with each other). For example, if the previous logical form applies a `Count` rule which returns a number, the next rule cannot be `LookupKey` since the expected argument is a database column instead of a number. If a `Filter` rule is applied to include only restaurants within 500 meters, it does not make sense to use a subsequent `Filter` rule to look for restaurants which are more than 1

1. Bottom-up construction of meaning representations (done by framework)

The first piece of meaning representation is constructed with two domain-general rules:

λs:(lookupKey (var s))

λsλpλv:(filter (var s) (var p) = (var v)) with corresponding domain-specific variables:

type.restaurant rel.cuisine cuisine.thai

By applying the domain-general rule to the variable, we get:

Result_1=(filter (lookupKey (type.restaurant)) (rel.cuisine) = (cuisine.thai))

The second piece of meaning representation is constructed with the domain-general rule:

λs:(lookupValue (var s) (var p))

and the domain-specific variables:

restaurant.kfc, rel.distance

By applying the domain-general rule to the variables, we get the third piece of meaning representation:

Result_2=(lookupValue (restaurant.kfc) (rel.distance))

The third piece of meaning representation is constructed with the domain-general rule:

λsλpλv:((var s) min (var p))

and the domain-specific variables:

Result_1, rel.price

By applying the domain-general rule to the variables, we get the final piece of meaning representation:

Result_3=((Result_1) argmin (rel.price))

The final piece of meaning representation is constructed with the domain-general rule:

λsλpλv:(filter (var s) (var p) < (var v))

and the domain-specific variables:

Result_3, rel.distance, Result_2

By applying the domain-general rule to the variables, we get the final piece of meaning representation:

Result_4=(filter (Result_3) (rel.distance) < (Result_2))

2. Each piece of meaning representation is converted to a canonical representation described by templates. Templates associated with domain-general rules are instantiated with with domain-specific and co-referential variables (shown in brackets):

Result_1 = find [*restaurants*] *where* [*cuisine*] *is* [*Thai*]

Result_2 = find [*distance*] of [*KFC*]

Result_3 = find [*Result_1 with smallest*] [*price rating*]

Result_4 = find [*Result_3*] *with* [*distance*] < [Result_2]

3. The templates are displayed to annotators, whose job is to summarize them into a pre-defined utterance:

Show me Thai restaurants.

How far is KFC?

Cheapest Thai food?

Which are closer than KFC?

Table 6.2: An example of the data collection process for sequential utterances.

kilometers away.

In the sequential case, we use additional constrains to guarantee the co-reference relations are valid: our method caches the n most recently generated logical forms, their consequents and antecedents, and their return types (e.g., the return type of a `Filter` is an entity set, while for `Count` it is a number). For the next logical form, the grammar can sample terminal-level rules which generate co-referential variables and their values, following one of the cases below:

1. the co-referential variable is `Coref`, and its value is one of the previously generated logical forms whose denotation is type checked and has no consequents. This results in the Exploitation pattern.
2. the co-referential variable is `Coref`, and its value is one of the previously generated logical forms whose denotation is type checked and has other consequents. This results in the Exploration pattern.
3. the co-referential variable is `Union_coref` or `Intersection_coref`, and its value is constructed from two previous logical forms whose denotations are type checked. Moreover, the two logical forms should not exhibit an Exploitation pattern. This results in the Merging pattern.
4. there is no co-referential variable in the meaning representation. This results in the Unrelated pattern.

- For each piece of logical form in the sequence, we retrieve the domain-general rule it uses, and look up the corresponding template. The template has missing entries, which are instantiated with domain-specific predicates, entities or co-referential index.

  Since each logical form may be constructed with more than one domain-general rules, there may be a sequence of templates associated with the logical form. We combine their templates by merging their constrains. For example, two templates "$Result_1$ = *find* [*restaurants*] *with* [*distance*] $<$ [*500m*]", $Result_2$ = *find* [$Result_1$] *with* [*price*] $>$ [*50$*] can be merged into "$Result_1$ = *find* [*restaurants*] *with* [*distance*] $<$ [*500m*] and [*price*] $>$ [*50$*]". This ensures that every logical form has exactly one canonical template representation which will be paraphrased by humans.

- We display the sequence of instantiated templates to crowdworkers. Different from the single-turn case where crowdworkers summarize *all* the templates into a

single utterance, in the sequential case we ask them to paraphrase *every* template into an utterance, while expressing the discourse structure of the whole task (by using implicit or explicit co-reference). This results in a sequence of inter-related utterances paired with logical forms.

6.3 Non-context Dependent Parsing for Sequential Utterances

On the modeling side, we decompose the sequential neural semantic parsing task into two steps: generating underspecified logical forms (e.g., `(filter (coref) (distance) < (500))`) and determining which utterances are co-referent (e.g., the value of `coref` is $\texttt{Result}_1$).

Generating Underspecified Logical Forms In the first step we parse every utterance in the session into an underspecified logical form containing co-reference placeholders (see Table 6.1). To accomplish the task we adopt the same decomposable neural semantic parser introduced in Section 5.2, which generates the derivation tree of each logical form. The only difference is that the terminal rules of the derivation tree additionally include co-reference placeholders. In comparison to Section 5.2, the prediction space of terminal rules is [`BinaryPredicate`, `UnaryPredicate`, `Entity`, `Coref`, `Union_coref`, `Intersection_coref`].

Recall that our neural semantic parser consists of a bidirectional LSTM encoder for utterance encoding and a stack-LSTM decoder that generates the derivation tree of the output logical form. During each step of the generation process, the model first selects a terminal or non-terminal rule, or the reduce rule which implies a beta reduction. If a predicate or entity rule (i.e., `BinaryPredicate`, `UnaryPredicate` or `Entity`) is selected, the model further predicts a specific predicate or entity choice. Similarly, if a co-reference rule (i.e., `Coref`, `Union_coref` or `Intersection_coref`) is selected, we further predict the detailed co-reference choice. This is accomplished with an intra-attention network described below.

Co-reference Resolution We rely on an intra-attention network (Cheng et al., 2016; Parikh et al., 2016) to predict the co-reference relations among utterances: given the current utterance with the corresponding underspecified logical form (which contains

co-reference placeholders), the network predicts which of the previous utterances it co-refers with. Note that the number of predictions is determined by the category of the placeholder. For the placeholder `Coref`, we need to select one of the previous utterances; while for `Union_coref` and `Intersection_coref`, we need two previous utterances.

We adopt a bidirectional LSTM architecture to encode each utterance similar to the neural semantic parser described above. Specifically, utterance x of length n is encoded with a bidirectional LSTM into a list of forward $[\overrightarrow{h_1}, \cdots, \overrightarrow{h_n}]$ and backward representations $[\overleftarrow{h_1}, \cdots, \overleftarrow{h_n}]$. We use the concatenation of $\overrightarrow{h_n}$ and $\overleftarrow{h_1}$ as the overall utterance representation, denoted by X.

Given utterance x, the previous bidirectional LSTM encoder obtains a list of forward representations $[\overrightarrow{h_1}, \cdots, \overrightarrow{h_n}]$, and backward representations $[\overleftarrow{h_n}, \cdots, \overleftarrow{h_1}]$. We use the concatenation of $\overrightarrow{h_n}$ and $\overleftarrow{h_1}$ as the utterance representation X, which is fed into the intra-attention network. This is different from the encoder of the neural semantic parser, which maintains a list of token representations instead of a single utterance representation.

Given the current utterance embedding X_c and previous utterance embeddings $X_1, \cdots, X_{c-1}$, the intra-attention network computes the relation distribution between X_c and every previous utterance in the list of $[X_1, \cdots, X_{c-1}]$. This is accomplished by first computing a score u_c^i for each of the previous utterances in the list, with index i ranging from 1 to $c-1$:

$$u_c^i = W_v \tanh(W_i X_i + W_X X_c) \tag{6.1}$$

Then, a softmax classifier is used to yield the relational distribution:

$$a_c^i = \text{softmax}(u_c^i) \tag{6.2}$$

where the softmax is taken over all the previous utterances in the list; a_c^i denotes the probability that the current utterance X_c co-refers with the ith utterance X_i. Ws are weight parameters. We then select the most probable utterance to replace `Coref`, or the top two most probable utterances to construct `Union_coref` or `Intersection_coref`, thus formulating the complete meaning representation.

6.4 Context Dependent Parsing for Sequential Utterances

One should note that when parsing sequential (context-dependent) utterances, our semantic parser presented above does not make an explicit use of context: the generation of a co-referential placeholder (in an underspecified logical form) replies only on the given utterance, although the placeholder's value is predicted within the context. In the following, we extend the semantic parser to make an explicit use of context when parsing sequential utterances.

The central idea is to maintain a context vector for the current user session. This vector is used as an additional feature when generating underspecified logical forms, in the softmax classifier (Equation 3.29) that predicts transition operations (which include those generate co-referential placeholders). To compute the context vector, we use the same intra-attention network described in Equation 6.1. Given the current utterance x_c with representation X_c, the network computes a score u_c^i for each of the previous utterances x_i, with representation X_i and the index i ranging from 1 to $c-1$:

$$u_c^i = W_v \tanh(W_i X_i + W_X X_c) \tag{6.3}$$

We then compute a context representation for the current utterance, denoted by $\bar{X}_c$

$$a_c^i = \text{softmax}(u_c^i) \tag{6.4}$$

$$\bar{X}_c = \sum a_c^i * X_c \tag{6.5}$$

The context representation $\bar{X}_c$ is used at every time step when the decoder generates the derivation tree of x_c, by extending Equation 3.29 as:

$$a_{t+1} \sim \text{softmax}(W_{oa} \tanh(W_f [\bar{b}_t, s_t, \bar{X}_c])) \tag{6.6}$$

We will experimentally verify if modeling context explicitly is useful for parsing sequential utterances.

6.5 Experiments

In this section we conduct experiments to collect sequential semantic parsing data on two domains, and present parsing results.

6.5.1 Data Collection

As explained previously the dataset we collected contains four co-reference patterns (see Figure 6.1), which are generated by a semantic grammar. For a user session, we set the maximum exploitation depth to 5. We also restrict each turn to have at most one pattern of exploration, merging or unrelated. These four patterns can co-exist with exploitation, but not with each other. More complex co-reference patterns (e.g., two explorations in the same turn) are difficult to describe in natural language; they rarely occur in practice and therefore we do not consider them.

We elicited data for two domains, namely restaurants and hotels, using Amazon's Mechanical Turk (AMT). Figure A.4 shows instructions and interfaces for the data collection task. For each domain and each pattern we generated 3,000 querying tasks (described by templates). These tasks were sampled randomly (without replacement) to show to crowdworkers. Each worker saw two tasks per HIT and was paid 0.2$. The average amount of time annotators spent on each domain was three hours. Finally, we obtained 12,000 examples for each domain. Table 6.3 shows basic statistics of the obtained data. These include input and output vocabulary size, the number of utterances per turn, the number of tokens per utterance, and the word overlap between the utterance and its corresponding template. As can be seen, the data exhibits a significant amount of paraphrasing (more than half of the tokens in each template are paraphrased).

	Hotels	Restaurants
Input (utterance) vocabulary size	3,813	4,628
Output (logical form) vocabulary size	386	386
#utterances/turn	3.98	4.01
#tokens/utterance	9.30	9.11
#word overlap/utterance	4.22	4.07

Table 6.3: Statistics of the sequential utterance-logical forms.

We evaluated the correctness of 100 randomly sampled turns of utterances (covering 531 utterances in total). Amongst these, 79 turns were correct and the remaining 21 contain wrongly paraphrased utterances. The paraphrasing accuracy for all 531 utterances was 94.4%. Perhaps unsurprisingly, we found that all mistakes workers made relate to co-reference. The most common mistake is to ignore co-reference during paraphrasing.

For example, instead of asking "*which ones of these restaurants have thai food*", a worker may simply write "*which restaurants have thai food*" which ignores previous constraints.

Table 6.4 and 6.5 presents examples of the sequential utterances we elicited for the hotel and restaurant domains together with their co-reference patterns and corresponding templates.

Domain	Pattern	Templates	Queries
hotel	Exploitation	R_1 = *find* [*hotels*] *with* [*price rating*] $\leq$ [$$] R_2 = *find* [R_1] *with number of* [*room type*] $\geq$ [*4*] R_3 = *find* [R_2] *which satisfies* [*near the sea*] R_3 = *find* [R_2] *which satisfies* [*can be reserved*]	*Which hotels have a price rating of 2 dollar signs or less?* *Can you find which of these have more than 4 different types of rooms?* *Among these, which are located near the sea?* *Among these, which takes reservations?*
hotel	Exploration	R_1 = *find* [*hotels*] *with largest number of* [*customer reviews*] R_2 = *find* [R_1] *with* [*customer rating*] $\geq$ [*5 stars*] R_3 = *find* [R_1] *with* [*customer rating*] $\geq$ [*3 stars*]	*Show me hotels with the largest number of customer reviews* *Which of those hotels have received a customer rating of 5 or more stars?* *Of the hotels with the largest number of reviews, which ones have a customer rating of 3 or more stars?*
hotel	Merging	R_1 = *find* [*hotels*] *with largest number of* [*room type*] R_2 = *find* [R_1] *with* [*customer rating*] $\geq$ [*5 stars*] R_3 = *find* [R_1] *with* [*distance*] $\leq$ [*500m*] R_4 = *find* [R_2 *and* R_3] *which satisfies* [*car parks*]	*Which hotel has the most variety of rooms?* *Which of these are rated at 5 stars or more?* *Which of the previous hotels are within 500 meters to me?* *Of all these hotels with 5 stars or near me, which offers car parks?*
hotel	Unrelated	R_1 = *find* [*hotels*] *with largest number of* [*room type*] R_2 = *find* [R_1] *with smallest* [*price rating*] R_3 = *find* [*hotels*] *which satisfies* [*airport shuttle*] *and* [*private bathroom*]	*What hotels have the largest amount of room types?* *Among those, which have the smallest price rating?* *Which hotels have an airport shuttle and private bathroom?*

Table 6.4: Examples of co-reference patterns, templates (filled values shown within brackets) and elicited utterances for the hotel domain. *R* is a shorthand notation for *Result*.

6.5.2 Semantic Parsing Results

Next, we evaluate our neural semantic parser on the collected sequential utterances. Specifically, each utterance in a session is paired with a meaning representation which can contain co-reference placeholders. The goal of the parser is to parse every utterance (and predict the value of co-referential variables) in the session. We adopt the same experimental setup as in Chapter 5. All LSTMs have one layer with 150 dimensions, and all word and rule embeddings have size 50. A dropout of 0.5 was used on the input

Domain	Pattern	Templates	Queries
restaurant	Exploitation	R_1 = *find* [*restaurants*] *with smallest* [*price rating*] R_2 = *find* [R_1] *with largest number of* [*food type*] R_3 = *find* [R_2] *which satisfies* [*good for groups*] *and* [*has waiter service*] R_4 = *find* [R_3] *with* [*distance*] $\leq$ [*500m*]	*Show me the cheapest restaurants.* *Which of these have the largest variety of food?* *Of these, are there any restaurants that are good for groups and offer waiter service?* *Are any of those restaurants within half a kilometer to me?*
restaurant	Exploration	R_1 = *find* [*restaurants*] *with largest* [*customer rating*] R_2 = *find* [R_1] *with* [*distance*] $\leq$ [*500m*] R_3 = *find* [R_2] *where* [*food type*] *is* [*thai food*] R_4 = *find* [R_2] *where* [*food type*] *is* [*american food*]	*Which restaurants have the largest customer ratings?* *Which of these restaurant is within 500 meters?* *Show me those serves Thai food?* *Of the previous restaurants, find the ones which serves American food instead.*
restaurant	Merging	R_1 = *find* [*restaurants*] *where* [*location*] *is* [*downtown*] R_2 = *find* [R_1] *which satisfies* [*has breakfast*] R_3 = *find* [R_1] *which satisfies* [*has lunch*] R_4 = *find* [R_2 and R_3] *with largest number of* [*customer reviews*]	*Show me restaurants located downtown?* *Which of these restaurants serve breakfast?* *Which restaurants serve lunch?* *Of all these restaurants with breakfast or lunch, which have the most customer reviews?*
restaurant	Unrelated	R_1 = *find* [*restaurants*] *which satisfies* [*can be reserved*] R_2 = *find* [R_1] *which satisfies* [*take credit card*] R_3 = *find* [R_2] *with number of* [*cuisine*] $>$ [1] R_4 = *find*[*restaurants*] *with smallest* [*price rating*] R_5 = *find* [R_4] *with largest* [*customer rating*]	*which restaurants have reservations option?* *which among these restaurants accept credit cards?* *among these restaurants find those with multiple types of cuisine?* *which restaurants have the smallest price rating?* *which among these restaurants have the largest customer rating?*

Table 6.5: Examples of co-reference patterns, templates (filled values shown within brackets) and elicited utterances for the restaurant domain. R is a shorthand notation for $Result$.

features of the softmax classifiers. We used momentum SGD (Sutskever et al., 2013) to update model parameters. Recall that we use an intra-attention network to compute the value of co-referential variables. This network relies on a bidirectional-LSTM to compute sentence vectors. In our experiments, the weights of this bidirectional-LSTM is shared with the encoder of the semantic parser (The encoder maintains a list of token vectors instead of a single sentence vector). We compare our non-context dependent semantic parser (S2D), and context dependent semantic parser (S2D-context) with the sequence-to-sequence baseline (S2S).

For evaluation metrics, we primarily consider the parsing accuracy of each individual utterance (ExM). This accuracy is first averaged within each session and then over sessions. We additionally consider cases when different sequential utterances lead to the same results and measure the match at the semantic or denotation level (SeM). We also categorized the results based on data sessions involving different co-referential

restaurant		Exploitation	Exploration	Merging	Unrelated	All
ExM	S2S	46.9	46.2	38.3	38.3	42.0
	S2D	72.5	67.7	53.1	54.3	60.2
	S2D-context	73.6	68.1	54.6	54.9	62.4
SeM	S2S	23.8	23.2	3.1	3.8	13.1
	S2D	42.5	47.1	5.7	7.0	25.1
	S2D-context	42.8	47.1	5.8	7.1	25.1
hotel		Exploitation	Exploration	Merging	Unrelated	All
ExM	S2S	48.3	48.2	43.6	41.5	44.5
	S2D	73.8	77.8	65.4	57.2	66.9
	S2D-context	73.7	78.0	65.0	59.5	67.0
SeM	S2S	24.6	19.5	6.3	4.1	13.6
	S2D	43.7	56.8	11.8	6.7	29.7
	S2D-context	43.6	56.8	11.8	7.1	29.8

Table 6.6: Sequential semantic parsing results on the restaurant and hotel domains.

patterns: with exploitation only, and exploitation combined with exploration/merging/unrelated.

Table 6.6 shows the semantic parsing results, which clearly reveal the advantage of S2D over S2S. We see that the results for semantic match is rather low, indicating that it is challenging to get all utterances in a session parsed correctly. Comparing among various sessions, those with merging and unrelated result in the lowest accuracy. To better understand how the model performs on exploration, merging and unrelated, we evaluate parsing accuracy (precision, recall and F1) for specific utterances involving these patterns.

First, for the prediction of the exploration pattern, the precision, recall and F1 are 84.4%, 73.6%, 78.6% respectively for the restaurant domain (87.7%, 71.9%, 79.0% for the hotel domain). Second, for the prediction of the merging pattern involving union or intersection, the precision, recall and F1 are 88.6%, 21.4% and 34.5% respectively for the restaurant domain (94.8%, 43.2% and 59.4% for the hotel domain). The low recall indicates that most of the union or intersection symbols in the meaning representations are not correctly discovered. Finally, in the unrelated pattern, we have two independent sets of querying tasks in one turn. We predict how the model does on predicting the first utterance of the second querying task, which indicates the model's ability in detecting goal shift. As a result, the precision, recall and F1 are 81.9%, 63.8% and

71.7% respectively for the restaurant domain (82.8%, 68.1% and 74.7% for the hotel domain). Overall, we see that the parser does fairly well on exploitation, exploration and unrelated, while it fails to recognize most merging patterns.

Next, we investigate if modeling context is helpful for parsing sequential utterances, by comparing the results of non-context dependent and context dependent semantic parsers presented in Table 6.6. We see that across different co-reference patterns, using explicit context modeling during decoding can lead to improvements of the sequential semantic parsing task. However, note that all gains are not large. We think the reason is that a single utterance often provides enough clues to infer an explicit or implicit co-reference variable. Explicit co-reference comes with indicative words such as pronouns (e.g., "*find restaurants in the oxford street?*" followed by "*those with car parks?*"), while in utterances with implicit co-reference the querying object is often missing (e.g., "*find restaurants in the oxford street?*" followed by "*with car parks?*"). It is not difficult for a non-context dependent neural network to learn these indicative patterns of co-reference from training data. An exception is when an utterance contains discourse referents (e.g., a definite noun phrase) for co-reference. The discourse entities denoted by discourse referents refer to the same real world entities as in the antecedent utterance. An example is "*find restaurants in the oxford street?*" followed by "*the oxford restaurants with car parks?*" In this scenario, modeling context helps the parser identify a co-referential relation in the consequent utterance, where "*oxford*" refers to "*oxford street*" and "*the oxford restaurants*" refers to the denotation of the antecedent utterance.

6.6 Summary

When expressing complex intentions, humans may either use a complete utterance at once, or incrementally specify a sequence of inter-related utterances. It is therefore important for neural semantic parsers to handle both types of inputs. In this chapter, we extend the previously proposed semantic parsing framework to collect and parse sequential utterances. The approach starts from the logical form space, and generates inter-related logical forms containing co-reference placeholders. These logical forms are mapped to formal descriptions and paraphrased by annotators to yield natural utterances. With the proposed approach, one can quickly build a sequential semantic parser for closed domains, starting with just a domain ontology.

Our work related to the previous work of Zettlemoyer and Collins (2009), who also

design a two-stage context-dependent semantic parser that constructs logical forms containing co-references. Different from our work, their first stage uses a probabilistic CCG parser to generate incomplete logical forms containing co-reference while the second stage resolves co-reference by making modifications to logical forms that depend on the context. Besides, Artzi and Zettlemoyer (2013b) and Long et al. (2016) both parse context-dependent instructions with chart-based semantic parsers under weak supervision. Similar to our work, Suhr et al. (2018) proposes a context-dependent neural semantic parser which maintains a session-level encoder updated after each turn.

A limitation of our work is that we only simulate the non-interactive scenario where the system does not generate natural language responses. Future work will be deployed to study sequential semantic parsing in a conversational agent (Young et al., 2013).

Chapter 7

Conclusions

7.1 Contributions

In this thesis, we introduced a novel neural semantic parser that converts natural language utterances to logical forms (Chapter 3). The semantic parser has the following novelties. First, it employs a sequence-to-tree model to generate syntactically valid logical forms following a transition system and grammar constrains. The outputs are guaranteed to be executable. Second, the transition system integrates the generation of domain-general and specific aspects in a unified way. Domain-general aspects are language-dependent predicates and structures, while domain-specific aspects are relations and entities defined by a domain ontology. This unification reduces the need of domain-specific engineering and makes the parser applicable across domains. Third, our neural semantic parser uses various neural attention mechanism to handle mismatches between natural language and KB predicates—a central challenge in semantic parsing. The attention mechanism reduces the need of lexicon learning. Experiment on the fully-supervised GEOQUERY dataset has demonstrated the effectiveness of our neural semantic parser.

We extended our neural semantic parser to a weakly-supervised setting within a parser-ranker framework (Chapter 4). Weakly-supervised training data consists of utterance-denotation pairs, which are easier to obtain than utterance-logical form pairs. The framework we proposed combines the merits of conventional and neural semantic parsing. We reply on the powerful neural network to generate candidate logical forms via beam search. These candidates are then ranked based on their likelihood of executing to the correct denotation, and their agreement with the utterance semantics. We present a scheduled training method, and propose to use a neurally encoded lexicon to inject

prior domain knowledge to the model. Experiments on three Freebase datasets demonstrate the effectiveness of our neural semantic parser, achieving results within the state-of-the-art range.

We also delved into a practical side of neural semantic parsing—how does one quickly build a neural semantic parser from a domain ontology (Chapter 5 and 6). Our approach follows and extends the previous work of Wang et al. (2015). Specifically we focused on semantic parsing tasks involving complex human intentions, which can be expressed by a compositional utterance or a sequence of inter-related utterances. We first developed an interface for efficiently collecting compositional utterance-logical form pairs as a summarization task. This is achieved by converting computer-generated utterances to formal descriptions in the form of templates, which are summarized by crowd workers into natural utterances. We then leveraged the data collection method to train a neural semantic parsers which generate derivation trees for logical forms. Next, the end-to-end system was extended to handle sequential utterances, which exhibit contextual dependencies. In general, our approach provides an end-to-end solution for the development of a closed-domain neural semantic parser from scratch.

7.2 Findings

In the research and development of neural semantic parsers, our findings are as follows.

Structure priors and grammar constraints are important in designing executable neural semantic parsers As described in Chapter 3, our neural semantic parser generates logical forms following a transition system and grammar constraints. This ensures the generation is interpretable, and the outputs are always syntactically valid and executable. Experiments in both fully- and weakly-supervised settings have demonstrated the effectiveness of structure priors and grammar constraints.

Soft attention is cost-effective in neural semantic parsing In Chapter 3, we compared soft, hard and structured attention mechanism. All attention aims to learn a mapping between natural language and logical language. It turns out that soft attention is the most cost-effective for downstream semantic parsing tasks. Hard and structured attention renders training more complex but does not always leads to superior performance.

Ranking helps improve weakly-supervised neural semantic parsing Chapter 4 presented a neural parser-ranker framework for weakly supervised semantic parsing. We argued that it is not practical to reply solely on a neural parser to greedily decode the best logical form, given that the parser is trained with weak signal and suffers from the label bias issue. In comparison, the ranker can leverage global features of utterance-logical form-denotation triplets to select the best logical form for execution.

Natural language templates act as an effective interface to present machine language underlying complex tasks In Chapter 5 and Chapter 6, we used a template-based approach to elicit utterance-logical form pairs. A computer program generates logical forms which are deterministically mapped to templates and presented to humans. We experimentally found out that in contrast to Wang et al. (2015)'s method, the readability of templates maintains when the compositionality of the task increases. The template-based approach also allows to elicit both single-turn and sequential utterances.

Non-context-dependent neural semantic parsers can parse sequential utterances well In Chapter 6, we presented both non-context-dependent and context-dependent neural semantic parsers for sequential utterances. Both can parse an utterance into a logical form containing co-referential variables, but the former parser does not make use of previous utterances (a.k.a, context). We experimentally found out that non-context dependent parsing is quite effective. We hypothesize the reason is that a single utterance often provides enough clues to infer an explicit or implicit co-reference variable. Explicit co-reference comes with indicative words such as pronouns (e.g., "*find restaurants in the oxford street?*" followed by "*those with car parks?*"), while in utterances with implicit co-reference the querying object is often missing (e.g., "*find restaurants in the oxford street?*" followed by "*with car parks?*"). It is not difficult for a non-context dependent neural network to learn these indicative patterns of co-reference from training data. An exception is when an utterance contains discourse referents (e.g., a definite noun phrase) for co-reference. The discourse entities denoted by discourse referents refer to the same real world entities as in the antecedent utterance. An example is "*find restaurants in the oxford street?*" followed by "*the oxford restaurants with car parks?*" In this scenario, modeling context may help the parser identify a co-referential relation in the consequent utterance, where "*oxford*" refers to "*oxford street*" and "*the oxford restaurants*" refers to the denotation of the antecedent utterance.

7.3 Future Work

General-purpose Neural Semantic Parsing In this thesis we demonstrate the effectiveness of our neural semantic parsers on domain-specific tasks. One direction of future work is to extend the framework to handle general-purpose meaning representations, such as the Abstract Meaning Representations (Banarescu et al., 2013), the Groningen Meaning Bank (Bos et al., 2017) and the Alexa Meaning Representation Language (Kollar et al., 2018). To handle them, the tree-based generation algorithm presented in Chapter 3 should be extended to generate graphs. The new algorithm requires additional transition actions which generate back-referencing links, similar to the co-reference resolution presented in Chapter 6. This also leads to the question about how to effectively use general-purpose meaning representations for domain-specific tasks. Another question yet to be answered is how to leverage the abundant unlabeled text data to build better general-purpose semantic parsers.

Cross-domain Neural Semantic Parsing An advantage of our modeling framework is that we decompose logical forms into domain-general and domain-specific aspects, but model their generation in a unified way. This characteristics allows to build a cross-domain neural semantic parser where domain-general knowledge is shared (Herzig and Berant, 2017). In this way, we can leverage cross-domain data to improve the model component responsible for domain-general aspects. The more ambitious idea would be to transfer parse structures learned in one or a few domains to another new domain for zero-short semantic parsing (Herzig and Berant, 2018b). This alleviates the cold start problem for new domain semantic parsing.

Data Elicitation and Augmentation Central to the success of neural networks is the availability of a large amount of training data. Two related questions regarding neural semantic parsing are 1) How can we obtain training data more cheaply and efficiently? 2) How can we bootstrap patterns (e.g., grammars or structures) from existing data? Future work can be employed to investigate alternative weak and distant supervision signals, and automatic question generation techniques like back-translation (Sennrich et al., 2016) and generative neural networks (Guu et al., 2018; Yin et al., 2018).

Semantic Parsing in Conversational Agents Context plays an important role in understanding user queries. In Chapter 6, we developed a neural semantic parser for context-dependent utterances. However, we adopted a relatively simple task setup

where a user continuously generates utterances based on intermediate denotations. But the system does not need to consider optimum strategies to interact with the user. As future work, we are interested in designing task-oriented conversational agents with the aid of semantic parsing techniques.

Appendix A

Amazon Mechanical Turk Interface for Data Collection

A.1 Instructions and interface for the static mode data collection of single-turn utterances

A.2 Instructions and interface for the comparative study between data collected by our approach and Wang et al. (2015).

A.3 Instructions and interface for the dynamic mode data collection of single-turn utterances

A.4 Instructions and interface for the static mode data collection of sequential utterances

Instructions

Write down a question that summarizes the task (which queries the table)

You will see a table containing information about meetings (e.g., the attendees, the location of the meeting, the date, its duration). You will also see a list of natural language queries which queries the table and produce an answer. Your job is to write down a natural language question that summarizes the queries. We provide examples of database queries and questions below. You must read the examples first, before doing the task.

We have a Table of meeting schedules.

Meeting Schedules

meeting	attendee	location	date	start time	end time	duration	is important
weekly standup	Alice	central office	jan 2	10 am	3 pm	5 hours	no
annual review	Bob	green cafe	jan 3	2 pm	3 pm	1 hour	yes

You will be given a task that queries the table, in mutiple steps. As an example:

Result1 = find all the **meetings**

Result2 = find **Result1** whose **date** is **jan 2**

Result3 = find **Result2** whose **location** is not **green cafe** (Result3 is the final result that we are interested in.)

Your job is to write down a question sentence that summarizes the task:

Example of a good solution: Which meeting takes place in 2nd January and is not held in green cafe?

Example of a bad solution: Which meeting takes place in 2nd January?

Reason: The solution that does not completely summarize the task.

Example of a bad solution: Find the meeting whose date is jan 2 and location is not green cafe.

Reason: The solution copies the original text, which is not a natural question. **For example: it is natural to ask "the earliest meeting", rather than "meeting whose start time is the smallest". You are encouraged to paraphrase the language to create real-world questions people may ask naturally. Rich language is encouraged.**

Another example:

Result1 = find all the **meetings**

Result2 = find **Result1** whose **date** is **jan 2**

Result3 = count the number of **Result2**

Example of a good solution: How many meetings take place in 2nd January?

Another solution: How many meetings are held in 2nd January?

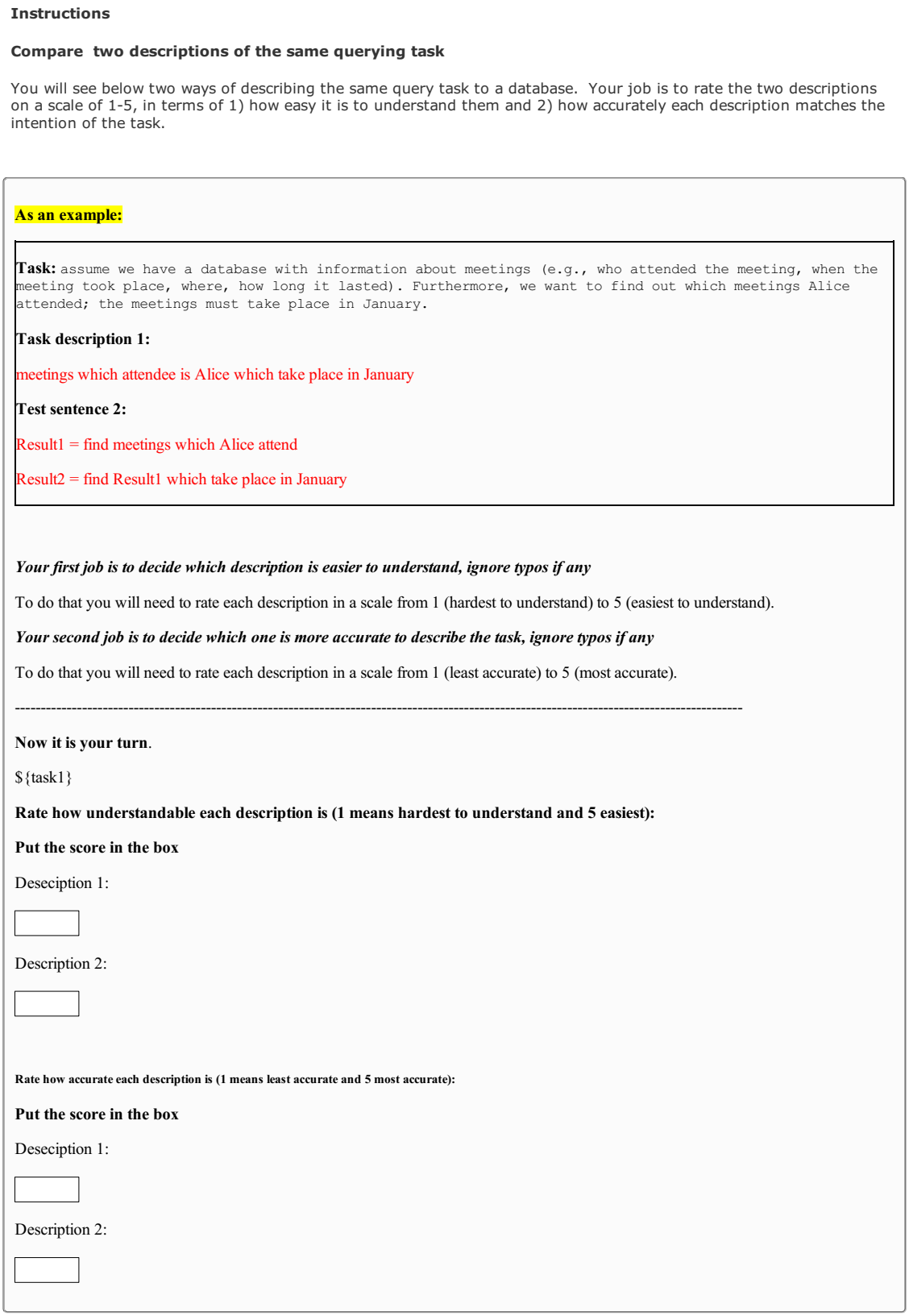
Now it is your turn. Again, ****you must read the good and bad solutions above****, before starting. You have **3** tasks to do before submit.

First task:

${task1}

Write down a question sentence that summarizes the task:

Figure A.1: Instructions and interface for the static mode data collection.



Instructions

Compare two descriptions of the same querying task

You will see below two ways of describing the same query task to a database. Your job is to rate the two descriptions on a scale of 1-5, in terms of 1) how easy it is to understand them and 2) how accurately each description matches the intention of the task.

As an example:

Task: assume we have a database with information about meetings (e.g., who attended the meeting, when the meeting took place, where, how long it lasted). Furthermore, we want to find out which meetings Alice attended; the meetings must take place in January.

Task description 1:

meetings which attendee is Alice which take place in January

Test sentence 2:

Result1 = find meetings which Alice attend

Result2 = find Result1 which take place in January

Your first job is to decide which description is easier to understand, ignore typos if any

To do that you will need to rate each description in a scale from 1 (hardest to understand) to 5 (easiest to understand).

Your second job is to decide which one is more accurate to describe the task, ignore typos if any

To do that you will need to rate each description in a scale from 1 (least accurate) to 5 (most accurate).

Now it is your turn.

${task1}

Rate how understandable each description is (1 means hardest to understand and 5 easiest):

Put the score in the box

Deseciption 1:

Description 2:

Rate how accurate each description is (1 means least accurate and 5 most accurate):

Put the score in the box

Deseciption 1:

Description 2:

Figure A.2: Instructions and interface for the comparative study between data collected by our approach and Wang et al. (2015).

Instructions

Create your querying task based on the given templates and table

You will see a table containing information about meetings (e.g., the attendees, the location of the meeting, the date, its duration). You will also see a list of templates which can be used to query some entry(s) of the table. Your job is to create a querying task by selecting a few templates and fills them in. Finally, you just need to write down a single sentence describing the querying task. You must read the examples first, before doing the task. It tells you how to use the templates and write down your solution.

We have a Table of meeting schedules.

Meeting Schedules

meeting	attendee	location	date	start time	end time	duration	is important
weekly standup	Alice	central office	jan 2	10 am	3 pm	5 hours	no
annual review	Bob	green cafe	jan 3	2 pm	3 pm	1 hour	yes

..... (more rows)

We also have a collection of templates, which can be used to query some entry(s) of the table.

Templates and instructions

template	instruction
find all the __	queries all entries under a key (e.g., find all the meetings)
find the __ of ___	queries the property of a particular key (e.g., find the location of weekly standup)
find the __ whose ___ is ___	queries the key whose property satisfies some is-condition (e.g., find the meeting whose location is green cafe)
find the __ whose ___ is not ___	queries the key whose property satisfies some not-condition (e.g., find the meeting whose location is not green cafe)
find the __ whose ___ is larger than ___	queries the key whose property satisfies some comparative condition (larger than) (e.g., find the meeting whose duration is larger than 5 hours)
find the __ whose ___ is smaller than ___	queries the key whose property satisfies some comparative condition (smaller than) (e.g., find the meeting whose start time is smaller than 2pm)
find the __ whose ___ is largest	queries the key whose property is the largest (e.g., find the meeting whose duration is largest)
find the __ whose ___ is smallest	queries the key whose property is the smallest (e.g., find the meeting whose start time is smallest)
count the number of __	count how many elements are there in a set (e.g., count the number of meeting)

You task is to create a multi-step querying task for the table, using a collection of templates. As an example:

First you are asked to create a querying task

Result1 = find all the **meetings;** **Result2** = find **Result1** whose **date** is **jan 2;** **Result3** = find **Result2** whose **location** is not **green cafe**

(Result3 is the final result that we are interested in.)

Next you are asked to write down a single sentence summarizing the querying task

Which meeting takes place in 2nd January and is not held in green cafe?

That's it!

Figure A.3: Instructions and interface for the dynamic mode data collection.

Instructions

Paraphrase a series of broken English queries into well-formed natural language queries

We have a database containing information about restaurants or hotels (e.g., their names, location, cuisine, meals served, room type, price, etc.). We have also built a computer program which generates queries to this database. Although understandable, the queries are somehwat unnatural and robotic.

Your task is to paraphrase a sequence of computer-generated queries into a natural ones. You must read the examples first, before doing the task.

An example of our database is shown below. We also provide an example of the computer generated queries in broken English. Your task is to paraprase these queries into natural and concise ones that a real user may ask.

Restaurant database used by our computer program

restaurant	cuisine	location	price rating	take credit card?	outdoor seating?
kfc	fast food	city center	$	yes	no
redbox	korean food	seashore	$$	yes	yes

Example

Computer queries	*Your paraphrased queries*	*Another example paraphrased queries*
Q1=find the restaurant where meal-served is lunch	Which restaurants serve lunch?	show me the restaurants that serve lunch?
Q2=find the Q1 where distance is within 1000m	and those near me, no further than 1 km?	Which of these restaurants are within a kilometer to me?
Q3=find the Q2 where customer-rating is 3 stars	Of these, find the ones with 3 star rating.	Which of these restaurants are rated 3 stars by customers?
Q4=find the Q2 where customer-rating is 4 stars	For the restaurants near me, find those with a 4 star rating.	For the restaurants near me, which have 4 star ratings by customers?

The column on the left stores computer generated queries, and the column on the right is for you to fill in. Each computer generated query starts with 'Qx', as an indication of the query ID. In order to create natural language queries, your answer should **take the dependencies between queries into account.** **Try to use concise language** to express these dependencies. (e.g, **if Q1 asks the restaurants serve lunch, then Q2 can simply starts with 'those are near me?', instead of copying the constrain in Q1 again such as 'which restaurants serve lunch and are near me?'. However, in cases when concise language causes ambiguity, you can still specify the right constrain directly**). You are also recouraged to **paraphrase the query as much as you can**, in other words, you should not simply copy the broken English query verbatim.

Just image that you are inputting consecutive queries to Yelp!

Now is your turn. You have to paraphrase alll the given queries in the table before submitting your answers. It is about hotels. Please don't press enter before submission!

Figure A.4: Instructions and interface for the sequential data collection.

Bibliography

Abend, O. and Rappoport, A. (2013). Universal conceptual cognitive annotation (ucca). In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 228–238.

Andreas, J., Vlachos, A., and Clark, S. (2013). Semantic Parsing as Machine Translation. In *Proceedings of Association for Computational Linguistics*, pages 47–52.

Angeli, G., Manning, C. D., and Jurafsky, D. (2012). Parsing time: Learning to interpret time expressions. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 446–455. Association for Computational Linguistics.

Artzi, Y., Das, D., and Petrov, S. (2014). Learning compact lexicons for ccg semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1273–1283.

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

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

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

Asri, L. E., Schulz, H., Sharma, S., Zumer, J., Harris, J., Fine, E., Mehrotra, R., and

Suleman, K. (2017). Frames: A corpus for adding memory to goal-oriented dialogue systems. *arXiv preprint arXiv:1704.00057*.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR 2015*, San Diego, California.

Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The berkeley framenet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics.

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186.

Barr, A. (1980). Natural language understanding. *AI Magazine*, 1(1).

Bast, H. and Haussmann, E. (2015). More accurate question answering on Freebase. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1431–1440. ACM.

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013a). Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington.

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013b). Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA.

Berant, J. and Liang, P. (2014). Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425, Baltimore, Maryland.

Berant, J. and Liang, P. (2015). Imitation learning of agenda-based semantic parsers. *Transactions of the Association for Computational Linguistics*, 3:545–558.

Bisk, Y., Reddy, S., Blitzer, J., Hockenmaier, J., and Steedman, M. (2016). Evaluating induced CCG parsers on grounded semantic parsing. In *Proceedings of the 2016*

Conference on Empirical Methods in Natural Language Processing, pages 2022–2027, Austin, Texas.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM.

Bordes, A., Chopra, S., and Weston, J. (2014). Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 615–620, Doha, Qatar.

Bos, J., Basile, V., Evang, K., Venhuizen, N. J., and Bjerva, J. (2017). The groningen meaning bank. In *Handbook of linguistic annotation*, pages 463–496. Springer.

Cai, J., Shin, R., and Song, D. (2017). Making neural programming architectures generalize via recursion.

Cai, Q. and Yates, A. (2013). Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 423–433, Sofia, Bulgaria.

Chen, D. L. and Mooney, R. J. (2011). Learning to interpret natural language navigation instructions from observations. In *AAAI*, volume 2, pages 1–2.

Cheng, J., Dong, L., and Lapata, M. (2016). Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, Texas.

Cheng, J., Reddy, S., Saraswat, V., and Lapata, M. (2017). Learning structured natural language representations for semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 44–55, Vancouver, Canada.

Cheng, J., Reddy, S., Saraswat, V., and Lapata, M. (2018). Learning an executable neural semantic parser. *Computational Linguistics*.

Clarke, J., Goldwasser, D., Chang, M.-W., and Roth, D. (2010). Driving semantic parsing from the world's response. In *Proceedings of the 14th Conference on Computational Natural Language Learning*, pages 18–27, Uppsala, Sweden.

Das, R., Zaheer, M., Reddy, S., and McCallum, A. (2017). Question answering on knowledge bases and text using universal schema and memory networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 358–365.

Dong, L. and Lapata, M. (2016). Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany.

Dong, L. and Lapata, M. (2018). Coarse-to-fine decoding for neural semantic parsing. *ACL*.

Dong, L., Mallinson, J., Reddy, S., and Lapata, M. (2017). Learning to paraphrase for question answering. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 886–897, Copenhagen, Denmark.

Dong, L., Wei, F., Zhou, M., and Xu, K. (2015). Question answering over Freebase with multi-column convolutional 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)*, pages 260–269, Beijing, China.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., and Smith, N. A. (2015). 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)*, pages 334–343, Beijing, China.

Fan, X., Monti, E., Mathias, L., and Dreyer, M. (2017). Transfer learning for neural semantic parsing. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 48–56.

Gabrilovich, E., Ringgaard, M., and Subramanya, A. (2013). FACC1: Freebase annotation of ClueWeb corpora, version 1 (release date 2013-06-26, format version 1, correction level 0).

Gardner, M. and Krishnamurthy, J. (2017). Open-vocabulary semantic parsing with both distributional statistics and formal knowledge.

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

Goldman, O., Latcinnik, V., Naveh, U., Globerson, A., and Berant, J. (2018). Weakly-supervised semantic parsing with abstract examples. *ACL*.

Goldwasser, D., Reichart, R., Clarke, J., and Roth, D. (2011). Confidence driven unsupervised semantic parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1486–1495, Portland, Oregon, USA.

Guu, K., Hashimoto, T. B., Oren, Y., and Liang, P. (2018). Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, 6:437–450.

Herzig, J. and Berant, J. (2017). Neural semantic parsing over multiple knowledge-bases. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 623–628.

Herzig, J. and Berant, J. (2018a). Decoupling structure and lexicon for zero-shot semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, Melbourne, Australia.

Herzig, J. and Berant, J. (2018b). Decoupling structure and lexicon for zero-shot semantic parsing. *arXiv preprint arXiv:1804.07918*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Iyer, S., Konstas, I., Cheung, A., Krishnamurthy, J., and Zettlemoyer, L. (2017). Learning a neural semantic parser from user feedback. *ACL*.

Iyyer, M., Yih, W.-t., and Chang, M.-W. (2017). Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1821–1831.

Jia, R. and Liang, P. (2016). Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany.

Kamp, H. and Reyle, U. (2013). *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, volume 42. Springer Science & Business Media.

Kate, R. J. and Mooney, R. J. (2006). Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 913–920. Association for Computational Linguistics.

Kate, R. J., Wong, Y. W., and Mooney, R. J. (2005). Learning to Transform Natural to Formal Languages. In *Proceedings for the 20th National Conference on Artificial Intelligence*, pages 1062–1068, Pittsburgh, Pennsylvania.

Kim, J. and Mooney, R. J. (2010). Generative alignment and semantic parsing for learning from ambiguous supervision. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 543–551. Association for Computational Linguistics.

Kim, Y., Denton, C., Hoang, L., and Rush, A. M. (2017). Structured attention networks. *ICLR*.

Kingsbury, P. and Palmer, M. (2002). From treebank to propbank. In *LREC*, pages 1989–1993. Citeseer.

Kitano, H., Tambe, M., Stone, P., Veloso, M., Coradeschi, S., Osawa, E., Matsubara, H., Noda, I., and Asada, M. (1997). The robocup synthetic agent challenge 97. In *Robot Soccer World Cup*, pages 62–73. Springer.

Kollar, T., Berry, D., Stuart, L., Owczarzak, K., Chung, T., Mathias, L., Kayser, M., Snow, B., and Matsoukas, S. (2018). The alexa meaning representation language. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, volume 3, pages 177–184.

Konstas, I., Iyer, S., Yatskar, M., Choi, Y., and Zettlemoyer, L. (2017). Neural amr: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 146–157.

Kočiský, T., Melis, G., Grefenstette, E., Dyer, C., Ling, W., Blunsom, P., and Hermann, K. M. (2016). Semantic parsing with semi-supervised sequential autoencoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1078–1087, Austin, Texas.

Krishnamurthy, J. (2016). 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*, pages 606–616.

Krishnamurthy, J., Dasigi, P., and Gardner, M. (2017). Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526.

Krishnamurthy, J. and Mitchell, T. (2012). Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 754–765, Jeju Island, Korea.

Krishnamurthy, J. and Mitchell, T. M. (2014). Joint syntactic and semantic parsing with combinatory categorial grammar. In *Proceedings of the Association for Computational Linguistics*, pages 1188–1198.

Krishnamurthy, J. and Mitchell, T. M. (2015). Learning a compositional semantics for freebase with an open predicate vocabulary. *Transactions of the Association for Computational Linguistics*, 3:257–270.

Kwiatkowksi, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2010a). Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233, Cambridge, MA.

Kwiatkowksi, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2010b). Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233, Cambridge, MA.

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

on Empirical Methods in Natural Language Processing, pages 1545–1556, Seattle, Washington, USA.

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2011). Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1512–1523, Edinburgh, Scotland.

Lafferty, J., Mccallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–298, San Francisco, CA. Morgan Kaufmann, San Francisco, CA.

Lang, J. and Lapata, M. (2010). Unsupervised induction of semantic roles. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 939–947. Association for Computational Linguistics.

Lawrence, C. and Riezler, S. (2018). Improving a neural semantic parser by counterfactual learning from human bandit feedback. *ACL*.

Lee, K., Artzi, Y., Dodge, J., and Zettlemoyer, L. (2014). Context-dependent semantic parsing for time expressions. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1437–1447.

Lemon, O., Georgila, K., Henderson, J., and Stuttle, M. (2006). An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the talk in-car system. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics: Posters & Demonstrations*, pages 119–122, Trento, Italy.

Liang, C., Berant, J., Le, Q., Forbus, K. D., and Lao, N. (2017). Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 23–33.

Liang, P. (2013). Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*.

Liang, P. (2016). Learning executable semantic parsers for natural language understanding. *Communications of the ACM*, 59(9):68–76.

Liang, P., Jordan, M., and Klein, D. (2011). Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 590–599, Portland, Oregon.

Liu, Y. and Lapata, M. (2017). Learning structured text representations. *EMNLP*.

Long, R., Pasupat, P., and Liang, P. (2016). Simpler context-dependent logical forms via model projections. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1456–1465, Berlin, Germany.

Lu, W., Ng, H. T., Lee, W. S., and Zettlemoyer, L. S. (2008). A generative model for parsing natural language to meaning representations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 783–792. Association for Computational Linguistics.

Matuszek, C., FitzGerald, N., Zettlemoyer, L., Bo, L., and Fox, D. (2012). A joint model of language and perception for grounded attribute learning. In Langford, J. and Pineau, J., editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1671–1678.

Matuszek, C., Herbst, E., Zettlemoyer, L., and Fox, D. (2013). Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer.

McTear, M. F. (2004). Spoken dialogue applications: Research directions and commercial deployment. In *Spoken Dialogue Technology*, pages 19–43. Springer.

Mesnil, G., Dauphin, Y., Yao, K., Bengio, Y., Deng, L., Hakkani-Tur, D., He, X., Heck, L., Tur, G., Yu, D., et al. (2015). Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539.

Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, pages 1791–1799.

Moe, W. W. and Fader, P. S. (2001). Uncovering patterns in cybershopping. *California Management Review*, 43(4):106–117.

Neelakantan, A., Le, Q. V., and Sutskever, I. (2016). Neural programmer: Inducing latent programs with gradient descent. *ICLR*.

Parikh, A., Täckström, O., Das, D., and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, Austin, Texas.

Pasupat, P. and Liang, P. (2015). Compositional semantic parsing on semi-structured tables. 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)*, volume 1, pages 1470–1480.

Pasupat, P. and Liang, P. (2016). Inferring logical forms from denotations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 23–32.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar.

Poon, H. and Domingos, P. (2010). Unsupervised semantic parsing. *Association for Computational Linguistics*, pages 1–10.

Pradhan, S. S., Ward, W. H., Hacioglu, K., Martin, J. H., and Jurafsky, D. (2004). Shallow semantic parsing using support vector machines. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*.

Price, P. J. (1990). Evaluation of spoken language systems: The atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

Rabinovich, M., Stern, M., and Klein, D. (2017). Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149, Vancouver, Canada.

Reddy, S., Lapata, M., and Steedman, M. (2014). Large-scale semantic parsing without question-answer pairs. *Transactions of the Association of Computational Linguistics*, 2(1):377–392.

Reddy, S., Täckström, O., Collins, M., Kwiatkowski, T., Das, D., Steedman, M., and Lapata, M. (2016). Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140.

Reddy, S., Täckström, O., Petrov, S., Steedman, M., and Lapata, M. (2017). Universal semantic parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 89–101, Copenhagen, Denmark.

Reed, S. and De Freitas, N. (2016). Neural programmer-interpreters. *ICLR*.

Sennrich, R., Haddow, B., and Birch, A. (2016). Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 86–96.

Steedman, M. (2000). The syntactic process.

Su, Y., Sun, H., Sadler, B., Srivatsa, M., Gur, I., Yan, Z., and Yan, X. (2016). On generating characteristic-rich question sets for qa evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 562–572, Austin, Texas.

Suhr, A., Iyer, S., and Artzi, Y. (2018). Learning to map context-dependent sentences to executable formal queries. NAACL.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, Atlanta, Georgia.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112. MIT Press.

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

Wang, Y., Berant, J., Liang, P., et al. (2015). Building a semantic parser overnight. In *Proceedings of the 53th Annual Meeting of the Association for Computational Linguistics*.

Webber, B. L. (1978). A formal approach to discourse anaphora. Technical report, BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA.

Wen, T.-H., Gasic, M., Mrkšić, N., Su, P.-H., Vandyke, D., and Young, S. (2015). Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721, Lisbon, Portugal.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

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

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

Winograd, T. (1975). Frame representations and the declarative/procedural controversy. In *Representation and understanding*, pages 185–210. Elsevier.

Wong, Y. W. and Mooney, R. (2006). Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446, New York City, USA.

Wong, Y. W. and Mooney, R. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967.

Woods, W. A., Kaplan, R. M., and Nash-Webber, B. (1972). The Lunar sciences: Natural language information system: Final report. Technical Report BBN Report 2378, Bolt Beranek and Newman.

Xu, K., Reddy, S., Feng, Y., Huang, S., and Zhao, D. (2016). Question answering on Freebase via relation extraction and textual evidence. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2326–2336, Berlin, Germany.

Xu, X., Liu, C., and Song, D. (2017). Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Yao, X. and Van Durme, B. (2014). Information extraction over structured data: Question answering with Freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 956–966, Baltimore, Maryland.

Yih, W.-t., Chang, M.-W., He, X., and Gao, J. (2015). Semantic parsing via staged query graph generation: Question answering with knowledge base. 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)*, pages 1321–1331, Beijing, China.

Yin, P. and Neubig, G. (2017). A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada.

Yin, P., Zhou, C., He, J., and Neubig, G. (2018). Structvae: Tree-structured latent variable models for semi-supervised semantic parsing. *ACL*.

Young, S., Gašić, M., Thomson, B., and Williams, J. D. (2013). Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179.

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

Zelle, J. M. (1996). Learning to parse database queries using inductive logic programming. In *AAAI*.

Zettlemoyer, L. and Collins, M. (2007). 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 (EMNLP-CoNLL), pages 678–687, Prague, Czech Republic.

Zettlemoyer, L. S. and Collins, M. (2005a). Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *Proceedings of 21st Conference in Uncertainilty in Artificial Intelligence*, pages 658–666, Edinburgh, Scotland.

Zettlemoyer, L. S. and Collins, M. (2005b). Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *Proceedings of 21st Conference in Uncertainilty in Artificial Intelligence*, pages 658–666, Edinburgh, Scotland.

Zettlemoyer, L. S. and Collins, M. (2009). Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 976–984, Suntec, Singapore.

Zhao, K. and Huang, L. (2015). Type-driven incremental semantic parsing with polymorphism. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1416–1421, Denver, Colorado.

Zhong, V., Xiong, C., and Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.