The Dissertation Committee for Islam Kamel Ahmed Beltagy
certifies that this is the approved version of the following dissertation:

Natural Language Semantics Using Probabilistic Logic

Committee:

Raymond J. Mooney, Supervisor

Katrin Erk, Co-Supervisor

Vibhav Gogate

Vladimir Lifschitz

Pradeep Ravikumar

Natural Language Semantics Using Probabilistic Logic

by

Islam Kamel Ahmed Beltagy, B.S.; M.S.; M.S.Comp.Sci.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2016

Acknowledgments

I wish to thank the multitudes of people who helped me throughout the past 5 years of my life. I would like to thank my advisors Ray Mooney and Katrin Erk. They formed a great advising team, and I am grateful for their continuous intellectual and personal support. I would like to thank my advising committee, Vibhav Gogate, Vladimir lifschitz and Pradeep Ravikumar for their comments and insights. Special thanks to Lydia Griffith and Stacy Miller for their tremendous efforts to make our lives easier. I also thank my friends and lab mates for being a good company, especially Stephen, Subhashini, Nazneen and Karl. I also thank my friends M. Moustafa, Abdulkarim, Ehab, Amr and Kamran for being a family throughout the years of the PhD. I am also grateful to my Mom and Dad for setting me on the path to success, brothers and sisters for being my big happy family and mom-in-law for being a great help. Finally, I thank my wife Sarah and daughter Zaina for being a loving family that life would have been a lot harder without them.

Islam Beltagy

The University of Texas at Austin

August 2016

Natural Language Semantics Using Probabilistic Logic

Publication No. _____

Islam Kamel Ahmed Beltagy, Ph.D.
The University of Texas at Austin, 2016

Supervisor: Raymond J. Mooney
Co-Supervisor: Katrin Erk

With better natural language semantic representations, computers can do more applications more efficiently as a result of better understanding of natural text. However, no single semantic representation at this time fulfills all requirements needed for a satisfactory representation. Logic-based representations like first-order logic capture many of the linguistic phenomena using logical constructs, and they come with standardized inference mechanisms, but standard first-order logic fails to capture the "graded" aspect of meaning in languages. Other approaches for semantics, like distributional models, focus on capturing "graded" semantic similarity of words and phrases but do not capture sentence structure in the same detail as logic-based approaches. However, both aspects of semantics, structure and gradedness, are important for an accurate language semantics representation.

In this work, we propose a natural language semantics representation that uses probabilistic logic (PL) to integrate logical with weighted uncertain

knowledge. It combines the expressivity and the automated inference of logic with the ability to reason with uncertainty. To demonstrate the effectiveness of our semantic representation, we implement and evaluate it on three tasks, recognizing textual entailment (RTE), semantic textual similarity (STS) and open-domain question answering (QA). These tasks can utilize the strengths of our representation and the integration of logical representation and uncertain knowledge. Our semantic representation [1] has three components, Logical Form, Knowledge Base and Inference, all of which present interesting challenges and we make new contributions in each of them.

The first component is the Logical Form, which is the primary meaning representation. We address two points, how to translate input sentences to logical form, and how to adapt the resulting logical form to PL. First, we use Boxer, a CCG-based semantic analysis tool to translate sentences to logical form. We also explore translating dependency trees to logical form. Then, we adapt the logical forms to ensure that universal quantifiers and negations work as expected.

The second component is the Knowledge Base which contains "uncertain" background knowledge required for a given problem. We collect the "relevant" lexical information from different linguistic resources, encode them as weighted logical rules, and add them to the knowledge base. We add rules from existing databases, in particular WordNet and the Paraphrase Database

[1] System is available for download at: `https://github.com/ibeltagy/pl-semantics`

(PPDB). Since these are incomplete, we generate additional on-the-fly rules that could be useful. We use alignment techniques to propose rules that are relevant to a particular problem, and explore two alignment methods, one based on Robinson's resolution and the other based on graph matching. We automatically annotate the proposed rules and use them to learn weights for unseen rules.

The third component is Inference. This component is implemented for each task separately. We use the logical form and the knowledge base constructed in the previous two steps to formulate the task as a PL inference problem then develop a PL inference algorithm that is optimized for this particular task. We explore the use of two PL frameworks, Markov Logic Networks (MLNs) and Probabilistic Soft Logic (PSL). We discuss which framework works best for a particular task, and present new inference algorithms for each framework.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Computational semantics studies mechanisms for encoding the meaning of natural language in a machine-friendly representation that supports automated reasoning and that, ideally, can be automatically acquired from large text corpora. Effective semantic representations and reasoning tools give computers the power to perform complex applications like question answering. But applications of computational semantics are very diverse and pose differing requirements on the underlying representational formalism. Some applications benefit from a detailed representation of the structure of complex sentences. Some applications require the ability to recognize near-paraphrases or degrees of similarity between sentences. Some applications require inference, either exact or approximate. Often it is necessary to handle ambiguity and vagueness in meaning. Finally, we frequently want to learn knowledge relevant to these applications automatically from corpus data.

There is no single representation for natural language meaning at this time that fulfills all of the above requirements, but there are representations that fulfill some of them. Logic-based representations (Montague, 1970; Dowty, Wall, & Peters, 1981; Kamp & Reyle, 1993) like first-order logic repre-

sent many linguistic phenomena like negation, quantifiers, or discourse entities. Some of these phenomena (especially representing negation scope and keeping track of discourse entities over larger stretches of discourse) can not be easily represented in syntax-based representations like Natural Logic (MacCartney & Manning, 2009). In addition, first-order logic has standardized inference mechanisms. Consequently, logical approaches have been widely used in semantic parsing where it supports answering complex natural language queries requiring reasoning and data aggregation (Zelle & Mooney, 1996; Kwiatkowski, Choi, Artzi, & Zettlemoyer, 2013; Pasupat & Liang, 2015). But logic-based representations often assume predetermined fixed ontology, which requires a lot of training data to learn the mapping to logical form, and limits the applicability of the representation to this one particular ontology. And first-order logic, being binary in nature, does not capture the graded aspect of meaning. Other models, like distributional models (Turney & Pantel, 2010), focus on representing the graded semantic similarity of words and phrases (Landauer & Dumais, 1997; Mitchell & Lapata, 2010). Both capabilities, representing logical structure and capturing graded information, are clearly complementary for an accurate semantic representation.

Our aim is to construct a general-purpose natural language understanding system that provides in-depth representations of sentence meaning amenable to automated inference, but that also allows for flexible and graded inferences involving word meaning. Therefore, our approach combines logical representation with a "weighted" knowledge base for robust semantic rep-

resentation. Specifically, we use first-order logic as a basic representation, providing a sentence representation that can be easily interpreted and manipulated. However, we also use a weighted knowledge base that encodes a graded representation for words and short phrases, providing information on near-synonymy and lexical entailment. Uncertainty and gradedness at the lexical and phrasal level should inform inference at all levels, so we rely on probabilistic logic (PL) to process the logical form and the weighted knowledge base. Our framework is three components, Logical Form, Knowledge Base and Inference, all of which present interesting challenges and we make new contributions in each of them.

Tasks and System Architecture To demonstrate the generality and effectiveness of our proposed semantic representation, we implement and evaluate it on three tasks. The first is recognizing textual entailment (RTE) (Dagan, Roth, Sammons, & Zanzotto, 2013), the task of finding if a sentence entails, contradicts or is neutral to another sentence. The second is semantic textual similarity (STS) (Agirre, Cer, Diab, & Gonzalez-Agirre, 2012), the task of evaluating the semantic similarity of two sentence on a scale from 1 to 5. The third is open-domain question answering (QA), the task of answering a question given a short document containing the answer (not from a database of facts) (Richardson, Burges, & Renshaw, 2013; Hermann et al., 2015). These tasks can utilize the strengths of our representation and the integration of logical representation and uncertain knowledge.

Our approach has three main components,

1. Logical Form, where input sentences are mapped to logical then the logical form is adapted for PL.

2. Knowledge Base, where we collect the relevant lexical information from different linguistic resources, encode them as weighted logical rules and add them to the inference problem.

3. Inference, takes the output of the previous two steps, formulate the task as PL inference problem then solve it using the appropriate PL implementation.

In our framework, we can view each language understanding task as consisting of a *text* T and a *hypothesis* H, along with a *knowledge base* KB. The text T describes some situation or setting, and the hypothesis H in the simplest case asks whether a particular statement is true in the situation described in T, and the knowledge base KB encodes relevant background knowledge. Formally, this is checking if the text and knowledge base entail the hypothesis: $T \wedge KB \Rightarrow H$, and its probabilistic counterpart is calculating the probability $P(H|T, KB)$. We will present all of our work from the perspective of solving this single entailment, and we will show in later chapters how each task can be represented in terms of this entailment.

Logical Form The Logical Form component starts with translating input sentences to logical form. For example, the sentences

T: A grumpy ogre is not smiling.

H: A sad ogre is not smiling.

will be translated to

T: $\exists x.\, ogre(x) \wedge grumpy(x) \wedge \neg\exists y.\, agent(y, x) \wedge smile(y)$

H: $\exists x.\, ogre(x) \wedge sad(x) \wedge \neg\exists y.\, agent(y, x) \wedge smile(y)$

We use Boxer (Bos, 2008), a rule-based semantic analysis tool that runs on top of a CCG parse (Clark & Curran, 2004), for this translation. We also developed a rule-based transformation to translate dependency trees to logical forms that are less expressive but more robust, which make them suitable for the long complex text (as in the QA task). Logical forms need to be adapted to PL because PL frameworks make the Domain Closure Assumption (DCA), which states that there are no objects in the universe other than the named constants (Richardson & Domingos, 2006). This means that constants need to be explicitly introduced in the domain in order to get the expected inferences using PL. We discuss how to do these adaptations and evaluate them on three entailment datasets including a synthetic dataset that exhaustively tests inference performance on sentences with two quantifiers.

Knowledge Base The knowledge base encodes relevant background knowledge: lexical knowledge, world knowledge, or both. We collect the relevant

rules for a particular T and H from a variety of linguistic resources then encode them as "weighted" first order rules. For the example above, we will need the rule:

$$r_1\colon\ \forall x.\, grumpy(x) \Rightarrow sad(x)\ \mid\ w_1$$

where w_1 is a weight indicating how true this rule is. We collect two types of rules, the first is precompiled rules collected from existing resources like WordNet (Princeton University, 2010) and PPDB (Ganitkevitch, Van Durme, & Callison-Burch, 2013). These resources are never complete, so the second type of rules are on-the-fly rules that we generate for a particular T and H pair. We align T and H then use the alignment to extract the rules relevant for them. We use a lexical entailment classifier to learn how to weight the extracted rules. We have two alignment techniques, a Robinson resolution-based alignment which works for short pairs of T and H (like for RTE and STS), and a graph-based alignment which works for longer text T (like in QA). We evaluate this component of our system by its impact on the end task. We evaluate the precompiled rules and the Robinson resolution alignment on the RTE task, and the graph-based alignment on the QA task.

Inference The third component is Inference. We use the logical form and the weighted knowledge base collected from the previous two steps to formulate a task specific PL inference problem of the form $P(H|T, KB)$. The RTE task is represented in terms of two PL inferences, one to decide between

entail and neutral, and the other to decide between contradiction and neutral. The STS task is represented in terms of two inferences, one from the first sentence to the second, and the other from the second to the first. The QA task is represented with the PL inference of finding an entity e from T that maximizes probability of H. Given the formulated inference problem, we use the appropriate PL framework to solve it. We use two PL frameworks, Markov Logic Networks (MLN) (Richardson & Domingos, 2006) and Probabilistic Soft Logic (PSL) (Kimmig, Bach, Broecheler, Huang, & Getoor, 2012; Bach, Huang, London, & Getoor, 2013). MLNs and PSL are Statistical Relational Learning (SRL) techniques (Getoor & Taskar, 2007) that combine logical and statistical knowledge in one uniform framework, and provide a mechanism for coherent probabilistic inference. PL frameworks represent uncertainty in terms of weights on the logical rules as in the example below:

$$\begin{aligned} &\forall x.\, ogre(x) \Rightarrow grumpy(x) \mid 1.5 \\ &\forall x, y.\, (friend(x, y) \wedge ogre(x)) \Rightarrow ogre(y) \mid 1.1 \end{aligned} \tag{1.1}$$

which states that there is a chance that ogres are grumpy, and friends of ogres tend to be ogres too. The PL tools we use, namely MLNs and PSL, employ such weighted rules to derive a probability distribution over possible worlds through an undirected graphical model. This probability distribution over possible worlds is then used to draw inferences. MLNs and PSL can be viewed as templates that facilitate construction of large complex graphical models, and do inference over them to answer queries.

The logical form and the knowledge base are mostly task independent.

Inference however, is task dependent, and we will split its discussion into three chapters, one for each task. Each chapter explains how a task is represented in PL, then presents inference algorithm(s) for the task using the appropriate PL tool. A general problem with PL inference is computational intractability because PL usually generates large graphical models and graphical model inference is intractable. This is an issue that we address in all of our implementations of PL inference.

For RTE, we used MLNs and we implement an inference algorithm that directly supports querying complex logical formula (which is not supported in the available MLN tools) and it exploits the closed-world assumption to make inference more tractable. For STS, we use MLNs and PSL, and show how to perform a form of "partial entailment" that is more fit for the task. For the QA task, instead of using PSL or MLNs to build a graphical model and do inference, we have our own implementation that is much faster. It takes T, H and KB and build a graphical model representing the problem, then implement our graphical model inference to answer the query.

Notation and Terminology This document refers to different types of entailments:

- Recognizing Textual Entailment (RTE): is the task of given a text T and hypothesis H, find if T entails, contradicts or neutral to H
- Entailment: is the more basic task of finding if T entails H. This is

the basic operation that we use to define the three other tasks. This entailment is denoted by the probabilistic inference $P(H|T, KB)$

- Lexical entailment: finding entailment relations between words or short phrases (Roller, Erk, & Boleda, 2014). We use it to weight rules of the knowledge base.

We call inputs for the RTE task text T and hypothesis H, inputs for the STS task are first and second sentences S_1, S_2, and inputs for the QA task are document D and query Q. Once a task is formulated as $P(H|T, KB)$, we will only refer to text T and hypothesis H. KB is the knowledge base rules. From the PL perspective, H is called query Q and T can be called evidence E.

1.1 Thesis Contributions

The main contribution of this thesis is developing a practical system that uses PL to integrate logical information with weighted uncertain knowledge. It explains the challenges of bringing together the three distinct components of our approach and how we address them. It addresses the following challenges:

- Adapting logical form for PL to make sure universal quantifiers and negations work as expected. Adaptations are evaluated on three entailment datasets including a synthetic dataset that exhaustively tests inference performance on sentences with two quantifiers.

- Two different methods for aligning the text and hypothesis and extract on-the-fly rules, then learning how to assess them. One alignment method is based on Robinson resolution (works for RTE and STS) and the other is based on graph matching (works for QA). We released a dataset of all rules extracted from one of the RTE datasets. This is a valuable resource for testing lexical entailment systems.

- An MLN inference algorithm that calculates the probability of a query formula (not a single ground atom) and it is computationally efficient for the type of inference problems we are interested in.

- New MLN and PSL inference algorithms for the STS task that support partial entailments. The MLN algorithm replaces conjunction with an average combiner, and the PSL algorithm uses a new relaxed conjunction operator and a heuristic grounding algorithm that fits it.

- Formulating our own graphical model representation for the the QA task (instead of using MLNs or PSL), and developing an inference algorithm that answers the question encoded in in this graphical model. Our formulation and inference is more than two orders of magnitude faster than PSL.

We also addressed the following smaller problems

- Formulating the RTE task, the STS task and the QA task as PL inference problems.

- A rule-based method to translate dependency parse trees into Boxer like logical form that is less expressive but more robust.
- A rule-based method to translate existing paraphrase resources into logical rules.
- A special entity coreference assumption that is necessary for the detection of contradictions in the RTE task.
- A simple weight learning approach to map rule weights to MLN weights.
- Using multiple CCG parses to increase robustness of the translation from text to logical form.
- Preliminary PSL inference algorithm that fits the requirements of the QA task. We only use this inference algorithm for the comparison with our graphical model formulation.

It should be noted that all of the contributions listed above except the graph-based alignment and the QA work, have appeared in our previous publications (Beltagy, Chau, Boleda, Garrette, Erk, & Mooney, 2013; Beltagy, Erk, & Mooney, 2014; Beltagy & Mooney, 2014; Beltagy & Erk, 2015; Beltagy, Roller, Cheng, Erk, & Mooney, 2016)

1.2 Thesis Outline

This thesis is organized by components. We present the logical form component, followed by the knowledge base component. These two chapters

are mostly task independent. Inference is task dependent so we dedicate a chapter for the details of each task, RTE, STS and QA. For readability, we present the QA-specific logical form and knowledge base in the QA chapter, not in the logical form and knowledge base chapters.

- Chapter 2 reviews background topics and discusses related work.
- Chapter 3 explains the Logical Form adaptations.
- Chapter 4 discusses how we build the Knowledge Base from precompiled rules and from on-the-fly rules using Robinson resolution alignment.
- Chapter 5 discusses the details of implementing our system for the RTE task and our inference algorithms for RTE.
- Chapter 6 presents MLN and PSL inference algorithms tuned for the STS task
- Chapter 7 presents our work for the QA task which include translating dependency parses to logical form, collecting knowledge base rules using a graph-based alignment and doing inference for the QA task.
- Chapter 8 lists some ideas for future work.
- Chapter 9 is the conclusion.

Chapter 2

Background and Related Work

2.1 Logical Semantics

Logical representations of meaning have a long tradition in linguistic semantics (Montague, 1970; Dowty et al., 1981; Kamp & Reyle, 1993; Alshawi, 1992) and computational semantics (Blackburn & Bos, 2005; van Eijck & Unger, 2010), and commonly used in semantic parsing (Zelle & Mooney, 1996; Berant, Chou, Frostig, & Liang, 2013; Kwiatkowski et al., 2013). They handle many complex semantic phenomena such as negation and quantifiers, they identify discourse referents along with the predicates that apply to them and the relations that hold between them. However, standard first-order logic and theorem provers are binary in nature, which prevents them from capturing the graded aspects of meaning in language: Synonymy seems to come in degrees (Edmonds & Hirst, 2000), as does the difference between senses in polysemous words (Brown, 2008). van Eijck and Lappin (2012) write: "The case for abandoning the categorical view of competence and adopting a probabilistic model is at least as strong in semantics as it is in syntax."

Recent wide-coverage tools that use logic-based sentence representations include Copestake and Flickinger (2000), Bos (2008), and Lewis and

Steedman (2013). We use Boxer (Bos, 2008), a wide-coverage semantic analysis tool that produces logical forms using Discourse Representation Structures (Kamp & Reyle, 1993). It builds on the C&C CCG (Combinatory Categorial Grammar) parser (Clark & Curran, 2004) and maps sentences into a lexically-based logical form, in which the predicates are mostly words in the sentence. For example, the sentence *An ogre loves a princess* is mapped to:

$$\exists x, y, z.\, ogre(x) \wedge agent(y, x) \wedge love(y) \wedge patient(y, z) \wedge princess(z) \quad (2.1)$$

As can be seen, Boxer uses a neo-Davidsonian framework (Parsons, 1990): y is an event variable, and the semantic roles *agent* and *patient* are turned into predicates linking y to the agent x and patient z.

2.2 Distributional Semantics

Distributional models (Turney & Pantel, 2010) use statistics on contextual data from large corpora to predict semantic similarity of words and phrases (Landauer & Dumais, 1997; Mitchell & Lapata, 2010). They are motivated by the observation that semantically similar words occur in similar contexts, so words and short phrase can be represented as vectors in high dimensional spaces generated from the contexts in which they occur (Landauer & Dumais, 1997; Lund & Burgess, 1996). Therefore, distributional models are relatively easier to build than logical representations, automatically acquire knowledge from large corpora, and capture the *graded* nature of linguistic meaning, but they do not adequately capture logical structure (Grefenstette, 2013).

Distributional information is being using in our system to assist building our knowledge base. In particular, we use it to inform the lexical entailment classifier about similarity of words and short phrases (section 4.3.2).

2.3 Probabilistic Logic (PL)

PL are tools that combine logical and statistical knowledge in one representation. The PL frameworks we use, namely MLNs and PSL, are Statistical Relational Learning (SRL) techniques (Getoor & Taskar, 2007) that combine logical and statistical knowledge through graphical model inference. PL frameworks typically employ weighted formulas in first-order logic to compactly encode complex probabilistic graphical models. Weighting the rules is a way of softening them compared to hard logical constraints and thereby allowing truth assignments in which not all instances of the rule hold. Equation 1.1 shows sample weighted rules: Friends of ogres tend to be ogres and ogres tend to be grumpy. Suppose we have two constants, A and B. Using these two constants and the predicate symbols in Equation 1.1, the set of all ground atoms we can construct is:

$$L_{A,B} = \{ogre(A), ogre(B), grumpy(A), grumpy(B), friend(A, A),$$
$$friend(A, B), friend(B, A), friend(B, B)\}$$

If we only consider models over a domain with these two constants as entities, then each truth assignment to $L_{A,B}$ corresponds to a model. PL frameworks usually make the assumption of a one-to-one correspondence between constants in the system and entities in the domain. We discuss the effects of this

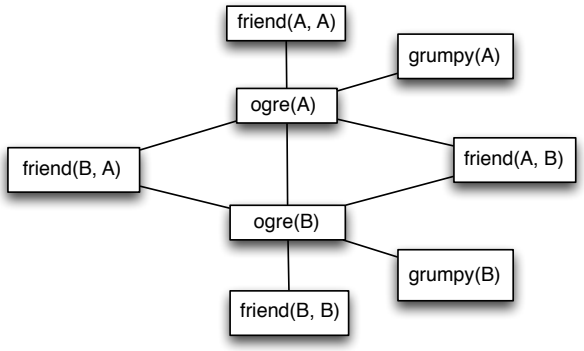


Figure 2.1: A sample MLN ground network

Domain Closure Assumption (DCA) in chapter 3.

2.3.1 Markov Logic Networks (MLNs)

MLNs (Richardson & Domingos, 2006) are one of the PL frameworks we use. MLNs work as template to construct undirected graphical models. Markov Networks or undirected graphical models (Pearl, 1988) compute the probability $P(X = x)$ of an assignment x of values to the sequence X of all variables in the model based on clique potentials, where a clique potential is a function that assigns a value to each clique (maximally connected subgraph) in the graph. Markov Logic Networks construct Markov Networks (hence their name) based on weighted first order logic formulas, like the ones in Equation 1.1. Figure 2.1 shows the network for Equation 1.1 with two constants. Every ground atom becomes a node in the graph, and two nodes are connected if they co-occur in a grounding of an input formula. In this graph, each clique corresponds to a grounding of a rule. For example, the clique in-

cluding $friend(A, B)$, $ogre(A)$, and $ogre(B)$ corresponds to the ground rule $friend(A, B) \wedge ogre(A) \Rightarrow ogre(B)$. A variable assignment x in this graph assigns to each node a value of either True or False, so it is a truth assignment (a world). The clique potential for the clique involving $friend(A, B)$, $ogre(A)$, and $ogre(B)$ is $\exp(1.1)$ if x makes the ground rule true, and 0 otherwise. This allows for nonzero probability for worlds x in which not all friends of ogres are also ogres, but it assigns exponentially more probability to a world for each ground rule that it satisfies.

More generally, an MLN takes as input a set of weighted first-order formulas $F = F_1, \ldots, F_n$ and a set C of constants, and constructs an undirected graphical model in which the set of nodes is the set of ground atoms constructed from F and C. It computes the probability distribution $P(X = x)$ over worlds based on this undirected graphical model. The probability of a world (a truth assignment) x is defined as:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right) \tag{2.2}$$

where i ranges over all formulas F_i in F, w_i is the weight of F_i, $n_i(x)$ is the number of groundings of F_i that are true in the world x, and Z is the partition function (i.e., it normalizes the values to probabilities). So the probability of a world increases exponentially with the total weight of the ground clauses that it satisfies.

Below, we use R (for rules) to denote the input set of weighted formulas. In addition, an MLN takes as input an evidence set E asserting truth values for

some ground clauses. For example, $ogre(A)$ means that A is an ogre. Marginal inference for MLNs calculates the probability $P(Q|E, R)$ for a query formula Q.

Alchemy (Kok, Singla, Richardson, & Domingos, 2005) is the most widely used MLN implementation. It is a software package that contains implementations of a variety of MLN inference and learning algorithms. However, developing a scalable, general-purpose, accurate inference method for complex MLNs is an open problem. MLNs have been used for various NLP applications including unsupervised coreference resolution (Poon & Domingos, 2008), semantic role labeling (Riedel & Meza-Ruiz, 2008) and event extraction (Riedel, Chun, Takagi, & Tsujii, 2009).

2.3.2 Probabilistic Soft Logic (PSL)

Probabilistic Soft Logic (PSL) is recently proposed alternative framework for PL (Kimmig et al., 2012; Bach et al., 2013). It uses logical representations to compactly define large graphical models with "continuous" variables, and includes methods for performing efficient probabilistic inference for the resulting models. A key distinguishing feature of PSL is that ground atoms have soft, continuous truth values in the interval [0, 1] rather than binary truth values as used in MLNs and most other PL tools. Given a set of weighted logical formulas, PSL builds a graphical model defining a probability distribution over the continuous space of values of the random variables in the model. A PSL model is defined using a set of weighted if-then rules in first-order logic,

as in the following example:

$$\begin{aligned} &\forall x, y, z.\ friend(x, y) \wedge votesFor(y, z) \Rightarrow votesFor(x, z) \mid 0.3 \\ &\forall x, y, z.\ spouse(x, y) \wedge votesFor(y, z) \Rightarrow votesFor(x, z) \mid 0.8 \end{aligned} \tag{2.3}$$

The first rule states that a person is likely to vote for the same person as his/her friend. The second rule encodes the same regularity for a person's spouse. The weights encode the knowledge that a spouse's influence is greater than a friend's in this regard.

In addition, PSL includes *similarity functions*. Similarity functions take two strings or two sets as input and return a truth value in the interval [0, 1] denoting the similarity of the inputs. For example, this is a rule that incorporate the similarity of two predicates:

$$\forall x.\ similarity(\text{“}predicate1\text{”}, \text{“}predicate2\text{”}) \wedge predicate1(x) \Rightarrow predicate2(x) \tag{2.4}$$

As mentioned above, each ground atom, a, has a soft truth value in the interval [0, 1], which is denoted by $I(a)$. To compute soft truth values for logical formulas, Lukasiewicz's relaxation of conjunctions($\wedge$), disjunctions($\vee$) and negations($\neg$) are used:

$$\begin{aligned} I(l_1 \wedge l_1) &= max\{0, I(l_1) + I(l_2) - 1\} \\ I(l_1 \vee l_1) &= min\{I(l_1) + I(l_2), 1\} \\ I(\neg l_1) &= 1 - I(l_1) \end{aligned} \tag{2.5}$$

Then, a given rule $r \equiv r_{body} \Rightarrow r_{head}$, is said to be *satisfied* (i.e. $I(r) = 1$) iff $I(r_{body}) \leq I(r_{head})$. Otherwise, PSL defines a *distance to satisfaction* $d(r)$

which captures how far a rule r is from being satisfied: $d(r) = max\{0, I(r_{body}) - I(r_{head})\}$. For example, assume we have the set of evidence: $I(spouse(B, A)) = 1$, $I(votesFor(A, P)) = 0.9$, $I(votesFor(B, P)) = 0.3$, and that r is the resulting ground instance of rule (2.3). Then $I(spouse(B, A) \wedge votesFor(A, P)) = max\{0, 1 + 0.9 - 1\} = 0.9$, and $d(r) = max\{0, 0.9 - 0.3\} = 0.6$.

Using *distance to satisfaction*, PSL defines a probability distribution over all possible interpretations I of all ground atoms. The pdf is defined as follows:

$$\begin{aligned} p(I) &= \frac{1}{Z} \exp\left[-\sum_{r \in R} \lambda_r (d(r))^p\right]; \\ Z &= \int_I \exp\left[-\sum_{r \in R} \lambda_r (d(r))^p\right] \end{aligned} \tag{2.6}$$

where Z is the normalization constant, λ_r is the weight of rule r, R is the set of all rules, and $p \in \{1, 2\}$ provides two different loss functions. For our application, we always use $p = 1$

PSL is primarily designed to support MPE inference (Most Probable Explanation). MPE inference is the task of finding the overall interpretation with the maximum probability given a set of evidence. Intuitively, the interpretation with the highest probability is the interpretation with the lowest distance to satisfaction. In other words, it is the interpretation that tries to satisfy all rules as much as possible. Formally, from equation 2.6, the most probable interpretation, is the one that minimizes $\sum_{r \in R} \lambda_r (d(r))^p$. In case of $p = 1$, and given that all $d(r)$ are linear equations, then minimizing the sum

requires solving a linear program, which, compared to inference in other PL tools such as MLNs, can be done relatively efficiently using well-established techniques. In case $p = 2$, MPE inference can be shown to be a second-order cone program (SOCP) (Kimmig et al., 2012).

2.4 Tasks

We evaluate our semantic representation on the following three tasks, RTE, STS and QA.

2.4.1 Recognizing Textual Entailment (RTE)

RTE is the task of determining whether one natural language text, the *Text T*, *entails*, *contradicts*, or is not related (*neutral*) to another, the *Hypothesis H* (Dagan et al., 2013). "Entailment" here does not mean logical entailment: The Hypothesis is entailed if a human annotator judges that it plausibly follows from the Text. When using naturally occurring sentences, this is a very challenging task that should be able to utilize the unique strengths of both logic-based and distributional semantics. Here are examples from the SICK dataset (Marelli, Menini, Baroni, Bentivogli, Bernardi, & Zamparelli, 2014):

- Entailment

T: A man and a woman are walking together through the woods.

H: A man and a woman are walking through a wooded area.

- Contradiction

T: Nobody is playing the guitar

H: A man is playing the guitar

- Neutral

T: A young girl is dancing

H: A young girl is standing on one leg

The SICK ("Sentences Involving Compositional Knowledge") dataset, which we use for evaluation in this paper, was designed to foreground particular linguistic phenomena but to eliminate the need for world knowledge beyond linguistic knowledge. It was constructed from sentences from two image description datasets, ImageFlickr[1] and the SemEval 2012 STS MSR-Video Description data.[2] Randomly selected sentences from these two sources were first simplified to remove some linguistic phenomena that the dataset was not aiming to cover. Then additional sentences were created as variations over these sentences, by paraphrasing, negation, and reordering. RTE pairs were then created that consisted of a simplified original sentence paired with one of the transformed sentences (generated from either the same or a different original sentence). The dataset was collected for the SemEval 2014 competition (an annual workshop that organizes semantic evaluation tasks and competitions) and it consists of 5,000 T/H pairs for training and 5,000 for testing.

[1] `http://nlp.cs.illinois.edu/HockenmaierGroup/data.html`

[2] `http://www.cs.york.ac.uk/semeval-2012/task6/index.php?id=data`

2.4.2 Semantic Textual Similarity (STS)

Semantic Textual Similarity (STS) is the task of judging the similarity of a pair of sentences on a scale from 0 to 5, and was recently introduced as a SemEval task (Agirre et al., 2012). Gold standard scores are averaged over multiple human annotations and systems are evaluated using the Pearson correlation between a system's output and gold standard scores. Here are some examples:

S_1: A man is playing a guitar.

S_2: A woman is playing the guitar.

- Score: 2.75

S_1: A woman is cutting broccoli.

S_2: A woman is slicing broccoli.

- Score: 5.00

S_1: A car is parking.

S_2: A cat is playing.

- Score: 0.00

For experiments, we use the SICK dataset which is used also for RTE experiments. The dataset has annotations for both tasks, RTE and STS.

2.4.3 Open-Domain Question Answering (QA)

There are many variations of the question answering task including answering factoid questions from a database of facts (Berant et al., 2013; Kwiatkowski et al., 2013), answering multiple choice questions about a short story (Richardson et al., 2013) and answering elementary school science questions (Schoenick, Clark, Tafjord, Turney, & Etzioni, 2016). We are interested in open-domain QA where the answers for a question Q can be extracted from a short document D (as in (Richardson et al., 2013)), not from a database of facts nor from all of the text on the web. This form of QA is more difficult than querying a database of facts because we do not have a fixed schema that we learn how to execute queries against. It is also more difficult than trying to answer a question from all the text in the web because we do not have the huge redundancy of representing the same piece of information that the web provides. In our case, answering the question requires accurate understanding of the document and the question.

The QA dataset we use is a large automatically collected dataset from CNN and Daily Mail (Hermann et al., 2015). Each news article in CNN and Daily Mail has a short summary in the form of a few bullet sentences. The articles and the summary are tokenized, lower cased and processed by a named entity recognizer and coreference resolution tool to find named entities (person, location, organization ...) in the document. Questions are constructed from summary sentences that contain named entities by replacing the named entity with a placeholder, and the task becomes finding the appropriate named entity

	CNN	Daily Mail
Train	380,298	879,450
Dev	3,924	64,835
Test	3,198	53,182

Table 2.1: Size of the QA dataset

from the news article to fill in the placeholder. In addition, all named entities are anonymized to prevent answering the questions from statistical knowledge about the entities and make it necessary to answer the question from the specified document. Table 2.1 lists the sizes of each section of the dataset. Here is a snippet from a document followed by the question:

Document: *@entity26 and @entity9 have spoken about the possibility of putting boots on the ground . the @entity33 is expected to give its official blessing to @entity28 on saturday , which could clear the way for a ground invasion , @entity4 's @entity48 reported . but a few member nations, such as @entity55 majority @entity53 or possibly @entity56 , could give military action a thumbs down . though the @entity26 kingdom has taken the lead with some 100 warplanes , the coalition partners include the @entity63 , @entity64 , @entity65 , @entity66 , @entity67 , @entity68 , @entity69 and @entity9 .*

Question: *@placeholder blessing of military action may set the stage for a ground invasion*

Answer: *@entity33*

This dataset has attracted a lot of attention and a several attempts to solve. The most notable is the recent work by Chen, Bolton, and Manning

(2016). They manually examined 100 examples to find the type of inferences required to answer them. They found that 13% of questions are exact matches of sentences in the document, 41% are paraphrases, 19% can be answered with a partial clue (can not infer the whole question but can infer enough of it to answer the question correctly), only 2% of the questions need an inference that involves multiple sentences, 8% can not be answered because of a coreference error in the dataset and 17% even humans can not answer confidently because of ambiguity or it is hard to find the answer. This analysis suggests that the best achievable performance is around 75%, and almost all of the questions can be answered by processing a single sentence. They also showed that an entity classifier with a few simple features can achieve high performance (around 68%) then presented a neural network approach that achieves 72.4% accuracy which is very close to the best achievable performance.

Despite this analysis and results, we believe this dataset is a reasonable testbed for our system. It helps us develop and evaluate our system on a larger more diverse and more natural dataset than the RTE and STS datasets.

2.5 Related Work

There has recently been several other attempts to integrate logical and distributional information, and we discuss some of them below. Lewis and Steedman (2013) use clustering on distributional data to infer word senses, and perform standard first-order inference on the resulting logical forms. The main difference between this approach and our proposed one lies in the role of

gradience. In Lewis and Steedman (2013), the uncertain knowledge is used to find the clusters and does not propagate to inference, while in our work the uncertain lexical knowledge propagates to the inference step.

Tian, Miyao, and Takuya (2014) represent sentences using Dependency-based Compositional Semantics (Liang, Jordan, & Klein, 2011). They construct phrasal entailment rules based on a logic-based alignment, and use distributional similarity of aligned words to filter rules that do not surpass a given threshold.

Also related to our work the work on distributional models where the dimensions of the vectors encode model-theoretic structures rather than observed co-occurrences (Clark, 2012; Sadrzadeh, Clark, & Coecke, 2013; Grefenstette, 2013; Herbelot & Vecchi, 2015), even though they are not strictly hybrid systems as they do not include contextual distributional information. Grefenstette (2013) represents logical constructs using vectors and tensors, but concludes that they do not adequately capture logical structure, in particular quantifiers.

Chapter 3

Logical Form

The first component of our semantic representation is the logical form. This chapter addresses two main question, how we translate natural sentences to logical form and how we adapt the resulting logical form for PL. All of the work presented in this chapter has been published in (Beltagy & Erk, 2015).

3.1 Chapter Overview

Logical form is the the basic representation we use which can be easily interpreted and manipulated. This chapter discusses two points regarding logical forms. The first point it addresses is the translation from natural sentences to logical form. By default, we use Boxer, which is a rule-based system that translates CCG parsed sentences to logical form. We also tried translating dependency parse trees to logical forms that are suitable for the QA task, but we leave the discussion of this approach to section 7.2.

The second point this chapter addresses is the adaptation of logical form to PL. In our framework, we can view each language understanding task as consisting of a *text* T and a *hypothesis* H, along with a *knowledge base* KB. The text T describes some situation or setting, and the hypothesis H

in the simplest case asks whether a particular statement is true in the situation described in T. The knowledge base KB encodes relevant background knowledge: lexical knowledge, world knowledge, or both. Formally, this is checking if the text and knowledge base entail the hypothesis: $T \wedge KB \Rightarrow H$. In the beginning of chapters 5, 6 and 7, we show how all three tasks can be represented as instances of this basic entailment.

In PL, the logical entailment $T \wedge KB \Rightarrow H$ is equivalent to calculating the probability of the hypothesis given the text and the knowledge base $P(H|T, KB, W_{T,H})$, where $W_{T,H}$ is the world configurations for a particular T, H pair (this is the more detailed formulation of $P(H|T, KB)$ mentioned in the introduction. We use the detailed on in this chapter only). A world configuration for a pair T, H is setting the domain size and prior probabilities of ground atoms of the PL program for an accurate representation of T and H. Practical PL frameworks usually make assumptions like the assumption of a finite domain, which changes how quantifiers and negation behave in PL. This chapter discusses how to adapt the logical form and set the world configurations $W_{T,H}$ to take PL assumptions into account. It also evaluates these adaptations on three entailment datasets.

3.2 Translating text to logical form

The default tool we use to translate text to logical form is Boxer (Bos, 2008), a rule-based semantic analysis system that translates a CCG parse into

a logical form. The formula

$$\exists x, y, z.\, ogre(x) \wedge agent(y, x) \wedge love(y) \wedge patient(y, z) \wedge princess(z) \quad (3.1)$$

is an example of Boxer producing discourse representation structures using a neo-Davidsonian framework. We call Boxer's output alone an "uninterpreted logical form" because the predicate symbols are simply words and do not have meaning by themselves. Their semantics derives from the knowledge base KB we build in Chapter 4.

Translating dependency parses to logical forms We have a rule-based approach to translate dependency parses to logical forms that are more robust but less expressive than what we get from Boxer. We only use this logical form for the QA task, so we leave the detailed discussion to section 7.2.

3.3 Adapting logical form to PL

Our natural language understanding tasks can be all solved as functions of the more basic operation of checking if the text and knowledge base entail the hypothesis: $T \wedge KB \Rightarrow H$, or probabilistically, calculating the probability $P(H|T, KB, W_{T,H})$. PL frameworks usually make assumptions that make the probabilistic version different from the standard first logic formulation, even with rules in KB are all hard (have infinite weights). One practical assumption is the DCA, or the assumption that the domain has a fixed size. This assumption has implications on how quantifiers and negations work. This section discusses how to adapt logical form to take the DCA assumption into

account. It addresses two points, how to get quantifiers to work while assuming a fixed-sized domain, and how to set prior probabilities for the negations to work as expected. This is done through setting the world configurations $W_{T,H}$, which encompasses number of constants in the domain (domain size) and prior probability of each ground atom. The world configurations $W_{T,H}$ is a function of T and H because it is set based on the quantifiers and negations of T and H.

3.3.1 Using a Fixed Domain Size

PL frameworks compute a probability distribution over possible worlds, as described in section 2.3. When we describe a task as a text T and a hypothesis H, the worlds over which the PL computes a probability distribution are "mini-worlds", just large enough to describe the situation or setting given by T. The probability $P(H|T, KB, W_{T,H})$ then describes the probability that H would hold given the probability distribution over the worlds that possibly describe T. The use of "mini-worlds" is by necessity, as most practical PL frameworks can only handle worlds with a fixed domain size, where "domain size" is the number of constants in the domain.

Formally, the influence of the set of constants on the worlds considered by a PL framework can be described by the Domain Closure Assumption (DCA, (Genesereth & Nilsson, 1987; Richardson & Domingos, 2006)): The only models considered for a set F of formulas are those for which the following three conditions hold: (a) Different constants refer to different objects in the

domain, (b) the only objects in the domain are those that can be represented using the constant and function symbols in F, and (c) for each function f appearing in F, the value of f applied to every possible tuple of arguments is known, and is a constant appearing in F. Together, these three conditions entail that *there is a one-to-one relation between objects in the domain and the named constants of* F. When the set of all constants is known, it can be used to ground predicates to generate the set of all ground atoms, which then become the nodes in the graphical model. Different constant sets result in different graphical models. If no constants are explicitly introduced, the graphical model is empty (no random variables).

This means that to obtain an adequate representation of an inference problem consisting of a text T and hypothesis H, we need to introduce a sufficient number of constants explicitly into the formula: The worlds that the PL considers need to have enough constants to faithfully represent the situation in T and not give the wrong entailment for the hypothesis H. In what follows, we explain how we determine an appropriate set of constants for the logical-form representations of T and H. The domain size that we determine is one of the two components of the parameter $W_{T,H}$.

3.3.1.1 Skolemization

We introduce some of the necessary constants through the well-known technique of *Skolemization* (Skolem, 1920). It transforms a formula of the form $\forall x_1 \ldots x_n \exists y.F$ to $\forall x_1 \ldots x_n.F^*$, where F^* is formed from F by replacing all

free occurrences of y in F by a term $f(x_1, \ldots, x_n)$ for a new function symbol f. If $n = 0$, f is called a *Skolem constant*, otherwise a *Skolem function*. Although Skolemization is a widely used technique in first-order logic, it is not frequently employed in PL since many applications do not require existential quantifiers.

We use Skolemization on the text T (but not the hypothesis H, as we cannot assume a priori that it is true). For example, the logical expression in Equation 3.1, which represents the sentence *T: An ogre loves a princess*, will be Skolemized to:

$$ogre(O) \wedge agent(L, O) \wedge love(L) \wedge patient(L, N) \wedge princess(N) \qquad (3.2)$$

where O, L, N are Skolem constants introduced into the domain.

Standard Skolemization transforms existential quantifiers embedded under universal quantifiers to Skolem functions. For example, for the text *T: All ogres snore* and its logical form $\forall x.\, ogre(x) \Rightarrow \exists y.\, agent(y, x) \wedge snore(y)$ the standard Skolemization is $\forall x. ogre(x) \Rightarrow agent(f(x), x) \wedge snore(f(x))$. Per condition (c) of the DCA above, if a Skolem function appeared in a formula, we would have to know its value for any constant in the domain, and this value would have to be another constant. To achieve this, we introduce a new predicate $Skolem_f$ instead of each Skolem function f, and for every constant that is an *ogre*, we add an extra constant that is a *loving* event. The example above then becomes:

$$T : \forall x.\, ogre(x) \Rightarrow \forall y.\, Skolem_f(x, y) \Rightarrow agent(y, x) \wedge snore(y)$$

If the domain contains a single *ogre* O_1, then we introduce a new constant C_1 and an atom $Skolem_f(O_1, C_1)$ to state that the Skolem function f maps the constant O_1 to the constant C_1.

3.3.1.2 Existence

But how would the domain contain an *ogre* O_1 in the case of the text *T: All ogres snore*, $\forall x.ogre(x) \Rightarrow \exists y.agent(y, x) \wedge snore(y)$? Skolemization does not introduce any variables for the universally quantified x. We still introduce a constant O_1 that is an *ogre*. This can be justified by pragmatics since the sentence presupposes that there are, in fact, *ogres* (Strawson, 1950; Geurts, 2007). We use the sentence's parse to identify the universal quantifier's restrictor and body, then introduce entities representing the restrictor of the quantifier. The sentence *T: All ogres snore* effectively changes to *T: All ogres snore, and there is an ogre.* At this point, Skolemization takes over to generate a constant that is an *ogre*. Sentences like *T: There are no ogres* is a special case: For such sentences, we do not generate evidence of an *ogre*. In this case, the non-emptiness of the domain is not assumed because the sentence explicitly negates it.

3.3.1.3 Universal quantifiers in the hypothesis

The most serious problem with the DCA is that it affects the behavior of universal quantifiers in the hypothesis. Suppose we know that *T: Shrek is a green ogre*, represented with Skolemization as $ogre(SH) \wedge green(SH)$. Then

we can conclude that *H: All ogres are green*, because by the DCA we are only considering models with this single constant which we know is both an *ogre* and *green*. To address this problem, we again introduce new constants.

We want a hypothesis *H: All ogres are green* to be judged true iff there is evidence that all *ogres* will be *green*, no matter how many *ogres* there are in the domain. So H should follow from T_2*: All ogres are green* but not from T_1*: There is a green ogre*. Therefore we introduce a new constant G for the hypothesis and assert $ogre(G)$ to test if we can then conclude that $green(G)$. The new evidence $ogre(G)$ prevents the hypothesis from being judged true given T_1. Given T_2, the new *ogre* G will be inferred to be *green*, in which case we take the hypothesis to be true. Again, with a hypothesis such as *H: There are no ogres*, we do not generate any evidence for the existence of an *ogre*.

3.3.2 Setting Prior Probabilities

The second adaptation of the logical form to PL is finding out how to set the prior probabilities of ground atoms. Suppose we have an empty text T, and the hypothesis *H: A is an ogre*, where A is a constant in the system. Without any additional information, the worlds in which $ogre(A)$ is true are going to be as likely as the worlds in which the ground atom is false, so $ogre(A)$ will have a probability of 0.5. So without any text T, ground atoms have a prior probability in PL that is not zero. This prior probability depends mostly on the size of the set R of input formulas. The prior probability of an individual ground atom can be influenced by a weighted rule, for example

$ogre(A) \mid -3$, with a negative weight, sets a low prior probability on A being an ogre. This is the second group of parameters that we encode in $W_{T,H}$: weights on ground atoms to be used to set prior probabilities.

Prior probabilities are problematic for our probabilistic encoding of natural language understanding problems. As a reminder, we probabilistically test for entailment by computing the probability of the hypothesis given the text, or more precisely $P(H|T, KB, W_{T,H})$. However, how useful this conditional probability is as an indication of entailment depends on the prior probability of H, $P(H|KB, W_{T,H})$. For example, if H has a high prior probability, then a high conditional probability $P(H|T, KB, W_{T,H})$ does not add much information because it is not clear if the probability is high because T really entails H, or because of the high prior probability of H. In practical terms, we would not want to say that we can conclude from *T: All princesses snore* that *H: There is an ogre* just because of a high prior probability for the existence of ogres.

We discuss two suggestions on how to solve this problem and make the probability $P(H|T, KB, W_{T,H})$ less sensitive to $P(H|KB, W_{T,H})$. The first is to use the ratio between the conditional and the prior probability of the hypothesis, $\frac{P(H|T,KB,W_{T,H})}{P(H|KB,W_{T,H})}$, with the intuition that the absolute value of $P(H|T, KB, W_{T,H})$ does not really matter, but what matters is how much adding T changes the probability of H positively (indicating entailment) or negatively (indicating contradiction). We also discuss the relation between this ration and mutual information. The second option is to pick a par-

ticular $W_{T,H}$ such that the prior probability of H is approximately zero, $P(H|KB, W_{T,H}) \approx 0$, so that we know that any increase in the conditional probability is an effect of adding T.

For the rest of this section, we argue why we believe the first option is not a good fit for natural language understanding tasks we are interested in, while the second is a better fit. Then we show how to set the world configurations $W_{T,H}$ such that $P(H|KB, W_{T,H}) \approx 0$ by enforcing the closed-world assumption (CWA). This is the assumption that all ground atoms have very low prior probability (or are false by default).

3.3.2.1 Problems with the ratio

The first problem with the ratio approach is that its motivation does not fit our intuitions of when a hypothesis should be entailed by a text. For example for *T: An ogre loves a princess*, and *H: An ogre loves a green princess*, T should not be entailing H because there is no evidence that the *princess* is *green*. However, the probability of H conditioned on T increases dramatically compared to the prior probability of H because T has evidence for a large part of H. So this means that a high ratio is not always an indication of entailment.

There are also cases of entailment with a not very high ratio. Consider for example *T: No ogre snores*, and *H: No ogre snores loudly*. T entails H, and $P(H|T, KB, W_{T,H})$ is greater than $P(H|KB, W_{T,H})$, but not too much greater because $P(H|KB, W_{T,H})$ is already a high value. Together, these two examples should demonstrate that the idea behind taking the ratio, that the

degree to which conditioning on T changes the probability of H is an indicator of degree of entailment, does not really fit the intuition of when a hypothesis should be entailed.

The last problem with the ratio is that it is very sensitive to the problem size (length of T and H). That is, entailing pairs of different sizes have different ratios. This makes reasoning with the ratio tricky. It could be possible to normalize the ratios given the problem size, but we did not explore this direction.

Ratio and Mutual Information Mutual information (MI) is a measure of dependence between two random variables. In our setting, it could be possible to use the mutual information between T and H to signal entailment. Mutual information is denoted by:

$$MI(H,T) = \sum_{h \in H} \sum_{t \in T} P(h,t) \log \frac{P(h,t)}{P(h)P(t)}$$

However, this formulation of mutual information is not appropriate for entailment because it is symmetric, while entailment is asymmetric. The other issue with this formulation is practicality considerations of calculating the joint probability $P(H,T)$ in PL. Remember that T and H are PL formulas not individual ground atoms. PL inference requires the explicit introduction of domain constants, and our adaptations in section 3.3.1 shows how to introduce these constants for a conditional probability $P(H \mid T)$, but it is not clear how to introduce these constants for the joint probability $P(H,T)$.

The MI equation can be rearranged to use conditional probabilities instead of joints, and include the world configurations $W_{T,H}$ which are set following section 3.3.1. The reformulation is:

$$MI(H,T) = \sum_{h \in H} \sum_{t \in T} P(h|t, W_{T,H}) P(t, W_{T,H}) \log \frac{P(h|t, W_{T,H})}{P(h, W_{T,H})}$$

Remember that $W_{T,H}$ is set such that T is True. As a result, this formula can be simplified to:

$$MI(H,T) = \sum_{h \in H} P(h|T, W_{T,H}) \log \frac{P(h|T, W_{T,H})}{P(h, W_{T,H})}$$

The ratio between the prior probability and conditional probability of H is a part of the calculation of the MI and it inherits the issues with the ratio discussed above. This makes MI another solution that does not work for entailment.

3.3.2.2 Using the CWA to set the prior probability of the hypothesis

The closed-world assumption (CWA) is the assumption that everything is false unless stated otherwise. We translate it to our probabilistic setting as saying that all ground atoms have very low prior probability. For most queries H, setting the world configuration $W_{T,H}$ such that all ground atoms have low prior probability is enough to achieve that $P(H|KB, W_{T,H}) \approx 0$ (not for negated Hs, and this case is discussed below). For example, *H: An ogre loves a princess*, in logic is:

$$H : \exists x, y, z.\, ogre(x) \wedge agent(y, x) \wedge love(y) \wedge patient(y, z) \wedge princess(z)$$

Having low prior probability on all ground atoms means that the prior probability of this existentially quantified H is close to zero.

We believe that this setup is more appropriate for probabilistic natural language entailment for the following reasons. First, this aligns with our intuition of what it means for a hypothesis to follow from a text: that H should be entailed by T not because of general world knowledge. For example, if *T: An ogre loves a princess*, and *H: Texas is in the USA*, then although H is true in the real world, T does not entail H. Another example: *T: An ogre loves a princess*, *H: An ogre loves a green princess*, again, T does not entail H because there is no evidence that the *princess* is *green*, in other words, the ground atom $green(N)$ has very low prior probability. As we said above, we construct the worlds over which PL reasons to encode the situation or setting described by T. Anything that is not explicitly stated in T should be assumed to be false by default. In the RTE task, this is an explicit part of the task specification. In sentence similarity (STS), we would not want a known fact like *Texas is in the USA* to be judged similar to every sentence. In question answering (QA), a text should only count as an answer to a query if it actually addresses the query.

The second reason is that with the CWA, the inference result is less sensitive to the domain size (number of constants in the domain). In logical forms for typical natural language sentences, most variables in the hypothesis are existentially quantified. Without the CWA, the probability of an existentially quantified hypothesis increases as the domain size increases, regardless

of the text. This makes sense in the PL setting, because in larger domains the probability that something exists increases. However, this is not what we need for testing natural language queries, as the probability of the hypothesis should depend on T and KB, not the domain size. With the CWA, what affects the probability of H is the non-zero evidence that T provides and KB, regardless of the domain size.

The third reason is computational efficiency. As discussed in Section 2.3, PL first compute all possible groundings of a given set of weighted formulas which can require significant amounts of memory. This is particularly striking for problems in natural language semantics because of long formulas. We show how to utilize the CWA to address this problem by reducing the number of ground atoms that the system generates. We discuss the details in section 5.5.

3.3.2.3 Setting the prior probability of negated H

While using the CWA is enough to set $P(H|KB, W_{T,H}) \approx 0$ for most Hs, it does not work for *negated* H (negation is part of H). Assuming that everything is false by default and that all ground atoms have very low prior probability (CWA) means that all negated queries H are true by default. The result is that all negated H are judged entailed regardless of T. For example, *T: An ogre loves a princess* would entail *H: No ogre snores.* This H in logic is:

$$H : \forall x, y.\, ogre(x) \Rightarrow \neg(agent(y, x) \wedge snore(y))$$

As both x and y are universally quantified variables in H, we generate evidence of an ogre $ogre(O)$ as described in section 3.3.1. Because of the CWA, O is assumed to be *does not snore*, and H ends up being true regardless of T.

To set the prior probability of H to ≈ 0 and prevent it from being assumed true when T is just uninformative, we construct a new rule A that implements a kind of anti-CWA. A is formed as a conjunction of all the predicates that were not used to generate evidence before, and are *negated* in H. This rule A gets a positive weight indicating that its ground atoms have high prior probability. As the rule A together with the evidence generated from H states the opposite of the negated parts of H, the prior probability of H is low, and H cannot become true unless T explicitly negates A. T is translated into unweighted rule, which are taken to have infinite weight, and which thus can overcome the finite positive weight of A. Here is a Neutral RTE example, *T: An ogre loves a princess*, and *H: No ogre snores.* Their representations are:

T: $\exists x, y, z.\, ogre(x) \wedge agent(y, x) \wedge love(y) \wedge patient(y, z) \wedge princess(z)$

H: $\forall x, y.\, ogre(x) \Rightarrow \neg(agent(y, x) \wedge snore(y))$

E: $ogre(O)$

A: $agent(S, O) \wedge snore(S)|w = 1.5$

E is the evidence generated for the universally quantified variables in H, and A is the weighted rule for the remaining negated predicates. The relation between T and H is Neutral, as T does not entail H. This means, we want

$P(H|T, KB, W_{T,H}) \approx 0$, but because of the CWA, $P(H|T, KB, W_{T,H}) \approx 1$. Adding A solves this problem and $P(H|T, A, KB, W_{T,H}) \approx 0$ because H is not explicitly entailed by T.

In case H has existentially quantified variables that occur in negated predicates, they need to be universally quantified in A for H to have a low prior probability. For example, *H: There is an ogre that is not green*:

$$H : \exists x.\, ogre(x) \wedge \neg green(x)$$

$$A : \forall x.\, green(x) | w = 1.5$$

If one variable is universally quantified and the other is existentially quantified, we need to do something more complex. Here is an example, *H: An ogre does not snore*:

$$H : \exists x.\, ogre(x) \wedge \neg(\, \exists y.\, agent(y, x) \wedge snore(y)\,)$$

$$A : \forall v.\, agent(S, v) \wedge snore(S) | w = 1.5$$

Notes about how inference proceeds with the rule A added If H is a negated formula that is entailed by T, then T (which has infinite weight) will contradict A, allowing H to be true. Any weighted inference rules in the knowledge base KB will need weights high enough to overcome A. So the weight of A is taken into account when computing inference rule weights.

In addition, adding the rule A introduces constants in the domain that are necessary for making the inference. For example, take *T: No monster snores*, and *H: No ogre snores*, which in logic are:

T: $\neg\exists x, y.\, monster(x) \wedge agent(y, x) \wedge snore(y)$

H: $\neg\exists x, y.\, ogre(x) \wedge agent(y, x) \wedge snore(y)$

A: $ogre(O) \wedge agent(S, O) \wedge snore(S)|w = 1.5$

KB: $\forall x.\, ogre(x) \Rightarrow monster(x)$

Without the constants O and S added by the rule A, the domain would have been empty and the inference output would have been wrong. The rule A prevents this problem. In addition, the introduced evidence in A fits the idea of "evidence propagation" mentioned above, (detailed in Section 5.5.2). For entailing sentences that are negated, like in the example above, the evidence propagates from H to T (not from T to H as in non-negated examples). In the example, the rule A introduces an evidence for $ogre(O)$ that then propagates from the LHS to the RHS of the KB rule.

3.3.3 Evaluation

We evaluate the proposed adaptations on three entailment datasets. The first is a synthetic dataset that exhaustively tests inference performance on sentences with two quantifiers. The second is the RTE part of the SICK dataset (Marelli et al., 2014). The third is FraCas (Cooper, Crouch, Van Eijck, Fox, Van Genabith, Jaspars, Kamp, Milward, Pinkal, Poesio, et al., 1996).

3.3.3.1 Synthetic dataset

We automatically generate an entailment dataset that exhaustively test inferences on sentences with two quantifiers. Each entailment pair (T, H) is

generated following this format:

$$T : Q_{t1}(L_{t1}, Q_{t2}(L_{t2}, R_{t2}))$$

$$H : Q_{h1}(L_{h1}, Q_{h2}(L_{h2}, R_{h2}))$$

where

- $Q_x \in$ {some, all, no, not all}
- $L_{t1}, L_{h1} \in$ {man, hungry man} and $L_{t1} \neq L_{h1}$
- $L_{t2}, L_{h2} \in$ {food, delicious food} and $L_{t2} \neq L_{h2}$
- $R_{t2}, R_{h2} =$ eat

Informally, the dataset has all possible combinations of sentences with two quantifiers. Also it has all possible combinations of monotonicity directions – upward and downward – between L_{t1} and L_{h1} and between L_{t2} and L_{h2}. The dataset size is 1,024 RTE pairs. Here is an example of a generated RTE pair:

T: No man eats all food

H: Some hungry men eat not all delicious food

The dataset is automatically annotated for entailment decisions by normalizing the logical forms of the sentences and then using standard monotonicity rules on the bodies and restrictors of the quantifiers. 72 pairs out of the 1,024 are entailing pairs, and the rest are non-entailing.

System	Accuracy	False positives	False negatives
Baseline (most common)	92.96%	0	72
Skolem	50.78%	472	32
Skolem + Existence	57.03%	440	0
Skolem + ($\forall$ in H)	82.42%	140	40
Skolem + ($\forall$ in H) + Prior	96.09%	0	40
Full system	**100%**	0	0

Table 3.1: Results of the Synthetic dataset on different configurations of the system. The most common class is Non-entail. False positives and False negatives results are counts out of 1,024

Our system computes $P(H|T, KB, W_{T,H})$. The resulting probability between 0 and 1 needs to be mapped to an Entail/Non-entail decision. In this dataset, and because we do not have weighted inference rules, all output probabilities greater than 0.9 denote Entail and probabilities less than 0.0 denote Non-entail.

Results Table 3.1 summarizes the results on the synthetic dataset in terms of accuracy. The baseline always judges non-entailment. Ablation tests are as follows. *Skolem* is a system that applies Skolemization to existentially quantified variables in the Text T (section 3.3.1.1) but none of the other adaptations. *Existence* is a system that makes the existence assumption for universal quantifiers in T (section 3.3.1.2). *($\forall$ in H)* is constant introduction for universal quantifiers in the hypothesis (section 3.3.1.3). Finally, *Prior* is a system that handles negation by setting the prior of ground atoms as discussed above in section 3.3.2. The results in table 3.1 show the importance of each part of

System	Accuracy
Baseline (most common)	56.36%
Skolem	68.10%
Skolem + Existence	68.10%
Skolem + ($\forall$ in H)	68.14%
Skolem + ($\forall$ in H) + Prior	76.48%
Full system	**76.52%**

Table 3.2: Results of the SICK dataset. The most common class is Neutral

the proposed system. Skolemization and the Existence assumption eliminate some false negatives from missing constants. All false positives are eliminated when constants are introduced for universal quantifiers in the Hypothesis ($\forall$ in H) and when the priors are set to handle negation. The full system achieves 100% accuracy, showing that our formulation is perfectly adapting the logical form for PL on these complex quantified sentences.

3.3.3.2 The SICK dataset

Pairs in SICK (as well as FraCas in the next section) are classified into three classes, Entailment, Contradiction, and Neutral. As we explain in section 5.2, we do this threeway classification using the two inferences: $P(H|T, KB, W_{T,H})$ and $P(\neg H|T, KB, W_{T,\neg H})$.

Results Table 3.2 reports results on the SICK dataset, again in terms of accuracy. Almost all sentences in the SICK dataset are simple existentially quantified sentences except for a few sentences with an outer negation. Accordingly, the system with Skolemization basically achieves the same accuracy

as when Existence and ($\forall$ in H) are added. Handling negation in H effectively improves the accuracy of our system by reducing the number of false positives.

3.3.3.3 The FraCas dataset

FraCas (Cooper et al., 1996)[1] is a dataset of hand-built RTE pairs. The dataset consists of 9 sections, each of which is testing a different set of phenomena. For this work, we use sentences from the first section, which tests quantification and monotonicity. However, we exclude pairs containing the determiners "few", "most", "many" and "at least" because our system does not currently have a representation for them. We evaluate on 46 pairs out of 74.[2][3] Because of that, we cannot compare with previous systems that evaluate on the whole section (MacCartney & Manning, 2008; Lewis & Steedman, 2013).

To map sentences to logical form, we use Boxer as discussed above. By default, Boxer relies on C&C (Curran, Clark, & Bos, 2009) to get the CCG parses of the sentences. Instead, we run Boxer on CCG parses produced by EasyCCG (Lewis & Steedman, 2014) because it is more accurate on FraCas. Like Lewis and Steedman (2013), we additionally test on gold-standard parses to be able to evaluate our technique of handling quantifiers in the absence of parser errors.

[1] We use the version by MacCartney and Manning (2007)

[2] The first section consists of 80 pairs, but like MacCartney and Manning (2007) we ignore the pairs with an undefined result.

[3] The gold standard annotation for pair number 69 should be Neutral not Entail. We changed it accordingly.

System	Gold parses	System parses
Baseline (most common)	47.82%	47.82%
Skolem	50.00%	43.48%
Skolem + Existence	43.48%	36.96%
Skolem + ($\forall$ in H)	63.04%	50.00%
Skolem + ($\forall$ in H) + Prior	100.0%	84.78%
Full system	**100.0%**	84.78%

Table 3.3: Results of the FraCas dataset. The most common Entail. Accuracies are reported for gold parses and parses from a CCG parser.

For multi-sentence examples, we add a simple co-reference resolution step that connects definite NPs across sentences. For example, *the right to live in Europe* in T1 and T3 should corefer in the following example:

T1: Every European has the right to live in Europe

T2: Every European is a person

T3: Every person who has the right to live in Europe can travel freely within Europe

We also added two rules encoding lexical knowledge, paraphrased as "a lot of $x \Rightarrow x$" and "one of $x \Rightarrow x$" to handle one of the examples, as lexical coverage is not the focus of our analysis.

Results Table 3.3 summarizes the results of our system for gold parses and system parses. We see that the Existence assumption is not needed in this dataset because it is constructed to test semantics and not presupposition. Results with Existence are lower because without Existence, three cases (the previous example is one of them) are correctly classified as Entail, but with

Existence they are classified as Neutral. Without Existence the domain is empty, and $P(\neg T|H) = 0$ because $\neg T$, which is existentially quantified, is trivially false. With Existence added, $P(\neg T|H) = 1$ because the domain is not empty, and the CWA is not handled. Also we see that, as in the two previous experiments, setting the priors appropriately has the biggest impact. With all components of the system added, and with gold parses, we get 100% accuracy. With system parses, all results are lower, but the relative scores for the different subsystems are comparable to the gold parse case.

3.4 Chapter Summary

This chapter discussed the logical form component of our semantic representation. We discussed how we translate text to logical form using a CCG parse and Boxer.

The natural language understanding tasks we are interested in can all be solved using the more basic probabilistic entailment $P(H|T, KB, W_{T,H})$ which we construct and calculate using PL. However, the logical form need to be adapted to PL because of the assumption of fixed domain size. We showed how to set the domain size and instantiate enough constants in the domain for the universal quantifiers to work properly. We also showed how to set the prior probability of each ground atom for negation to work as expected. Finally, we evaluated the logical form adaptations on three datasets and we show their effectiveness.

Chapter 4

Knowledge Base

The second component of our semantic representation is the knowledge base. This chapter addresses the question of the automatic collection of a weighted knowledge base from various linguistic resources. We will evaluate the contributions of this chapter in later chapters that are dedicated to individual tasks. All of the work presented in this chapter has been published in (Beltagy et al., 2016)

4.1 Chapter Overview

Natural language understanding usually requires reasoning about the relations between words and short phrases. This is called the lexical entailment task; the task of finding if a lexical term (a word or a short phrase) entails another (Kotlerman, Dagan, Szpektor, & Zhitomirsky-Geffet, 2010). Another equally important question in language understanding is finding out which pairs or lexical terms are relevant to a given inference. This latter is the focus of this chapter.

In this chapter, we are given two pieces of text, a text T and a hypothesis H representing a language understanding task, and the goal is to collect a

knowledge base KB that contains the lexical information required to find the relation between T and H. The KB is encoded as a set of "weighted" logical rules.

Our knowledge base has a set of rules from pre-existing rule collections, and we add all possibly matching rules to the pair T and H (Section 4.2). The main limitation with existing collections or rules is that they are never complete, and we always need more rules that do not exist in these collections. The second group of rules are "on-the-fly" rules that are generated for a particular text/hypothesis pair. We align the text and hypothesis and use the alignment to determine additionally useful inference rules. Our alignment procedure aligns the logical form of T and H not the textual T and H. It tries to find an alignment between the "entities" in T and the entities in H. Rules can be true or false, so we still need to weight them using a lexical entailment tool [1]. For example,

T: A kid is playing guitar

H: A woman is playing a musical instrument

Our alignment procedure will find that the relevant lexical rules are

r_1: kid $\Rightarrow$ woman $\mid w_1$

r_2: guitar $\Rightarrow$ musical instrument $\mid w_2$

[1] The lexical entailment classifier is the work of Stephen Roller and Pengxiang Cheng and its details are presented in the paper (Beltagy et al., 2016).

Then the lexical entailment tool should give the first rule a low weight while giving the second rule a high weight. We use the rules extracted from the training set as training data from the lexical entailment tool. This can be roughly viewed as transforming the full entailment task into multiple smaller lexical entitlements, then rely on PL to combine these little lexical entailment decisions into the overall entailment decision.

The first alignment procedure is more suited for short text as in RTE and STS, and it uses a variant of Robinson resolution (Robinson, 1965). The second alignment algorithm is a graph-based alignment that it is more suitable for the QA task which has long text. For readability, we leave the discussion of the graph-based alignment to the QA chapter (section 7.3.1).

We will evaluate the contributions of this chapter; the precompiled rules and the Robinson resolution alignment in the context of the RTE task (chapter 5).

4.2 Precompiled Rules

The first group of rules is collected from existing databases. We collect rules from WordNet (Princeton University, 2010) and the paraphrase collection PPDB (Ganitkevitch et al., 2013). We use simple string matching to find the set of rules that are relevant to a given text/hypothesis pair T and H. If the left-hand side of a rule is a substring of T and the right-hand is a substring of H, the rule is added, and likewise for rules with LHS in H and RHS in T. Rules that go from H to T are important in case T and H are negated, e.g.

T: *No ogre likes a princess*, H: *No ogre loves a princess*. The rule needed is *love* $\Rightarrow$ *like* which goes from H to T.

4.2.1 WordNet

WordNet (Princeton University, 2010) is a lexical database of words grouped into sets of synonyms. In addition to grouping synonyms, it lists semantic relations connecting groups. We represent the information on WordNet as "hard" logical rules. The semantic relations we use are:

- Synonymy: $\forall x.\, man(x) \Leftrightarrow guy(x)$
- Hypernymy: $\forall x.\, car(x) \Rightarrow vehicle(x)$
- Antonymy: $\forall x.\, tall(x) \Leftrightarrow \neg short(x)$

One advantage of using logic is that it is a powerful representation that can effectively represent these different semantic relations.

4.2.2 Paraphrase collections

Paraphrase collections are precompiled sets of rules, e.g: *a person riding a bike* $\Rightarrow$ *a biker*. We translate paraphrase collections, in this case PPDB (Ganitkevitch et al., 2013), to logical rules. We use the Lexical, One-To-Many and Phrasal sections of the XL version of PPDB.

We use a simple rule-based approach to translate natural-language rules to logic. First, we can make the assumption that the translation of a PPDB

rule is going to be a conjunction of positive atoms. PPDB does contain some rules that are centrally about negation, such as *deselected* $\Rightarrow$ *not selected*, but we skip those as the logical form analysis already handles negation. As always, we want to include in KB only rules pertaining to a particular text/hypothesis pair T and H. Say $LHS \Rightarrow RHS$ is a rule such that LHS is a substring of T and RHS is a substring of H. Then each word in LHS gets represented by a unary predicate applied to a variable, and likewise for RHS – note that we can expect the same predicates to appear in the logical forms $L(T)$ and $L(H)$ of the text and hypothesis. For example, if the rule is *a person riding a bike* $\Rightarrow$ *a biker*, then we get the atoms $person(p)$, $riding(r)$ and $bike(b)$ for the LHS, with variables p, r, b. We then add Boxer meta-predicates to the logical form for LHS, and likewise for RHS. Say that $L(T)$ includes $person(A) \wedge ride(B) \wedge bike(C) \wedge agent(B, A) \wedge patient(B, C)$ for constants A, B, and C, then we extend the logical form for LHS with $agent(r, p) \wedge patient(r, b)$. We proceed analogously for RHS. This gives us the logical forms: $L(LHS) = person(p) \wedge agent(r, p) \wedge riding(r) \wedge patient(r, b) \wedge bike(b)$ and $L(RHS) = biker(k)$.

The next step is to bind the variables in $L(LHS)$ to those in $L(RHS)$. In the example above, the variable k in the RHS should be matched with the variable p in the LHS. We determine these bindings using a simple rule-based approach: We manually define paraphrase rule *templates* for PPDB, which specify variable bindings. A rule template is conditioned on the part-of-speech tags of the words involved, or an X for variables with no words

involved. In our example it is $N_1V_2N_3 \Rightarrow N_1$, which binds the variables of the first N on the left to the first N on the right, unifying the variables p and k. The final paraphrase rule is: $\forall p, r, b.\, person(p) \wedge agent(r, p) \wedge riding(r) \wedge patient(r, b) \wedge bike(b) \Rightarrow biker(p)$. In case some variables in the RHS remain unbound, they are existentially quantified, e.g.: $\forall p.\, pizza(p) \Rightarrow \exists q.\, slice(p) \wedge of(p, q) \wedge pizza(q)$. Another common case is when a variable does not have an involved word, for example *staring at* X $\Rightarrow$ *looking at* X which has the template $V_1X_2 \Rightarrow V_1X_2$ which in logic will be: $\forall s, x.star(s) \wedge at(s, x) \wedge look(s) \wedge at(s, x)$. Appendix B list all templates with examples. Any rule that does not match one of these templates is discarded because it is usually a result of misparse.

Each PPDB rule comes with a set of similarity scores which we need to map to a single MLN weight. We use the simple log-linear equation suggested by Ganitkevitch et al. (2013) to map the scores into a single value:

$$weight(r) = -\sum_{i=1}^{N} \lambda_i \log \varphi_i \tag{4.1}$$

where, r is the rule, N is number of the similarity scores provided for the rule r, φ_i is the value of the ith score, and λ_i is its scaling factor. For simplicity, following Ganitkevitch et al. (2013), we set all λ_i to 1. To map this weight to a final MLN rule weight, we use the weight-learning method discussed in Section 5.5.3.

4.2.3 Handcoded rules

We also add a few handcoded rules to the KB that we do not get from other resources. We only add several lexical rules where one side of the rule is the word *nobody*, e.g: *nobody* $\Leftrightarrow \neg$ *somebody* and *nobody* $\Leftrightarrow \neg$ *person*. Appendix C lists all handcoded rules.

4.3 On-the-fly rules from Alignment

Existing collections of rules are never complete, and we always need more rules that do not exist in these collections. The second group of rules we generate are "on-the-fly" rules that generated for a particular text/hypothesis pair. A naive way of generating on-the-fly rules would be to generate rules matching any word or short phrase in T with any word or short phrase in H. This includes many unnecessary rules, for example for *T: An ogre loves a princess* and *H: A monster likes a lady*, the system generates rules linking *ogre* to *lady*.

A better way would be to find an *alignment* between words and phrases in T and words or phrases in H guided by the logic, and the rules we need are the rules dictated by the alignment. We have two alignment procedures, one relies on a variant of Robinson resolution (Robinson, 1965), and the other uses graph matching. The first (which we discuss below) is more suitable for short sentences with relatively similar structure (as in the RTE and STS datasets we use for evaluation). The graph-based alignment is more suitable for longer more diverse sentence as in QA, so for readability, we leave its discussion to the

QA chapter (section 7.3.1). Finally, it briefly mentions the lexical entailment classifier we use to weight the rules.

4.3.1 Robinson Resolution Alignment

Robinson resolution is a theorem proving method for testing unsatisfiability that has been used in some previous RTE systems Bos (2009). It assumes a formula in conjunctive normal form (CNF), a conjunction of clauses, where a clause is a disjunction of literals, and a literal is a negated or non-negated atom. More formally, the formula has the form $\forall x_1, \ldots, x_n \big(C_1 \wedge \ldots \wedge C_m\big)$, where C_j is a clause and it has the form $L_1 \vee \ldots \vee L_k$ where L_i is a literal, which is an atom a_i or a negated atom $\neg a_i$. The resolution rule takes two clauses containing complementary literals, and produces a new clause implied by them. Writing a clause C as the set of its literals, we can formulate the rule as:

$$\frac{C_1 \cup \{L_1\} \qquad C_2 \cup \{L_2\}}{(C_1 \cup C_2)_\theta}$$

where θ is a most general unifier of L_1 and $\neg L_2$.

4.3.1.1 Alignment algorithm

In our case, we use a variant of Robinson resolution to remove the parts of text T and hypothesis H that the two sentences have in common. Instead of one set of clauses, we use two: one is the CNF of T, the other is the CNF of $\neg H$. The resolution rule is only applied to pairs of clauses where one clause is from T, the other from H. When no further applications of the

resolution rule are possible, we are left with remainder formulas rT and rH. If rH contains the empty clause, then H follows from T without inference rules. Otherwise, inference rules need to be generated. In the simplest case, we form a single inference rule as follows. All variables occurring in rT or rH are existentially quantified, all constants occurring in rT or rH are un-Skolemized to new universally quantified variables, and we infer the negation of rH from rT. That is, we form the inference rule

$$\forall x_1 \ldots x_n \exists y_1 \ldots y_m.\, rT\overline{\theta} \Rightarrow \neg rH\overline{\theta}$$

where $\{y_1 \ldots y_m\}$ is the set of all variables occurring in rT or rH, $\{a_1, \ldots a_n\}$ is the set of all constants occurring in rT or rH and $\overline{\theta}$ is the inverse of a substitution $\theta : \{a_1 \rightarrow x_1, \ldots, a_n \rightarrow x_n\}$ for distinct variables $x_1, \ldots, x_n$.

For example, consider *T: An ogre loves a princess* and *H: A monster loves a princess*. This gives us the following two clause sets. Note that all existential quantifiers have been eliminated through Skolemization. The hypothesis is negated, so we get five clauses for T but only one for H.

$$\begin{array}{rl} T: & \{ogre(A)\}, \{princess(B)\}, \{love(C\}, \{agent(C, A)\}, \{patient(C, B)\} \\ \neg H: & \{\neg monster(x), \neg princess(y), \neg love(z), \neg agent(z, x), \neg patient(z, y)\} \end{array}$$

The resolution rule can be applied 4 times. After that, C has been unified with z (because we have resolved $love(C)$ with $love(z)$), B with y (because we have resolved $princess(B)$ with $princess(y)$), and A with x (because we have resolved $agent(C, A)$ with $agent(z, x)$). The formula rT is $ogre(A)$, and rH is $\neg monster(A)$. So the rule that we generate is:

$$\forall x.ogre(x) \Rightarrow monster(x)$$

The modified Robinson resolution thus does two things at once: It removes words that T and H have in common, leaving the words for which inference rules are needed, and it aligns words and phrases in T with words and phrases in H through unification.

One important refinement to this general idea is that we need to distinguish content predicates that correspond to content words (nouns, verb and adjectives) in the sentences from non-content predicates such as Boxer's meta-predicates $agent(X, Y)$. Resolving on non-content predicates can result in incorrect rules, for example in the case of *T: A person solves a problem* and *H: A person finds a solution to a problem*, in CNF:

$$\begin{aligned} T: \quad & \{person(A)\}, \{solve(B)\}, \{problem(C)\}, \{agent(B, A)\}, \\ & \{patient(B, C)\} \\ \neg H: \quad & \{\neg person(x), \neg find(y), \neg solution(z), \neg problem(u), \neg agent(y, x), \\ & \neg patient(y, z), \neg to(z, u)\} \end{aligned}$$

If we resolve $patient(B, C)$ with $patient(y, z)$, we unify the problem C with the solution z, leading to a wrong alignment. We avoid this problem by resolving on non-content predicates only when they are fully grounded (that is, when the substitution of variables with constants has already been done by some other resolution step involving content predicates).

In this variant of Robinson resolution, we currently do not perform any search, but unify two literals *only* if they are fully grounded or if the literal in T has a *unique* literal in H that it can be resolved with, and vice versa. This usually works if T and H are short and have relatively similar structure as in the RTE and STS datasets we use. For longer sentences, we use the

graph-based alignment discussed in section 7.3.1.

Rule Refinement The modified Robinson resolution algorithm gives us one rule per text/hypothesis pair. This rule needs postprocessing, as it is sometimes too short (omitting relevant context), and often it combines what should be several inference rules.

In many cases, a rule needs to be extended. This is the case when it only shows the difference between text and hypothesis is too short and needs context to be usable as a distributional rule, for example: *T: A dog is running in the snow*, *H: A dog is running through the snow*, the rule we get is $\forall x, y.\, in(x, y) \Rightarrow through(x, y)$. Although this rule is correct, it does not carry enough information to compute a meaningful vector representation for each side. What we would like instead is a rule that infers "run through snow" from "run in snow".

Remember that the variables x and y were Skolem constants in rT and rH, for example $rT : in(R, S)$ and $rH : through(R, S)$. We extend the rule by adding the content words that contain the constants R and S. In this case, we add the *running* event and the *snow* back in. The final rule is: $\forall x, y.\, run(x) \wedge in(x, y) \wedge snow(y) \Rightarrow run(x) \wedge through(x, y) \wedge snow(y)$. Here is another example: *T: A person is pouring olive oil into a pot* , *H: A person is pouring cooking oil into a pot*, and the rule is $\forall x.\, olive(x) \Rightarrow cooking(x)$ which we extend to $\forall x.\, olive(x) \wedge oil(x) \Rightarrow cooking(x) \wedge oil(x)$

In some cases however, extending the rule adds unnecessary complexity,

for example: *T: A man is jumping into an empty pool*, *H: A man is jumping into a full pool* and the rule is $\forall x.\, empty(x) \Rightarrow full(x)$, and extending it gives $\forall x.\, empty(x) \wedge pool(x) \Rightarrow full(x) \wedge pool(x)$ which makes the rule unnecessary complex. In other examples, part of the rule better be extended and another part not, for example: *T: A man is eating a bowl of cereal*, *H: A man is eating cereal*, and the rule is: $\forall x, y, z patient(x, y) \wedge bowl(y) \wedge of(y, z) \Rightarrow patient(x, z)$, which better be extended by adding the *cereal* but not adding the *eating* event.

At the moment, we have no general algorithm for when to extend a rule, which would have to take context into account. At this time, we extend all rules as described above.

Sometimes, long rules need to be split. A single pair T and H gives rise to one single pair rT and rH, which often conceptually represents multiple inference rules. So we split rT and rH as follows. First, we split each formula into disconnected sets of predicates. For example, consider *T: The doctors are healing a man*, *H: The doctor is helping the patient* which leads to the rule $\forall x, y.\, heal(x) \wedge man(y) \Rightarrow help(x) \wedge patient(y)$. The formula rT is split into $heal(x)$ and $man(y)$ because the two literals do not have any variable in common and there is no relation (such as $agent()$) to link them. Similarly, rH is split into $help(x)$ and $patient(y)$. If any of the splits has more than one verb, we split it again, where each new split contains one verb and its arguments.

After that, we create new rules that link any part of rT to any part of rH with which it has at least one variable in common. So for our example we get $\forall x\, heal(x) \Rightarrow help(x)$ and $\forall y\, man(y) \Rightarrow patient(y)$.

There are cases where splitting the rule does not work, for example with *A person, who is riding a bike* $\Rightarrow$ *A biker* . Here, splitting the rule and using *person* $\Rightarrow$ *biker* loses crucial context information.

4.3.1.2 Rules as lexical entailment dataset

The output from the previous steps is a set of rules $\{r_1, ..., r_n\}$ for each pair T and H. From these rules, we would like to learn how to evaluate unseen rules. We can use rules extracted from the training RTE examples as a dataset to train a lexical entailment classifier (section 4.3.2) that can evaluate unseen rules. Computing inference-rule training data from RTE data requires deriving labels for individual rules from the labels on RTE pairs (Entailment, Contradiction and Neutral). The Entailment cases are the most straightforward. Knowing that $T \wedge r_1 \wedge ... \wedge r_n \Rightarrow H$, then it must be that all r_i are entailing. We automatically label all r_i of the entailing pairs as entailing rules.

For Neutral pairs, we know that $T \wedge r_1 \wedge ... \wedge r_n \not\Rightarrow H$, so at least one of the r_i is non-entailing. We experimented with automatically labeling all r_i as non-entailing, but that adds a lot of noise to the training data. For example, if *T: A man is eating an apple* and *H: A guy is eating an orange*, then the rule *man* $\Rightarrow$ *guy* is entailing, but the rule *apple* $\Rightarrow$ *orange* is non-entailing. So we automatically compare the r_i from a Neutral pair to the entailing rules derived from entailing pairs. All rules r_i found among the entailing rules from entailing pairs are assumed to be entailing (unless $n = 1$, that is, unless we only have one rule), and all other rules are assumed to be non-entailing. We

found that this step improved the accuracy of our system. To further improve the accuracy, we performed a manual annotation of rules derived from Neutral pairs, focusing only on the rules that do not appear in Entailing. We labeled rules as either entailing or non-entailing. In our experiments with the SICK dataset, from around 5,900 unique rules, we found 737 to be entailing.

For Contradicting pairs, we make a few simplifying assumptions that fit almost all such pairs in RTE datasets. In most of the contradiction pairs, one of the two sentences T or H is negated. For pairs where T or H has a negation, we assume that this negation is negating the whole sentence, not just a part of it. We first consider the case where T is not negated, and $H = \neg S_h$. As T contradicts H, it must hold that $T \Rightarrow \neg H$, so $T \Rightarrow \neg\neg S_h$, and hence $T \Rightarrow S_h$. This means that we just need to run our modified Robinson resolution with the sentences T and S_h and label all resulting r_i as entailing.

Next we consider the case where $T = \neg S_t$ while H is not negated. As T contradicts H, it must hold that $\neg S_t \Rightarrow \neg H$, so $H \Rightarrow S_t$. Again, this means that we just need to run the modified Robinson resolution with H as the "Text" and S_t as the "Hypothesis" and label all resulting r_i as entailing.

The last case of contradiction is when both T and H are not negated, for example: *T: A man is jumping into an empty pool*, *H: A man is jumping into a full pool*, where *empty* and *full* are antonyms. As before, we run the modified Robinson resolution with T and H and get the resulting r_i. Similar to the Neutral pairs, at least one of the r_i is a contradictory rule, while the rest could be entailing or contradictory rules. As for the Neutral pairs, we

take a rule r_i to be entailing if it is among the entailing rules derived so far. All other rules are taken to be contradictory rules. We did not do the manual annotation for these rules because they are few.

Dataset We publish a dataset of the on the fly rules rules that our system constructs when running on SICK, along with gold standard annotations. The training and testing sets are extracted from the SICK training and testing sets respectively. The total number of rules (training + testing) is 12,510, only 10,213 are unique with 3,106 entailing rules, 177 contradictions and 6,928 neutral. This is a valuable resource for testing lexical entailment systems, containing a variety of entailment relations (hypernymy, synonymy, antonymy, etc.) that are actually useful in an end-to-end RTE system.

4.3.2 Lexical Entailment Classifier

For natural language understanding, we usually need lexical knowledge to reason about the relation between words and short phrases. Lexical entailment is the task of deciding if a lexical term (word or short phrase) entails another lexical term (Kotlerman et al., 2010). Lexical entailment systems are important within natural language understanding systems because it is not possible to collect all lexical information of a language.

The simplest form of lexical entailment is using cosine similarity of the vectors of the lexical terms. The issue with cosine similarity is that it is symmetric, which does not really match the task. For example, we want to find

that *car* is a *vehicle* but *vehicle* is not a *car*. Therefor, lexical entailment tools are usually implemented as supervised classifiers with various linguistics features extracted from the left hand side and right hand side of the rule. To give this classifier the ability to generalized to unseen vocabulary (which is a basic requirement for lexical entailment), the classifier uses the vector representations of the lexical terms and learns how to use them to signal entailment. The classifier's confidence is usually used as a weight indicating the strength of the rule.

We use the lexical entailment classifier developed by Roller and Cheng discussed in the paper (Beltagy et al., 2016). They use a set of linguistic features to train a logistic regression classifier. The features are word form features like POS and lemma, WordNet features and distributional features. We use their lexical entailment classifier in our RTE experiments. Given a trained lexical entailment classifier, we run it on unseen rules and use the classifier confidence as a rule weight.

4.4 Chapter Summary

This chapter discussed the details of our knowledge base component which encompasses all the lexical knowledge relevant to a particular text and a hypothesis. It discussed the collection of precompiled rules from WordNet and PPDB. It also discussed our Robinson resolution alignment algorithm, and how we use it to find relevant on-the-fly rules. Contributions of this chapter will be evaluated later in chapter 5 with the RTE task evaluation.

Chapter 5

RTE Task

In prior chapters, we discussed how to translate sentences to logical form, how to adapt logical form to PL and how to collect a relevant weighted knowledge base. These were mostly task independent discussions. This chapter, on the other hand, discusses RTE specific details. It shows how to represent the RTE task in PL, how to perform efficient MLN inference for RTE, and it evaluates this work on one of the RTE datasets. The work presented in this chapter has been published in (Beltagy & Mooney, 2014; Beltagy et al., 2016).

5.1 Chapter Overview

In prior chapters, we showed how to translate sentences to logical form, and how to adapt the logical form to PL in the form of the probabilistic representation $P(H|T, KB)$. We also showed how to build the KB for a given T and H.

This chapter discusses the details of how we perform the RTE task using PL. The RTE task is finding if T entails, contradicts or neutral to H. The first part of this chapter shows how to find the relation between T and

H using the more basic entailment problem $P(H|T, KB)$. It also discusses a special entity coreference assumptions that is necessary for the detection of contradictions. Finally, it shows how to use multiple parses to reduce the impact of parsing errors.

The second part of this chapter is inference. We discuss why MLNs are more appropriate for the task that PSL, then explain the details of our MLN implementation. We present an efficient MLN inference algorithm that calculates probability of complex logical forms not individual ground atoms at it is the case in standard MLN implementations. It also performs an optimization that utilizes the CWA to reduce size of graphical model and make inference more tractable. It also uses a simple weight learning procedure to map the weights in the KB to MLN weights.

Finally, this work is evaluated on the RTE task. We evaluate the runtime of the inference optimizations, and evaluate the effect of other components of our system as well. Our results show a state of the art performance on the SICK dataset.

5.2 Representing RTE

In the RTE task, we are given a text T and a hypothesis H and asked for a categorical decision between three categories, Entailment, Contradiction, and Neutral. In standard logic, we determine entailment by checking whether $T \wedge KB \Rightarrow H$, or its probabilistic version $P(H|T, KB)$ (section 3). A decision about Entailment/Neutral can be made by learning a threshold on the prob-

ability $P(H|T, KB)$. To differentiate between Contradiction and Neutral, we additionally calculate the probability $P(\neg H|T, KB)$. If $P(H|T, KB)$ is high while $P(\neg H|T, KB)$ is low, this indicates entailment. The opposite case indicates contradiction. If the two probabilities values are close, this means T does not significantly affect the probability of H, indicating a neutral case. To learn the thresholds for these decisions, we train an SVM classifier with LibSVM's default parameters (Chang & Lin, 2001) to map the two probabilities to the final decision. The learned mapping is always simple and reflects the intuition described above.

5.3 Coreference for RTE contradiction

This section discusses a special entity coreference assumption that is important for the detection of contradictions. Here is an RTE example for demonstration: *T: An ogre does not snore* and *H: An ogre snores*. Strictly speaking T and H are not contradictory because it is possible that the two sentences are referring to different *ogres*. Although the sentence uses ***an*** *ogre* not ***the*** *ogre*, RTE dataset annotators usually make the assumption that the *ogre* in H refers to the same *ogre* in T. In the SICK textual entailment dataset, many of the pairs that annotators have labeled as contradictions are only contradictions if we assume that some expressions corefer across T and H.

For the above examples, here are the logical formulas with coreference

in the *updated* $\neg H$:

$$T : \exists x.\, ogre(x) \wedge \neg(\exists y.\, agent(y, x) \wedge snore(y))$$

$$Skolemized\, T : ogre(O) \wedge \neg(\exists y.\, agent(y, O) \wedge snore(y))$$

$$H : \exists x, y.\, ogre(x) \wedge agent(y, x) \wedge snore(y)$$

$$\neg H : \neg\exists x, y.\, ogre(x) \wedge agent(y, x) \wedge snore(y)$$

$$updated\, \neg H : \neg\exists y.\, ogre(O) \wedge agent(y, O) \wedge snore(y)$$

Notice how the constant O representing the *ogre* in T is used in the *updated*$\neg H$ instead of the quantified variable x.

We use a rule-based approach to determining coreference between T and H, considering both coreference between entities and coreference of events. Two items (entities or events) corefer if they 1) have different polarities, and 2) share the same lemma or share an inference rule. Two items have different polarities in T and H if one of them is embedded under a negation and the other is not. For the example above, *ogre* in T is not negated, and *ogre* in $\neg H$ is negated, and both words are the same, so they corefer.

A pair of items in T and H under different polarities can also corefer if they share an inference rule. In the example of *T: A monster does not snore* and *H: An ogre snores*, we need *monster* and *ogre* to corefer. For cases like this, we rely on the inference rules found using the modified Robinson resolution method discussed in Section 4.3.1. In this case, it determines that *monster* and *ogre* should be aligned, so they are marked as coreferring. Here is another example: *T: An ogre loves a princess*, *H: An ogre hates a princess.* In this case, *loves* and *hates* are marked as coreferring.

5.4 Using multiple parses

In our framework that uses probabilistic inference followed by a classifier that learns thresholds (section 5.2), we can easily incorporate multiple parses to reduce errors due to misparsing. If we can obtain multiple parses for a text T and hypothesis H, and hence multiple logical forms, this should increase our chances of getting a good estimate of the probability of H given T.

The default CCG parser that Boxer uses is C&C (Clark & Curran, 2004). This parser can be configured to produce multiple ranked parses (Ng & Curran, 2012); however, we found that the top parses we get from C&C are usually not diverse enough and map to the same logical form. Therefore, in addition to the top C&C parse, we use the top parse from another recent CCG parser, EasyCCG (Lewis & Steedman, 2014).

So for a natural language text N_T and hypothesis N_H, we obtain two parses each, say S_{T1} and S_{T2} for T and S_{H1} and S_{H2} for H, which are transformed to logical forms T_1, T_2, H_1, H_2. We now compute probabilities for all possible combinations of representations of N_T and N_H: the probability of H_1 given T_1, the probability of H_1 given T_2, and conversely also the probabilities of H_2 given either T_1 or T_2. When we use multiple parses in this manner, the thresholding classifier is simply trained to take in all of these probabilities as features. In section 5.6.2, we evaluate using C&C alone and using both parsers.

5.5 MLN Inference

We believe that MLNs are more appropriate for the RTE task than PSL for various reasons. PSL conjunction uses Lukasiewicz's equation $I(l_1 \wedge l_1) = max\{0, I(l_1) + I(l_2) - 1\}$ which is more strict than taking the minimum. This equation works fine for short formulas, but for long lists of conjuncted predicates (as it is the case in the hypothesis), value of most formulas will be zero. Another issue with PSL is that PSL has a few syntax constraints on the logical formulas it accepts. These constraints do not allow text and hypothesis with negations and universal quantifiers. Because of these issues, we find MLNs to be more suitable for the RTE task.

MLN inference is usually intractable, and using MLN implementations "out of the box" does not work for our application. This section discusses an MLN implementation that supports complex queries and uses the closed world assumption (CWA) to decrease problem size, hence making inference more efficient. Finally, this section discusses a simple weight learning scheme to learn global scaling factors for weighted rules in KB from different sources.

5.5.1 Complex formulas as queries

Current implementations of MLNs like Alchemy (Kok et al., 2005) do not allow queries to be complex formulas, they can only calculate probabilities of ground atoms. This section discusses an inference algorithm for arbitrary query formulas.

The standard work-around Although current MLN implementations can only calculate probabilities of ground atoms, they can be used to calculate the probability of a complex formula through a simple work-around. The complex query formula H is added to the MLN using the hard formula:

$$H \Leftrightarrow result(D) \mid \infty \tag{5.1}$$

where $result(D)$ is a new ground atom that is not used anywhere else in the MLN. Then, inference is run to calculate the probability of $result(D)$, which is equal to the probability of the formula H. However, this approach can be very inefficient for some queries. For example, consider the following query:

$$H : \exists x, y, z.\, ogre(x) \wedge agent(y, x) \wedge love(y) \wedge patient(y, z) \wedge princess(z) \tag{5.2}$$

This form of an existentially quantified formula with a list of conjunctively joined atoms is very common in the inference problems we are addressing, so it is important to have efficient inference for such queries. In general, logical forms of natural language sentences usually comprise large numbers of existentially quantified variables and only rarely any other quantifiers. However, using this particular H in Equation 5.1 results in a very inefficient MLN. For the direction $H \Leftarrow result(D)$ of the double-implication in Equation 5.1, the existentially quantified formula is replaced with a large disjunction over all possible combinations of constants for variables x, y and z (Gogate & Domingos, 2011). Generating this disjunction, converting it to clausal form, and running inference on the resulting ground network becomes increasingly intractable as the number of variables and constants grow.

New inference method Instead, we propose an inference algorithm to directly calculate the probability of complex query formulas. In MLNs, the probability of a formula is the sum of the probabilities of the possible worlds that satisfy it. Gogate and Domingos (2011) show that to calculate the probability of a formula H given a set of probabilistic rules R (in our case of calculating $P(H|T, KB)$, $R = T \wedge KB$), it is enough to compute the partition function Z of R with and without H added as a hard formula:

$$P(H \mid R) = \frac{Z(R \cup \{(H, \infty)\})}{Z(R)} \tag{5.3}$$

Therefore, all we need is an appropriate algorithm to estimate the partition function Z of a Markov network. Then, we construct two ground networks, one with the query and one without, and estimate their Zs using that estimator. The ratio between the two Zs is the probability of H.

We tried to estimate Z using a harmonic-mean estimator on the samples generated by MC-SAT (Poon & Domingos, 2006), a popular and generally effective MLN inference algorithm, but we found that the estimates are highly inaccurate as shown by Venugopal and Gogate (2013). Instead we use SampleSearch (Gogate, 2009) to estimate the partition function. SampleSearch is an importance sampling algorithm that has been shown to be effective when there is a mix of probabilistic and deterministic (hard) constraints, a fundamental property of the inference problems we address. Importance sampling in general is problematic in the presence of determinism, because many of the generated samples violate the deterministic constraints, and they get rejected. Instead, SampleSearch uses a base sampler to generate samples then

uses backtracking search with a SAT solver to modify the generated sample if it violates the deterministic constraints. We use an implementation of SampleSearch that uses a generalized belief propagation algorithm called Iterative Join-Graph Propagation (IJGP) (Mateescu, Kask, Gogate, & Dechter, 2010) as a base sampler. This version is available online (Gogate, 2014).

For cases like the example H in Equation 5.2, we need to avoid generating a large disjunction because of the existentially quantified variables. So we replace H with its negation $\neg H$, replacing the existential quantifiers with universals, which are easier to ground and perform inference upon. Finally, we compute the probability of the query $P(H) = 1 - P(\neg H)$. Note that replacing H with $\neg H$ cannot make inference with the standard work-around faster, because with $\neg H$, the direction $\neg H \Rightarrow result(D)$ suffers from the same problem of existential quantifiers that we previously had with $H \Leftarrow result(D)$.

5.5.2 Inference Optimization using the Closed-World Assumption

This section explains why our MLN inference problems are computationally difficult, then explains how the closed-world assumption (CWA) (section 3.3.2) can be used to reduce the problem size and speed up inference.

In the inference problems we address, formulas are typically long, especially the query H. The number of ground clauses of a first-order formula is exponential in the number of variables in the formula, it is $O(c^v)$, where c is number of constants in the domain and v is number of variables in the formula. For a moderately long formula, the number of resulting ground clauses

is infeasible to process.

We have argued before (section 3.3.2) that for probabilistic inference problems based on natural language text/hypothesis pairs, it makes sense to make the closed world assumption: If we want to know if the query is true in the situation or setting laid out in the text, we should take as false anything not said in the text. In our probabilistic setting, the CWA amounts to giving low prior probabilities to all ground atoms unless they can be inferred from the text and knowledge base. However, we found that a large fraction of the ground atoms cannot be inferred from the text and knowledge base, and their probabilities remain very low. As an approximation, we can assume that this small probability is exactly zero and these ground atoms are false, without significantly affecting the probability of the query. This will remove a large number of the ground atoms, which will dramatically decrease the size of the ground network and speed up inference.

We assume that all ground atoms are false by default unless they are can be inferred from the text and the knowledge base $T \wedge KB$. For example:

$$T : ogre(O) \wedge agent(S, O) \wedge snore(S)$$

$$KB : \forall x.\, ogre(x) \Rightarrow monster(x)$$

$$H : \exists x, y.\, monster(x) \wedge agent(y, x) \wedge snore(y)$$

Ground atoms $\{ogre(O), snore(S), agent(S, O)\}$ are not false because they can be inferred from T. Ground atom $monster(O)$ is also not false because it can be inferred from $T \wedge KB$. All other ground atoms are false.

Here is an example of how this simplifies the query H. H is equivalent to a disjunction of all its possible groundings:

$$\begin{aligned} H : &(monster(O) \wedge agent(S,O) \wedge snore(S)) \\ &\vee(monster(O) \wedge agent(O,O) \wedge snore(O)) \\ &\vee(monster(S) \wedge agent(O,S) \wedge snore(O)) \\ &\vee(monster(S) \wedge agent(S,S) \wedge snore(S)) \end{aligned}$$

Setting all ground atoms to false except the inferred ones, then simplifying the expression, we get: $H : monster(O) \wedge agent(S,O) \wedge snore(S)$. Notice that most ground clauses of H are removed because they are False. We are left just with the ground clauses that potentially have a non-zero probability. Dropping all False ground clauses leaves an exponentially smaller number of ground clauses in the ground network. Even though the inference problem remains exponential in principle, the problem is much smaller in practice, such that inference becomes feasible. In our experiments with the SICK dataset, the number of ground clauses for the query ranges from 0 to 19,209 with mean 6. This shows that the CWA effectively reduces the number of ground clauses for the query from millions (or even billions) to a manageable number. With the CWA, the average number of inferrable ground atoms (ignoring ground atoms from the text) ranges from 0 to 245 with an average of 18.

Algorithm and Implementation Algorithm 5.1 describes the details of the grounding process with the CWA applied. Lines 1 and 2 initialize the set of inferrable ground atoms with the evidence and any ground atom in

Algorithm 5.1 MLN grounding with the CWA

Input: R: $\{K \cup Q\}$ set of first-order clauses, where K is the set of clauses from the input MLN, and Q is the set of clauses from the query.
Input: E: set of evidence (list of ground atoms)
Output: : a set of ground clauses with the inference optimization applied

```
1: Add all E to the inferrable ground atoms
2: Add all ground atoms in R to inferrable
3: repeat
4:     for all r ∈ R do
5:         p = propagate inferrable ground atoms between predicates sharing the same variable
6:         add propagated ground atoms (p) to inferrable
7:         if p not empty then
8:             changed = true
9:         end if
10:    end for
11: until not changed
12: Generate False evidence for ground atoms ∉ inferrable and add them to E
13: GC = Use MLN's grounding process to ground clauses R
14: for all gc ∈ GC do
15:    gc = gc after substituting values of known ground atoms in E
16:    if gc = True then
17:        drop gc
18:    else if gc = False then
19:        if gc is a grounding of one of Q's clauses then
20:            Terminate inference with Z = 0
21:        else if gc is hard clause then
22:            Error inconsistent MLN
23:        else drop gc
24:        end if
25:    else keep gc in GC
26:    end if
27: end for
28: return GC
```

R. Lines 3-11 repeatedly propagate evidence until there is no change in the inferrable set. Line 12 generates False evidence for all un-inferrable ground atoms. Line 13 generates all ground clauses, then lines from 14-31 substitute values of the known ground atoms in the ground clauses. Alchemy drops all True and False ground clauses, but this does not work when the goal of the inference algorithm is to calculate Z. Lines from 16-30 describe the change. True ground clauses are dropped, but not False ground clauses. If a False ground clause is a grounding of one of Q's clauses, then $Z = 0$ and there is no need to perform inference since there is no way to satisfy Q given E and R. If there is False hard clause, then this MLN is inconsistent. Otherwise, the False ground clause can be dropped. The resulting list of ground clauses GC are then passed to the inference algorithm to estimate Z.

5.5.3 Weight Learning

Our KB is a collection of weighted rules and the weights are all in the interval $[0, 1]$ (this is the case for classification confidence scores and PPDB weights) where 0 indicating a completely ineffective rules and 1 indicating a hard rule. MLN weights are in the interval $[0, \text{inf}]$ (negative weighted rules in MLNs are actually replaced with the negation of the rule and a positive weight), and the weights will be exponentiated before being used in the graphical model that MLNs construct. This makes it an important question to find out how to map the $[0, 1]$ weights from the knowledge base to $[0, \text{inf}]$ MLN weights. We do not have the same problem in PSL or our implementation for

PL inference and we can use the $[0, 1]$ weights directly.

Another reason for weight learning is that our KB contains weighted rules from different sources, and these weights are not necessarily on the same scale, for example one source could produce systematically larger weights than the other. Weight learning will help map them into uniform weights. This is not MLN specific, but we implemented and evaluated it only on our RTE on MLN experiments.

As with Zirn, Niepert, Stuckenschmidt, and Strube (2011), we learn a single mapping parameter for each source of rules that functions as a scaling factor:

$$MLNweight = scalingFactor \times ruleWeight \tag{5.4}$$

We use a simple grid search to learn the scaling factors that optimize performance on the RTE training data.

Given that all rule weights are in $[0, 1]$, we also try the following mapping:

$$MLNweight = scalingFactor \times \log(\frac{ruleWeight}{1 - ruleWeight}) \tag{5.5}$$

This function assures that for an MLN that only contains a single rule $LHS \Rightarrow RHS \mid MLNweight$, it is the case that $P(RHS|LHS) = ruleWeight$, given that $scalingFactor = 1$.

5.6 Evaluation

This section evaluates our system on the RTE task. It first evaluates the runtime of the described MLN inference algorithm, then it evaluates the impact of the knowledge base, multiple parses, coreferences and weight learning on RTE accuracy. We use the SICK dataset (section 2.4) in both experiments.

5.6.1 Inference Evaluation

This section evaluates the two inference processes mentioned above; the support of a complex query formula and the inference optimization using CWA. The goal of this section is to evaluate the computational efficiency of the proposed inference algorithm, therefor, we do not employ most of the ideas discussed above and we leave that for the next section. We translate text to logical form using C&C and Boxer, apply the Skolemization adaptation (section 3.3.1.1) and we use a KB of the precompiled rules only. We do not employ the rest of the logical adaptations (section 3.3.1), the RTE coreferences for contradiction (section 5.3) and multiple parses (section 5.4).

Systems Compared

- **mln**: This system uses MC-SAT (Richardson & Domingos, 2006) for inference without any modifications. It uses the work-around explained in section 5.5.1 to calculate the probability of a complex query formula.
- **mln+qf**: This system uses our SampleSearch inference to directly cal-

culate the probability of a query formula (qf).

- **mln+cwa**: This system uses MC-SAT with the work-around for computing the probability of a complex query formula, but uses our CWA-based inference optimization.

- **mln+qf+cwa**: This is our full proposed inference algorithm, supporting a query formula (qf) and doing the inference optimization (cwa)

We use a 30 minute timeout for each MLN inference run in order to make the experiments tractable. If the system times out, it outputs -1 indicating an error, and the classifier learns to assign it to one of the three RTE classes. Usually, because the Neutral class is the largest, timeouts are classified as Neutral.

Evaluation metrics

- Accuracy: Percentage of correct classifications (Entail, Contradict, or Neutral)

- CPU Time (completed runs): Average CPU time per run for the completed runs only, i.e. timed out runs are not included.

- Timeouts: Percentage of inferences that timeout after 30 minutes.

	Accuracy	CPU Time	Timeouts
mln	56.94%	2min 27s	95.78%
mln+qf	68.74%	1min 51s	29.64%
mln+cwa	65.80%	10s	2.52%
mln+qf+cwa	71.80%	7s	2.12%

Table 5.1: Systems' performance, accuracy, CPU Time for completed runs only, and percentage of Timeouts

Results and Discussion Table 5.1 summarizes the results of the experiments. First, for all systems, the CPU time (average time per run for completed runs only) is very short compared to the length of the timeout (30 minutes). This shows the exponential nature of the inference algorithms, either the problem is small enough to finish in a few minutes, or if it is slightly larger, it fails to finish in reasonable time.

Comparing the systems, results clearly show that the base system, **mln**, is not effective for the type of inference problems that we are addressing, almost all of the runs timed out. System **mln+qf** shows the impact of being able to calculate the probability of a complex query directly. It significantly improves the accuracy, and it lowers the number of timeouts; however, the number of timeouts is still large. System **mln+cwa** shows the impact of the inference optimization, demonstrating that it makes inference significantly faster, since the number of uninferrable ground atoms in our application is large compared to the total number of ground atoms. However, the accuracy of **mln+cwa** is lower than that of **mln+qf**, since calculating the probability of a query directly is more accurate than the standard work-around. Finally, **mln+qf+cwa** is

both more accurate and faster than the other systems, clearly demonstrating the effectiveness of our overall approach.

5.6.2 RTE and Knowledge Base Evaluation

This section evaluates the knowledge base (chapter 4) in the context of the RTE task. It also evaluates the RTE specific adaptations mentioned in this chapter, namely coreferences, multiple parses and weight learning. Evaluation shows that our full implementation achieves a state-of-the-art results on the SICK RTE dataset.

We evaluate the following system parts. The **logic** is our basic MLN-based logic system that computes two inference probabilities (section 5.2). It includes all logical form adaptations mentioned in chapter 3 and it uses the inference algorithm presented in section 5.5. The **coref** is coreference resolution to identify contradictions (section 5.3), and the **multiparse** signals the use of two parsers, the top C&C parse and the top EasyCCG parse (section 5.4).

We also evaluate different configurations of the knowledge base. The **rr** is a knowledge base containing rules extracted using Robinson resolution alignment 4.3.1 and weighted using the lexical entailment classifier 4.3.2. The **ppdb** component adds rules from PPDB paraphrase collection (Section 4.2.2). The **wlearn** component learns a scaling factor for **ppdb** rules, and another scaling factor for the **rr** rules that maps the classification confidence scores to MLN weights (Section 5.5.3). Without weight learning, the scaling factor for **ppdb** is set to 1, and all **rr** rules are used as hard rules (infinite weight).

Components Enabled	Train Acc.	Test Acc.
logic	72.1	71.7
+ coref	73.8	73.4
+ coref + ppdb	75.3	74.8
+ coref + ppdb + wlearn	76.5	76.3
+ coref + ppdb + wlearn + wn	78.8	78.4
+ coref + ppdb + wlearn + wn + handcoded	79.2	78.8
+ coref + ppdb + wlearn + wn + handcoded + multiparse	80.8	**80.4**

Table 5.2: Ablation experiment for the system components without **rr**

The **wlearn_log** component is similar to **wlearn** but uses equation 5.5, which first transforms a rule weight to its log odds. The **wn** component adds rules from WordNet (Section 4.2.1). In addition, we have a few **handcoded** rules (Section 4.2.3). Like **wn**, the components **hyp** and **mem** repeat information that is used as features for entailment rules classification but is not always picked up by the classifier. As the classifier sometimes misses hypernyms, **hyp** marks all hypernymy rules as entailing (so this component is subsumed by **wn**), as well as all rules where the left-hand side and the right-hand side are the same. (The latter step becomes necessary after splitting long rules derived by our modified Robinson resolution; some of the pieces may have equal left-hand and right-hand sides.) The **mem** component memorizes all entailing rules seen in the training set of **rr**.

Sometimes inference takes a long time, so we set a 2 minute timeout for each inference run. If inference does not finish processing within the time limit, we terminate the process and return an error code. About 1% of the dataset times out.

Ablation Experiment without rr Because **rr** has the most impact on the system's accuracy, and when enabled suppresses the contribution of the other components, we evaluate the other components first without **rr**. In the following section, we add the **rr** rules. Table 5.2 summarizes the results of this experiment. The results show that each component plays a role in improving the system accuracy. Our best accuracy without **rr** is 80.4%.

Each rule set (**ppdb**, **wn**, **handcoded**) improves accuracy by reducing the number of false negatives. We also note that applying weight learning (**wlearn**) to find a global scaling factor for PPDB rules makes them more useful. The learned scaling factor is 3.0. When the knowledge base is lacking other sources, weight learning assigns a high scaling factor to PPDB, giving it more influence throughout. When **rr** is added in the following section, weight learning assigns PPDB a low scaling factor because **rr** already includes a large set of useful rules, such that only the highest weighted PPDB rules contribute significantly to the final inference.

The last piece we tested is the use of multiple parses (**multiparse**). Many of the false negatives are due to misparses. Using two different parses reduces the impact of the misparses, improving the system accuracy.

Ablation Experiment with rr In this experiment, we first use **rr** as a knowledge base, then incrementally add the other system components. Table 5.3 summarizes the results. First, we note that adding **rr** to the knowledge base KB significantly improves the accuracy from 73.4% to 83.0%. This is

Components Enabled	Train Acc.	Test Acc.
logic + coref	73.8	73.4
logic + coref + rr	84.0	83.0
+ handcoded	84.6	83.2
+ handcoded + multiparse	85.0	83.9
+ handcoded + multiparse + hyp	85.6	83.9
+ handcoded + multiparse + hyp + wlearn	85.7	84.1
+ handcoded + multiparse + hyp + wlearn_log	85.9	84.3
+ handcoded + multiparse + hyp + wlearn_log + mem	93.4	**85.1**
+ handcoded + multiparse + hyp + wlearn_log + mem + ppdb	93.4	84.9
current state of the art (Lai & Hockenmaier, 2014)	–	84.6

Table 5.3: Ablation experiment for the system components with **rr**, and the best performing configuration

higher than what **ppdb** and **wn** achieved without **rr**. Adding **handcoded** still improves the accuracy somewhat.

Adding **multiparse** improves accuracy, but interestingly, not as much as in the previous experiment (without **rr**). The improvement on the test set decreases from 1.6% to 0.7%. Therefore, the rules in **rr** help reduce the impact of misparses. Here is an example to show how: *T: An ogre is jumping over a wall*, *H: An ogre is jumping over the fence* which in logic are:

$$T\colon\ \exists x, y, z.\, ogre(x) \wedge agent(y, x) \wedge jump(y) \wedge over(y, z) \wedge wall(z)$$

$$H\colon\ \exists x, y, z.\, ogre(x) \wedge agent(y, x) \wedge jump(y) \wedge over(y) \wedge patient(y, z) \wedge wall(z)$$

T should entail H (strictly speaking, *wall* is not a *fence* but this is a positive entailment example in SICK). The modified Robinson resolution yields the following rule:

F: $\forall x, y. jump(x) \wedge over(x, y) \wedge wall(y) \Rightarrow jump(x) \wedge over(x) \wedge patient(x, y) \wedge wall(y)$

Note that in T, the parser treats *over* as a preposition, while in H, *jump over* is treated as a particle verb. A lexical rule *wall* $\Rightarrow$ *fence* is not enough to get the right inference because of this inconsistency in the parsing. The rule F reflects this parsing inconsistency. When F is translated to text for the entailment classifier, we obtain *jump over wall* $\Rightarrow$ *jump over fence*, which is a simple phrase that the entailment classifier addresses without dealing with the complexities of the logic. Without the modified Robinson resolution, we would have had to resort to collecting "structural" inference rules like $\forall x, y.\, over(x, y) \Rightarrow over(x) \wedge patient(x, y)$.

Table 5.3 also shows the impact of **hyp** and **mem**, two components that in principle should not add anything over **rr**, but they do add some accuracy due to noise in the training data of **rr**.

Weight learning results are the rows **wlearn** and **wlearn_log**. Both weight learning components help improve the system's accuracy. It is interesting to see that even though the SICK dataset is not designed to evaluate "degree of entailment", it is still useful to keep the rules uncertain (as opposed to using hard rules) and use probabilistic inference. Results also show that **wlearn_log** performs slightly better than **wlearn**.

Finally, adding **ppdb** does not improve the accuracy. Apparently, **rr** already captures all the useful rules that we were getting from **ppdb**. It is

interesting to see that simple distributional information can subsume a large paraphrase database like PPDB.

The system comprising **logic**, **cwa**, **coref**, **multiparse**, **rr**, **hand-coded**, **hyp**, **wlearn_log**, and **mem** achieves a state-of-the-art accuracy score of 85.1% on the SICK test set.

5.7 Chapter Summary

This chapter discussed the details of how we perform the RTE task using PL. We perform the RTE task using two PL inferences, the first is checking if the hypothesis can be entailed, and the second is checking if the negation of the hypothesis can be entailed. We also discussed an implicit coreference assumption that is usually being made, and we discussed how to add it to our logical form. In addition, we used multiple parser to reduce parsing errors.

On the inference side, we showed an MLN inference algorithm that calculates probabilities of complex logical queries not just individual ground atoms, and it employs an inference optimization based on CWA that makes inference more tractable.

We evaluated the proposed inference algorithm and showed a dramatical performance improvement. We also used this chapter to evaluate the knowledge base discussed in chapter 4 in the context of the RTE task. Our task evaluation shows that using the full implementation of our system achieves

a state of the art results on the SICK dataset.

Chapter 6

STS Task

The main goal of this chapter is to show that PL can be used to perform the STS task, and it discusses the details of this implementation. It shows how to represent the STS task in PL, then discusses how to solve this PL inference problem using MLNs and PSL. The work presented in this chapter has been published in (Beltagy et al., 2013, 2014).

6.1 Chapter Overview

The third component of our semantic representation is inference. We discussed the first two in chapters 3 and 4. This chapter mainly focuses on how to do inference for the STS task.

We show that we can view the STS task as two entailments, one entailment is entailing the second sentence from the first, and the other is entailing the first sentence from the second. However, we found this view of STS a lot more "strict" than how the task is defined, and what we actually need is a form of "partial" entailment where we assign partial credit if a part of the hypothesis is entailed. In the following sections, we present inference algorithms that support this form of inference.

We implemented PL inference for STS on MLNs and on PSL. In MLNs, we use standard MLN inference algorithms but we change the encoding of the hypothesis H to allow for partial entailment. In PSL, we define a new conjunction operator that is less strict that logical conjunction, and it also allows for partial entailments. This new conjunction operator requires a different grounding procedure than what PSL provides, which we develop.

Finally, we evaluate the two proposed inference procedures on three STS datasets. Our experiments show that PSL performs much better than MLNs when evaluated on the STS task.

6.2 Representing STS

In the STS task, we are given two sentences, S_1 and S_2. The STS task can be viewed as two entailment, one from S_1 to S_2 and one from S_2 and S_1. As a text and hypothesis, we need the two inferences $P(S_1|S_2, KB)$ and $P(S_2|S_1, KB)$. To produce a final similarity score, we train a regressor to learn the mapping between the two probabilities to the overall similarity score.

One issue with viewing STS as two entitlements is that the STS is a lot more "relaxed" that the RTE task. For example, S_1: "A man is driving fast" and S_2: "A man is driving a car", which in logic are

S_1: $\exists x_0, e_1.\, man(x_0) \wedge agent(e_1, x_0) \wedge drive(e_1) \wedge fast(e_1)$

S_2: $\exists x_0, e_1, x_2.\, man(x_0) \wedge agent(e_1, x_0) \wedge drive(e_1) \wedge patient(e_1, x_2) \wedge car(x_2)$

S_1 does not entail S_2 because there is no evidence for a *car* and S_2 does not entail S_1 because there is no evidence that the driving is $fast$. However, humans find these sentences to be quite similar because they share a *man* and *driving* event. It looks like annotators rated the data points in this task for a form of partial entailment where they give partial credit if part of the hypothesis is entailed. The following sections show how to support this form of inference in MLNs and PSL.

6.3 MLN inference

We showed in section 6.2 how to represent STS as an inference problem in the form $P(H|T, KB)$ and argued for the need of a form of partial entailments. In MLNs, we can replace the deterministic logical conjunction with a different way of combining evidence. We chose to use the average evidence combiner for MLNs introduced by Natarajan, Khot, Lowd, Tadepalli, Kersting, and Shavlik (2010). To use the average combiner, the full logical form is divided into smaller clauses (which we call mini-clauses), then the combiner averages their probabilities. In case the formula is a list of conjuncted predicates, a mini-clause is a conjunction of a single-variable predicate with a relation predicate (as in the example below). In case the logical form contains a negated sub-formula, the negated sub-formula is also a mini-clause. For the inference problem $P(S_2|S_1, KB)$ above, the hypothesis S_2 after dividing

clauses for the average combiner looks like this:

$$\begin{aligned} man(x_0) \wedge agent(e_1, x_0) &\Rightarrow result(x_0, e_1, x_2) \mid w \\ drive(e_1) \wedge agent(e_1, x_0) &\Rightarrow result(x_0, e_1, x_2) \mid w \\ drive(e_1) \wedge patient(e_1, x_2) &\Rightarrow result(x_0, e_1, x_2) \mid w \\ car(x_2) \wedge patient(e_1, x_2) &\Rightarrow result(x_0, e_1, x_2) \mid w \end{aligned} \tag{6.1}$$

where *result* becomes the query predicate. Here, *result* has all of the variables in the clause as arguments in order to maintain the binding of variables across all of the mini-clauses. The weights w are the following function of n, the number of mini-clauses (4 in the above example):

$$w = \frac{1}{n} \times log(\frac{\alpha}{1-\alpha}) \tag{6.2}$$

where α is a value close to 1 that is set to maximize performance on the training data. Setting w this way produces a probability of α for the *result*() in cases that satisfy the antecedents of *all* mini-clauses. For the example above, the antecedents of the first two mini-clauses are satisfied, while the antecedents of the last two are not since the premise provides no evidence for an object of the verb *drive*. The similarity is then computed to be the maximum probability of any grounding of the *result* predicate, which in this case is around $\frac{\alpha}{2}$.

The average combiner is very memory consuming since the number of arguments of the *result*() predicate can become large (there is an argument for each individual and event in the sentence). Consequently, the inference algorithm needs to consider a combinatorial number of possible groundings of the *result*() predicate, making inference very slow.

6.4 PSL inference

For several reasons, we believe PSL is a more appropriate PL tool for STS than MLNs. First, it is explicitly designed to support efficient inference, therefore it scales better to longer sentences with more complex logical forms. Second, it is also specifically designed for computing *similarity* between complex structured objects rather than determining PL entailment. In fact, the initial version of PSL (Broecheler, Mihalkova, & Getoor, 2010) was called *Probabilistic Similarity Logic*, based on its use of *similarity functions*. This initial version was shown to be very effective for measuring the similarity of noisy database records and performing *record linkage* (i.e. identifying database entries referring to the same entity, such as bibliographic citations referring to the same paper).

This section explains how we adapt PSL's inference to be more suitable for the STS task. As mentioned above, we need a form of partial entailment with a relaxed conjunction to fit the STS task. PSL does not support any of that "out of the box", so we develop a relaxed conjunction, and make the required changes to the optimization problem and the grounding technique.

6.4.1 Relaxed Conjunction

As mentioned above, Lukasiewicz's formula for conjunction is very restrictive and does not work well for STS. Therefore, we replace it with a new averaging interpretation of conjunction that we use to interpret the hypothesis

H. The truth value of the proposed average function is defined as:

$$I(p_1 \wedge .... \wedge p_n) = \frac{1}{n} \sum_{i=1}^{n} I(p_i) \tag{6.3}$$

where p_i is one of the conjuncted ground atoms. This averaging function is linear, and the result is a valid truth value in the interval [0, 1], therefore this change is easily incorporated into PSL without changing the complexity of inference which remains a linear-programming problem.

6.4.2 Heuristic Grounding

Grounding is the process of instantiating the variables in the quantified rules with concrete constants in order to construct the nodes and links in the final graphical model. In principle, grounding requires instantiating each rule in all possible ways, substituting every possible constant for each variable in the rule. However, this is a combinatorial process that can easily result in an explosion in the size of the final network (same problem in MLN). Therefore, PSL employs a "lazy" approach to grounding that avoids the construction of irrelevant groundings. If there is no evidence for one of the antecedents in a particular grounding of a rule, then the normal PSL formula for conjunction guarantees that the rule is trivially satisfied ($I(r) = 1$) since the truth value of the antecedent is zero. Therefore, its distance to satisfaction is also zero, and it can be omitted from the ground network without impacting the result of MPE inference. This approach has similar effect as the MLN inference optimization using the closed-world assumption from section 5.5.2.

However, this technique does not work once we switch to using averaging to interpret the query. For example, given the rule $\forall x.p(x) \wedge q(x) \Rightarrow t()$ and only one piece of evidence $p(C)$ there are no relevant groundings because there is no evidence for $q(C)$, and therefore, for normal PSL, $I(p(C) \wedge q(C)) = 0$ which does not affect $I(t())$. However, when using averaging with the same evidence, we need to generate the grounding $p(C) \wedge q(C)$ because $I(p(C) \wedge q(C)) = 0.5$ which *does* affect $I(t())$.

One way to solve this problem is to eliminate lazy grounding and generate all possible groundings. However, this produces an intractably large network. Therefore, we developed a heuristic approximate grounding technique that generates a subset of the most impactful groundings. Pseudocode for this heuristic approach is shown in algorithm 6.1. Its goal is to find constants that participate in ground atoms with high truth value and preferentially use them to construct a limited number of groundings of the query rule.

The algorithm takes the antecedents of a rule (in this case, the query formula Q) employing averaging conjunction as input. It also takes the *grounding limit* which is a threshold on the number of groundings to be returned. The algorithm uses several subroutines, they are:

- $Ant(v_i)$: given a variable v_i, it returns the set of rule antecedent atoms containing v_i. E.g, for the rule: $a(x) \wedge b(y) \wedge c(x)$, $Ant(x)$ returns the set of atoms $\{a(x), c(x)\}$.

- $Const(v_i)$: given a variable v_i, it returns the list of possible constants

Algorithm 6.1 PSL heuristic grounding for STS

Input: $r_{body} = a_1 \wedge .... \wedge a_n$: antecedent of a rule with average interpretation of conjunction
Input: V: set of variables used in r_{body}
Input: $Ant(v_i)$: subset of antecedents a_j containing variable v_i
Input: $Const(v_i)$: list of possible constants of variable v_i
Input: $Gnd(a_i)$: set of ground atoms of a_i.
Input: $GndConst(a, g, v)$: takes an atom a, grounding g for a, and variable v, and returns the constant that substitutes v in g
Input: gnd_limit: limit on the number of groundings

1: **for all** $v_i \in V$ **do**
2: **for all** $C \in Const(v_i)$ **do**
3: $score(C) = \sum_{a \in Ant(v_i)}(max \quad I(g))$ for $g \in Gnd(a) \wedge GndConst(a, g, v_i) = C$
4: **end for**
5: sort $Const(v_i)$ on scores, descending
6: **end for**
7: **return** For all $v_i \in V$, take the Cartesian-product of the sorted $Const(v_i)$ and return the top gnd_limit results

that can be used to instantiate the variable v_i.

- $Gnd(a_i)$: given an atom a_i, it returns the set of all possible ground atoms generated for a_i.

- $GndConst(a, g, v)$: given an atom a and grounding g for a, and a variable v, it finds the constant that substitutes for v in g. E.g, assume there is an atom $a = a_i(v_1, v_2)$, and the ground atom $g = a_i(A, B)$ is one of its groundings. $GndConst(a, g, v_2)$ would return the constant B since it is the substitution for the variable v_2 in g.

Lines 1-6 loop over all variables in the rule. For each variable, lines 2-5 con-

struct a list of constants for that variable and sort it based on a heuristic score. In line 3, each constant is assigned a score that indicates the importance of this constant in terms of its impact on the truth value of the overall grounding. A constant's score is the sum, over all antecedents that contain the variable in question, of the maximum truth value of any grounding of that antecedent that contains that constant. Pushing constants with high scores to the top of each variable's list will tend to make the overall truth value of the top groundings high. Line 7 computes a subset of the Cartesian product of the sorted lists of constants, selecting constants in ranked order and limiting the number of results to the grounding limit.

One point that needs to be clarified about this approach is how it relies on the truth values of ground atoms when the goal of inference is to actually find these values. PSL's inference is actually an iterative process where in each iteration a grounding phase is followed by an optimization phase (solving the linear program). This loop repeats until convergence, i.e. until the truth values stop changing. The truth values used in each grounding phase come from the previous optimization phase. The first grounding phase assumes only the ground atoms in the evidence set have non-zero truth values.

6.5 Evaluation

This section shows result of evaluating our PL on STS and compares MLNs and PSL. We compare the accuracy of their predictions and their computational efficiency. We also evaluate the effect of PSL grounding limit on

the results.

Experimental setup We translate text to logical form using C&C and Boxer. We use a simple knowledge base KB of WordNet rules in addition to lexical rules generated between all pairs of works in T and H, and rules are weighted using cosine similarity of vectors of the words in each rule.

Our experiments used the following three STS datasets:

- **SICK:** the same dataset we used for RTE experiments but annotated with human similarity scores (section 2.4).
- **msr-vid:** Microsoft Video Paraphrase Corpus from the competition of SemEval 2012 (Agirre et al., 2012). The dataset consists of 1,500 pairs of short video descriptions collected using crowdsourcing (Chen & Dolan, 2011) and subsequently annotated for the STS task. Half of the dataset is for training, and the second half is for testing.
- **msr-par:** Microsoft Paraphrase Corpus from SemEval 2012 (Agirre et al., 2012). The dataset is 5,801 pairs of sentences collected from news sources (Dolan, Quirk, & Brockett, 2004). Then, for STS 2012, 1,500 pairs were selected and annotated with similarity scores. Half of the dataset is for training, and the second half is for testing.

We evaluate the performance of MLNs and PSL using Pearson correlation. We also compare their computational efficiency in terms of average CPU time

per run and number of timeouts. We terminate inference after 10 minutes and count number of pairs that did not finish processing within the time limit. In case of a timeout, we return an error code. We train a regressor to map the two inference results to a final similarity score. The regressor also decides how to deal with the error code. We used Additive Regression (Friedman, 1999) model with WEKA's default parameters (Hall, Frank, Holmes, Pfahringer, Reutemann, & Witten, 2009).

	PSL		MLN			state of the art
	Corr.	runtime	Corr.	runtime	timeouts	Corr.
msr-vid	0.79	8s	0.63	1m 31s	8.8%	0.87
msr-par	0.53	30s	0.16	11m 49s	97.1%	0.68
SICK	0.74	10s	0.47	4m 24s	35.82%	0.82

Table 6.1: Pearson correlation, average CPU time per STS pair, and percentage of timed-out pairs in MLN with a 10 minute time limit. PSL grounding limit is set to 10,000 groundings.

Results Table 6.1 summarizes results of evaluating our experiments. Results show that PSL performance is a lot better that MLNs and it is an order of magnitude faster. One obvious reason for MLNs to perform worse than PSL is that MLNs are not computationally efficient and a large percentage of the dataset times out, while PSL always finished processing within the time limit. The other reason could be that MLN results are sensitive to the parameters of the average combiner.

As an attempt to improve MLN inference, it should be possible to apply the MLN inference optimization based on the CWA that we developed

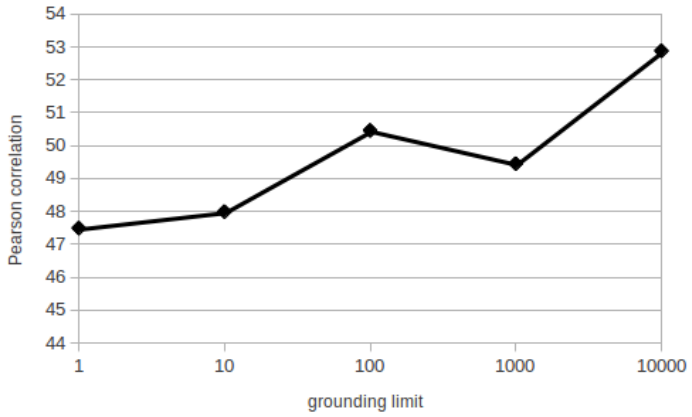
for the RTE task (section 5.5.2). This has the potential to significantly reduce the size of the ground network and make MLN inference faster. It will also make the comparison with PSL more fair, because PSL is already enforcing a comparable technique (lazy grounding) that plays a similar role in reducing the problem size. This is left for future work.

Our results are a bit lower than the state of the art results, but our results can potentially improve with more rules in the knowledge base. Notice that we only used the precompiled rules in this experiment, and our results might improve if we add on-the-fly rules. We have seen the same behavior with the RTE experiments.

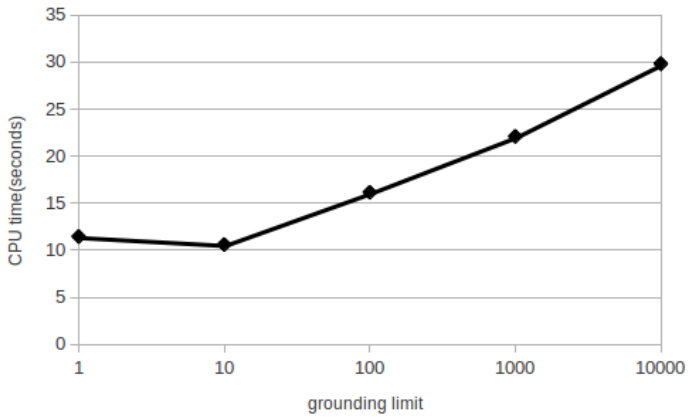
PSL grounding limit experiments We also evaluate the effect of changing the grounding limit on both Pearson correlation and CPU time for the **msr-par** dataset. Most of the sentences in **msr-par** are long, which results is large number of groundings, and limiting the number of groundings has a visible effect on the overall performance. In the other two datasets, the sentences are fairly short, and the full number of groundings is not large; therefore, changing the grounding limit does not significantly affect the results.

Figures 6.1a and 6.1b show the effect of changing the grounding limit on Pearson correlation and CPU time. As expected, as the grounding limit increases, accuracy improves but CPU time also increases. However, note that the difference in scores between the smallest and largest grounding limit tested is not large, suggesting that the heuristic approach to limit groundings is quite

effective.



(a) Correlation score



(b) CPU time

Figure 6.1: Effect of PSL's grounding limit on performance for the **msr-par** dataset

6.6 Chapter Summary

This section evaluated our semantic representation on the STS task. We showed how to view STS as PL inferences, and how to adapt PL inferences to the partial entailment of STS. We implemented it on MLNs and PSL and our experiments show that PSL is a better fit for the task; it is much faster and more accurate that MLNs for STS.

Chapter 7

QA Task

This chapter address our implementation for the QA task (introduced in section 2.4.3). The diversity and complexity of the text in QA datasets imposes additional challenges that are not in RTE and STS, and that require revisiting how we translate text to logical form, how we generate the knowledge base and the tools we use for inference. For the logical form, we present a new approach to translate dependency trees to logical form. For the knowledge base, we discuss a new graph-based alignment technique to generate on-the-fly rules. For the inference, this chapter explains the QA inference requirements and why MLNs and PSL are not very appropriate for the task. Then, it discusses a graphical model formulation for the problem and an inference algorithm to answer questions encoded in this graphical model. Finally, it presents and discusses the results of our evaluation.

7.1 Chapter Overview

In previous chapters, we discussed the translation of text to logical form and the collection of a KB, however, for the QA task, these need to be revisited. The text in the QA task is a lot longer and more complex and

diverse than what we had in RTE and STS, which means that we need more robust techniques for the translation to logical form and to build the KB. We use a rule-based technique to translate dependency parses (usually more accurate than CCG parses) to logical form. For the KB, we use a graph-based alignment algorithm to align T and H and extract the relevant rules.

The objective of the inference in QA is different than that for RTE and STS. Given the logical form of the document T and query $H(x)$ and the KB, inference should find an entity e from T such that it maximizes probability of $H(e)$ given the information in T and KB (for notation consistency, we refer to document and query with T and H). In addition to finding the answer entity for the question, there are two more requirements for the inference to fit the QA task. The first is that we need not just the answer entity, but the full assignment of entities from T to H. This full assignment is useful because we can project it back and find out which KB rules were relevant to producing the assignment. We need these rules for the training of the lexical entailment classifier (section 7.3.1). The second inference requirement is partial entailment -as in the STS task-, which is giving partial credit for the entailed part of H. This is important because it is often the case that parts of T can not be easily entailed (even for the right answer), and we want to give partial credit in this case.

This inference can be implemented in MLNs and PSL to some extent, but they are not the most appropriate tools for the task. Partial entailment in MLNs is not efficient as we have shown in our STS experiments. For PSL,

both requirements can be implemented with some effort. Partial entailment can be implemented with some changes to the averaging equation we used for STS, and finding the set of relevant rules can be found by inspecting the inference procedure. We have a preliminary implementation that does the first. However, we found that it would be easier to develop our own graphical model formulation for the task than modifying PSL inference. For every T, $H(x)$ and KB, we formulate a graphical model that encodes all possible ways of entailing H from T then we develop a search procedure that finds the best one.

Finally, we evaluate our implementation on one of the QA datasets. Our implementation is more than two orders of magnitude faster than our preliminary PSL implementation and it is a lot more accurate. However, we found that our results are limited by the accuracy of the lexical entailment classifier (which is not the focus of this work), and that we can achieve strong results with better lexical entailment classification.

7.2 Logical form

In section 3.2, we mentioned that we use Boxer to translate text to logical form and that Boxer runs on top of a CCG parse of the text. However, because of the complexity and diversity of the text in the QA task, the current CCG parsers are not very accurate. In this section we present an alternative approach to transform dependency parses to logical form. Dependency parsers are currently more accurate and mature than CCG parsers, which makes the

resulting logical form more robust. However, this logical form is less expressive because it does not capture all the linguistic phenomena that Boxer represents. The main limitation of this logical representation is that it fails to represent any phenomena that requires scope, e.g: negation (don't, no ...), generalized quantifiers (few, most, all ..) and relative clauses (forgot to, said that ...). Reddy, Täckström, Collins, Kwiatkowski, Das, Steedman, and Lapata (2016) recently introduced the same idea but with a few implementational differences. The main difference is that they have special rules for the translation of conjunctions, relative clauses and Wh questions.

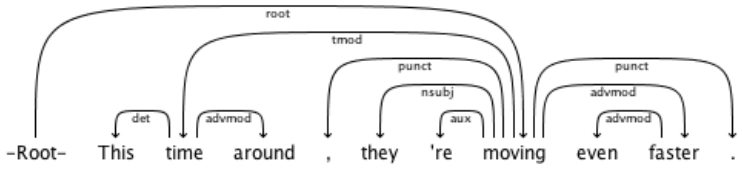


Figure 7.1: Dependency parse tree (Chen & Manning, 2014)

Translating dependency parses to logical forms A dependency parse is a structure that analyzes the relations between words in a sentence (Metzler, Noreault, Richey, & Heidorn, 1984). Words are connected with directed arrows points from a "head" word to a word that modifies it. Figure 7.1 shows an example dependency parse. The word *faster* modified the word *moving*. The dependency parse has a root which is usually the main verb in the sentence. In this example, it is the verb *moving*. Each dependency relation (arrow) has a name indicating the type of modification.

In our implementation, we use Stanford Dependency Parse (Chen & Manning, 2014) with Universal Dependencies. Universal dependencies are a special dependency standard that uses the same set of dependency relations for all languages. It has 42 dependency relations and 17 part-of-speech tags. Universal dependencies allow for language-specific relation sub-types. English has 7 additional relation sub-types.

Each dependency relation points from the "governor" word (head) to the "dependent" word (modifies the head). To translate a dependency parse to logical form, we iterate over all dependency relations starting from the root, and for each relation we greedily decide either to drop the dependent word, use the variable of the governor word for it, or introduce a new variable. The relations are split into three sets, `DROP`, `MERGE` and `NEW`, each set corresponds to one of the decisions. The relation `root` is a special case.

Here is the procedure in more details. Given a relation `rel`, a governor word `gov` and a dependent word `dep`, do one of the following:

- if `rel` = "root", then introduce a new variable `v` for `gov`. This adds the predicate `gov(v)` to the logical form
- if `rel` $\in$ `DROP`, then drop `dep`. This does not change the logical form (e.g for punctuation)
- if `rel` $\in$ `MERGE`, then get the variable `u` used for `gov`, and use it for `dep`. This adds the predicate `dep(u)` to the logical form (e.g for adjectives)

- if `rel` $\in$ `NEW`, then get the variable `u` used for `gov`, introduce a new variable `v` for `dep` and add `rel` connecting `u` and `v`. This adds the predicates `dep(u)` $\wedge$ `rel(u, v)` to the logical form (e.g for subjects)

The sets of relations mentioned above are:

`DROP`: det, det:predet, neg, case, dislocated, remnant, discourse, aux, auxpass, cop, mark, punct, cc, cc:preconj

`MERGE`: nmod:npmod, amod, advmod, compound, compound:prt, name, mwe, foreign, goeswith

`NEW`: nsubj, nsubjpass, dobj, iobj, csubj, csubjpass, ccomp, xcomp, nummod, appos, nmod, nmod:poss, nmod:tmod, acl, acl:relcl, advcl, list, parataxis, reparandum, vocative, expl, conj, dep

For the dependency parse in figure 7.1, we get the logical form:

$$\exists v_1, v_2, v_3.\, moving(v_1) \wedge faster(v_1) \wedge even(v_1) \wedge tmod(v_1, v_2) \wedge time(v_2) \wedge around(v_2) \wedge tmod(v_1, v_3) \wedge they(v_3)$$

7.3 Knowledge base

In section 4.3, we introduced the idea of alignment to generate on-the-fly rules, then presented an alignment technique using Robinson resolution. This alignment and rule extraction approach works reasonably well for short sentences with relatively similar structure (like in the RTE and STS datasets we used) but it is not suitable for longer more diverse sentences or documents as in the QA task. The main limitation of the Robinson resolution alignment

is that it assumes there is a unique way of aligning T and H and it matches entities of T with entities of H only when the words associated with the entities uniquely match. With longer text, these assumptions do not hold; there might be multiple valid alignments from T to H, and entities of H might have many potential matches in T. This section discusses a graph-based alignment that views T and H as graphs, finds an alignment between the nodes of two graphs, extracts rules from this node alignment, automatically weight the rules and use them to train a lexical entailment classifier.

7.3.1 Graph-based Alignment

Graph representation For the purpose of this algorithm, we view T and H as graphs. To do so, we assume that each of T and H is a set of conjuncted predicates like the one we construct from translating a dependency parse to logical form (section 7.2). We can use the logical form produced by Boxer after dropping negations and universal quantifiers. Each logical form will look like: $\exists e_1 \ldots\ pred_a(e_1) \wedge rel_u(e_1, e_2) \wedge pred_b(e_2) \wedge pred_c(e_2) \wedge rel_v(e_2, e_3) \ldots$ where e_i is a Boxer-type "entity", $pred_x(e_i)$ is a unary atom and the "predicate" $pred_x$ is one of the content words associated with entity e_i, and $rel_x(e_i, e_j)$ is a binary atom and the "relation" rel_x is the syntactic relation between the entities e_i and e_j. We will use the terms entity, predicate and relation throughout this section and later in QA inference (chapter 7).

It is straight forward to view the described logical form as an undirected graph. In this graph, entities are nodes, relations are undirected edges

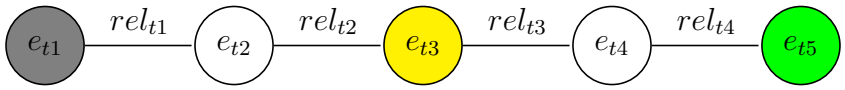
connecting nodes (entities), relation names are additional labels for edges, and predicates are additional labels for the nodes. Notice that the syntactic relations rel_x are originally directional, but we drop this direction and just use undirected edges.

Potential entity matches Our alignment algorithm can be viewed as trying to find the list of entities in T that best matches the list of entities in H (an alignment is a set of entity matches). For example, if we have an entity in H that is a *person* and we have multiple *person* entities in T, we want to find which one of them is the *person* entity of H. An alignment should respect two things; the lexical knowledge and similarity between words and phrases (which we cover in this section in the lexical rules we generate), and the overall structure of T and H (which we do through inference in chapter 7). Because we are trying to infer H from T, we typically assume that T has a lot of content and we are tying to find out which of this content entails H. In the graph sense, we want to find which subgraph of T matches the graph of H.

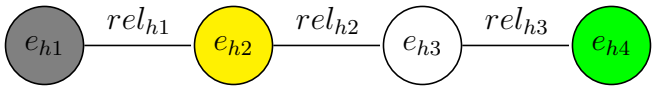
The first step of the alignment is to find potential matchings from entities of T to entities of H. The simplest form of deciding if two entities (nodes) can potentially match is if their associated words (node labels) share a common lemma. Entities can also potentially match if there is a rule that indicates that the associated words of the two entities are similar. The rule can be from a lexical resource like WordNet or a cosine similarity higher than some threshold. In the QA task, H has a special entity indicating the query

entity. The potential matches of this entity are all "named" entities in T. The result of this step is to have for each entity in H, a set of potentially matching entities from T.

Extracting rules From the graphs of T and H and the potential matching between entities, we can find the set of relevant lexical rules that we will later evaluate using lexical entailment. Notice that these rules correspond to multiple potential alignments from T to H. It is the role of inference to find out which group of rules belong to the same alignment, use that to generate all alignments and evaluate each one of them to find the best one (chapter 7). This is different from the Robinson resolution alignment where all generated rules are part of the same alignment from T to H.



(a) Subgraph of T. Each node e_{ti} has an associated label $pred_{ti}$



(b) Graph of H. Each node e_{hi} has an associated label $pred_{hi}$

Figure 7.2: Node color indicate a potential entity match. White nodes do not have any potential matches

To explain how we use the graph of T and H and the potential matches to extract rules, we will use the simplified example in figure 7.2 for demonstration. Typically T is larger than H but for simplicity, the figure shows a small subgraph of T that can be used to infer H. Another simplification is that the

graphs are linear chains. Same node color indicates a potential match, while white nodes do not have any potential matches. Because the figure shows a subgraph of T, presumably there are many more gray, yellow and green nodes in T, but we focus just on this subgraph. Also each node is associated with one predicate, but in general nodes can be associated with more than one. Again, this is just a simplified example for the sake of demonstration, but the algorithm explained below works for any arbitrary graph.

Our procedure needs a start node in H. This node should have potential matches. Typically in the QA task, we start from the query node (query entity). In figure 7.2, we will assume it is the gray node e_{h1}. Given the graph of H, we list the nodes of the graph following a breadth first traversal starting from the query node. This is to say, list the nodes of the graph starting from the query node followed by all its neighbor nodes followed by their neighbors and so on. This gives us a sorted list of node to loop over and generate rules.

As a high level view, our algorithm generates rules from the query node to all other nodes, then if it finds two rules r_1 and r_2 where r_1 is already part of r_2, it updates r_2 by removing the common part.

Algorithm 7.1 describes the details of how we align and extract the rules. Line 1 gets a sorted list of nodes of H sorted by their breadth first traversal order, then line 3 loops over the nodes excluding the first which is q. Line 4 finds the shortest path from the query node to the current node, then excludes the part of this path that has been generated before. This gives us a start and end node in H. From the potential matches, we also have start

Algorithm 7.1 Graph alignment and rule extraction algorithm

Input: `H`: graph of the hypothesis
Input: `q`: query node of `H`
Input: `T`: graph of the text
Output: : a set of rules relevant to the pair `T` and `H`

```
 1: L = nodes of H sorted following breadth-first-traversal starting from q
 2: visited = Set (q)                                  ▷ A set of visited nodes
 3: for all hTo ∈ ( L except q) do
 4:     path = shortest path from q to hTo in H
 5:     hFrom = find node in path that ∈ visited and closest to hTo
 6:     rules = generateRules(hFrom, hTo)
 7:     allRules += rules
 8:     if count(rules) > 0 then
 9:         add hTo to visited
10:     end if
11: end for
12: return allRules
13: ==========================
14: def generateRules(hFrom, hTo)
15: pathH = shortest path from hFrom to hTo in H
16: for all tFrom ∈ potentialMatchs(hFrom) do
17:     for all tTo ∈ potentialMatchs(hTo) do
18:         pathT = shortest path from tFrom to tTo in T
19:         if pathT exists then
20:             rules += create rule pathT ⇒ pathH
21:         end if
22:     end for
23: end for
24: return rules
25: end
```

and end points in T (lines 16 and 17). Finally, a rule is generated from the shortest path connecting the two nodes of T and the shortest path connecting the two nodes of H.

Applying this procedure to the example in figure 7.2, the first iteration will generate the rule between the gray and yellow nodes which is $\forall e_{t1}, e_{t2}, e_{t3}.pred_{t1}(e_{t1}) \wedge rel_{t1}(e_{t1}, e_{t2}) \wedge pred_{t2}(e_{t2}) \wedge rel_{t2}(e_{t2}, e_{t3}) \wedge pred_{t3}(e_{t3}) \Rightarrow pred_{h1}(e_{t1}) \wedge rel_{h1}(e_{t1}, e_{t2}) \wedge pred_{h2}(e_{t2})$ then it will add node e_{h2} (yellow) to the list of visited node. Notice the change of variable names in the right hand side of the rule. In the second iteration, it will find the long path in H from the gray to the green node, then it will remove its beginning and leave just the path from the yellow to the green node. This will generate the rule $\forall e_{t3}, e_{t4}, e_{t5}.pred_{t3}(e_{t3}) \wedge rel_{t3}(e_{t3}, e_{t4}) \wedge pred_{t4}(e_{t4}) \wedge rel_{t4}(e_{t4}, e_{t5}) \wedge pred_{t5}(e_{t5}) \Rightarrow \exists e_{h3}.pred_{h2}(e_{t3}) \wedge rel_{h2}(e_{t3}, e_{h3}) \wedge pred_{h3}(e_{h3}) \wedge rel_{h3}(e_{h3}, e_{t5}) \wedge pred_{h4}(e_{t5})$. Notice that variables on the right hand side are either coming from the left hand side or existentially quantified.

Weighting rules Annotating the extracted rules the way we did for the rules extracted using Robinson resolution (section 4.3.1.2) can not be applied here because, as mentioned before, rules from Robinson resolution correspond to one alignment while the rules extracted here correspond to multiple potential alignments. This means, we first need to find out which alignment is the best one and use its rules to train a lexical entailment tool which will weight the rules. However, we can not find the best assignment without having weights on the rules in the first place. Such circular problems can be solved by expectation maximization, where we iterate between weighting the rules and finding the best alignment. Weighting the rules is done through training

a lexical entailment classifier.

Algorithm 7.2 Expectation maximization to train a lexical entailment classifier

Input: QA training set
Output: : Trained lexical entailment classifier

```
1: initialize lexical entailment classifier randomly or using a simple baseline
2: repeat
3:     weight KB using lexical entailment classifier
4:     POS = empty set of positive rules
5:     NEG = empty set of negative rules
6:     for all T, H pairs for QA training problems  do
7:         KB = unweighted rules extracted for T and H using Algorithm 7.1
8:         KB = weight KB using current lexical entailment classifier
9:         ALIGN = run inference to find the best alignment ( of the correct
   answer)
10:        R = find the KB rules used to produce ALIGN
11:        POS += R
12:        NEG += KB - R
13:    end for
14:    train lexical entailment classifier on POS and NEG
15: until no change in QA results
```

Algorithm 7.2 briefly shows our expectation maximization procedure to train the lexical entailment classifier. It starts with an initial lexical entailment classifier, uses it to weight the rules, does inference, automatically weights rules resulting into the right answer as positive and the rest as positive, retrains the lexical entailment classifier, and repeats.

The inference role is to find the best alignment (chapter 7). The algorithm requires an initial lexical entailment classifier. We tried to initially weight rules randomly and use a simple baseline and usually both reach the same final result.

Lexical Entailment Classifier We use the same lexical entailment classifier discussed in section 4.3.2 with small changes. We use Random Forest classifier instead of logistic regression. We experimented with different classifiers and found that the Random Forest classifier performs the best on the validation set. We also added a few additional features. The most important one is a lexicalized feature for words in the LHS and RHS of the rule. We scan all all rules and find the top 1,000 common words, then we have two features for each word indicating that the word exists in the LHS or the RHS of the rule. The rationale behind this feature is that it allows the classifier to learn common words that can be ignored. The most common ones are like *say*, *have* and *when*.

7.4 Inference

This section discusses how we perform inference for the QA task. It discusses how to represent the QA task in PL, then explains two requirements that the inference needs to be suitable for the QA task. We show how such inference might be implemented in MLNs and PSL and why they are not the most appropriate tools for the task, then present our own graphical model formulation for the task and show an inference algorithm that answers the question encoded in this graphical model.

7.4.1 Representing QA

In previous chapters, we had inference problems that are trying to calculate probability of some query formula. In QA, the goal of inference is different; we are given a document T and a query $H(x)$ for a variable x; and we want to find an entity e from T that maximizes the probability of $H(e)$ given the information in T and the KB. Formally, the inference problem is

$$\arg\max_x P(H(x)|T, KB) \tag{7.1}$$

In PL (or in graphical models in general), this inference problem is called maximum a posteriori inference (MAP), which is computing the most likely assignment for some variable. Notice that we refer to the entities of T as "entities" while we refer to the entities of H as "variables" because, from the graphical model's perspective, they are random variables and inference is looking for their best assignment.

However, for the purpose of our implementation for the QA task, and for the particular dataset we are evaluating on, solving the inference problem mentioned above is not sufficient. Our inference procedure has two additional requirements; finding the KB rules corresponding to the most likely assignment and supporting partial entailments centered around the query variable. The rest of this section discusses these two requirements in details then briefly discusses how we might implement them in MLN and PSL and why we think that none of them is appropriate for the task. The following section discusses

how we solve this inference problem using our own graphical model formulation.

7.4.2 Inference Requirements

$H(x)$ has the form $\exists v_1 \ldots \; pred_a(v_1) \wedge rel_u(v_1, v_2) \wedge \ldots$, and the query variable x is one of the variables v_is. We mentioned in section 7.3.1 that our goal is to find an alignment from T to H. An alignment is an "assignment" of entities from T to the variables v_i in H. We need the full assignment (not just value of x) because we need to project this assignment back to the KB and find out which lexical rules are relevant to this assignment, then use these rules as positive training instances for the lexical entailment classifier (algorithm 7.1).

To add this additional inference requirement (finding the full assignment, not just x), the inference problem changes to

$$\arg\max_V P(H(V)|T, KB) \tag{7.2}$$

where V is the list of all variables v_i of H, and the $\arg\max$ finds the list of entities from T that maximizes probability of H. This form of inference is called Most Probable Explanation (MPE); the task of finding the most probable assignment to all variables.

The second inference requirement is supporting partial entailments. The idea of partial entailment has been discussed before in the context of the STS task (chapter 6), which is assigning partial credit if a part of the hypothesis can be inferred. The same idea is relevant for QA because it is often

the case that only a part of the query $H(x)$ can be inferred for a given value of x, and we still need to be able to compare between different values of x even if $H(x)$ can not be completely entailed for all of them. The main difference between the partial entailment in STS and in QA is that in STS, we would like to assign partial credit if any part of H is inferable, while in QA we only assign partial credit for parts of H that are "syntactically connected" to the query variable x. This makes the QA task a form of STS between H and the context around an entity e_i in T.

Here is an example,

Document: *the @entity33 blessing to @entity28 on saturday could clear the way for a ground invasion . though the @entity26 has taken the lead with some 100 warplanes , the coalition partners include the @entity63 , @entity64 and @en tity65 .*

Question: *@placeholder blessing of military action may set the stage for a ground invasion*

First, *@entity33* is the one associated with the *blessing* which makes it the right answer to the question even though we can not infer *military action* from the sentence. Second, even though the first sentence has evidence of a *blessing* and a *ground invasion*, that does not increase the probability of any of the entities in the second sentence because they are not syntactically connected to the evidence in the first sentence.

7.4.3 MLN and PSL implementations

We made preliminary attempts to implement the explained inference requirements using MLNs and PSL and we found that they are not appropriate for the task. This section discusses the implementations and their issues. Even though we do not have results for our MLN and PSL attempts, this discussion is still useful because we use similar ideas in our own graphical model implementation.

MLN The inference problem $\arg\max_V P(H(V)|T, KB)$ can be represented in MLNs using the workaround mentioned in section 5.5.1. The query formula H will be added to the PL program in the form of the following rule:

$$H(v_1, v_2 \dots) \Leftrightarrow result(v_1, v_2 \dots) \mid \infty \tag{7.3}$$

then use MLNs to query the probability of all ground atoms of $result$. Every ground atom has the form $result(e_a, e_b \dots)$ where each e_x is an entity from T. The ground atom with the maximum probability will encode the best assignment of entities e_x to variables v_i. This approach however is very inefficient because we have an exponential number of groundings of $result$. To speed it up, we need the inference optimization discussed in section 5.5.2 which will identify and remove ground atoms that are known to have zero probability. In our preliminary implementation, we found inference to be reasonably fast with this optimization applied.

The problem with MLN inference is that we did not find a good way to implement partial entailment which is very crucial for this task. Our MLN

implementation for the STS task (section 6.3) showed that using the average combiner to implement partial entailment is not a promising direction. That is why we concluded that MLNs are not good fit for the task.

PSL Implementing partial entailment in PSL is relatively easy. Similar to our STS implementation, we encode the query using the averaging equation instead of conjunction. For grounding, we have a different (and a lot easier) procedure. We simply generate all possible "partial groundings" of $H(x)$ starting from the query variable x. For example,

$$T : book(B_1)$$
$$\wedge read(R_2) \wedge patient(R_2, B_2) \wedge book(B_2)$$
$$\wedge person(P_3) \wedge agent(R_3, P_3) \wedge read(R_3) \wedge patient(R_3, B_3) \wedge book(B_3)$$
$$\wedge person(P_4) \wedge agent(R_4, P_4) \wedge read(R_4)$$
$$\wedge person(P_5)$$
$$H : \exists x, y, z.\, person(x) \wedge agent(y, x) \wedge read(y) \wedge patient(y, z) \wedge book(z)$$

Grounding starts from x, grounds any evidence of $person$ then follows the graph structure of H to extend the groundings with any additional evidence it finds. In the example above, we get only three possible groundings for H which are (each line is one grounding):

$$person(P_3) \wedge agent(R_3, P_3) \wedge read(R_3) \wedge patient(R_3, B_3) \wedge book(B_3)$$
$$person(P_4) \wedge agent(R_4, P_4) \wedge read(R_4)$$
$$person(P_5)$$

We implemented this approach on PSL and inference was reasonably fast. We will use the same idea of partial grounding again in our graphical model implementation.

So far we have discussed how to implement partial entailment. Next, we explain how we might find the KB rules that are relevant to the best assignment. With PSL, we can find the rules of the best alignment without first finding the full assignment itself. We can keep the query $H(x)$ where x is the query variable, and add the following rule to PSL

$$H(x) \Leftrightarrow result(x) \mid \infty \tag{7.4}$$

then use PSL to calculate the probability of ground atoms of $result$, which is easy because number of ground atoms of $result$ is not large, and it is limited by number of named entities in T. Then, we can examine the PSL inference procedure to find the rules that were relevant to the result. Remember that PSL inference is a linear program where every ground clause is a linear constraint in the linear program. The solution to the optimization problem is a point that lies at the intersection of a set of constraints. This set of constraints is what defines the result, and the ground clauses corresponding to these constraints are the set of rules we are looking for. We did not explore how to practically implement this technique because we found that it would be simpler to develop our own graphical model formulation, and that it is faster than PSL.

7.4.4 Graphical model formulation and inference

So far we have discussed the requirements of an inference that would fit the QA task, then we showed how we might implement this inference in MLNs and PSL and discussed the limitations of this implementation. In this section, we develop our own graphical model formulation for the problem instead of relying on MLNs or PSL to do that, then present an inference algorithm that answers the question encoded in the graphical model.

Formulating graphical model A graphical model is a structure that encodes probability distribution over a set of random variables. Nodes are random variables, and each random variable has a set of possible values. Cliques encode dependencies between possible values of nodes in terms of potential functions.

We are given T, $H(x)$ and KB and we formulate a graphical model that encodes all possible ways of aligning T and H. As mentioned before, an alignment between T and H is specified by an assignment of entities from T to variables in H. In our graphical model formulation, we have a node corresponding to every variables v_i of H. Every node has a set of possible values that are specified by T and KB. The document T has a set of entities, and the KB specifies which entities have chances of being assigned to variables of H. The KB we use for the QA experiments is the graph-based alignment KB constructed in section 7.3.1. This KB is constructed based on a set of potential matches between the entities of T and the variables of H. As a

result, possible values for every node v_i in our graphical model is the set of potentially matching entities from T. We also add an additional possible value N indicating the introduction of a new entity that is not in T.

Using variables with multiple possible values is an important distinction between our graphical model formulation and how MLNs and PSL define their graphical models. Let's say we have the predicate $pred_i(v_j)$ in H, and that v_j has the possible values $e_1, \ldots e_n$. Our graphical model encodes this as one variable v_j that has n possible values. However, MLNs and PSL will generate the ground atoms $pred_i(e_1), \ldots pred_i(e_n)$, where each ground atom is a binary random variable (or continuous random variable with values between 0 and 1 in PSL). The n ground atoms have 2^n possible values while in our formulation we only have n possible values. Typically we can add additional constraints to the MLNs/PSL saying that only one of the ground atoms should be true, which will successfully reduce number of possible values to n but at the expense of additional processing during inference time to make sure this constraint is satisfied. In this regard, our formulation is more efficient because such constraint is automatically enforced by construction.

The next step of our graphical model formulation is defining the cliques and potential functions. All the rules in KB have the form

$$\forall a_1 \ldots a_m \, LHS(a_1 \ldots a_m) \Rightarrow$$
$$\exists b_1 \ldots b_l \, RHS(a_1 \ldots a_m, b_1 \ldots b_l) \;\mid\; w = score(LHS, RHS)$$

where the left hand side (LHS) matches part of T and the right hand side (RHS) matches part of H. Variables of the LHS correspond to a list of entities

$e_y \dots e_{y+m}$ in T and variables of the RHS correspond to entities $v_z \dots v_{z+m+l}$ in H. Even thought the rule is universally quantified, it actually fires once because it is constructed for a particular T/H pair, and the LHS of the rule matches one specific part of T. As a result, it would be a waste of computation to keep the rule universally quantified, and it would be a bit more efficient to directly use a ground clause. The rule above becomes:

$$LHS(e_y \dots e_{y+m}) \Rightarrow RHS(e_y \dots e_{y+m}, N \dots N) \mid w = score(LHS, RHS) \tag{7.5}$$

where $e_y \dots e_{y+m}$ are entities from T and $N \dots N$ are newly introduced entities (Skolemization, section 3.3.1.1). The rule can be simplified further knowing that the LHS of the rule is always true, which simplifies the rule to

$$RHS(e_y \dots e_{y+m}, N \dots N) \mid w = score(LHS, RHS) \tag{7.6}$$

Notice that this simplified rule is what we get if we use MLNs with the inference optimization of section 5.5.2, but not with PSL which keeps the full rules. What this rule does is increase the probability of assigning the entities $e_y \dots e_{y+m}, N \dots N$ to the variables $v_z \dots v_{z+m+l}$ of the graphical model. This is equivalent to a clique connecting the variables $v_z \dots v_{z+m+l}$ and a potential function of weight $\exp(w)$ for the assignment $e_y \dots e_{y+m}, N \dots N$ and $\exp(0)$ for all other combinations of values. For the rest of this chapter, we will follow the PL terminology and refer to "rules" instead of referring to "cliques" and "potential functions". Whenever the rule is "satisfied" the $\exp(w)$ is used, and the $\exp(0)$ is used otherwise.

To summarize, we construct the graphical model $G(V, R)$, where V is a set of variables and R is a set of rules, as follows:

- Variables $v_i \in V$: the same variables of the logical form H
- Possible values for each v_i: entity potential matches from section 7.3.1 in addition to another special value (N) indicating the introduction of a new entity that is not in T
- Potential functions $r_i \in R$: every rule in KB is skolemized, grounded and simplified to a clique over the variables of the RHS of the rule with a potential function that has one weight equals $\exp(w)$ whenever the rule is satisfied and $\exp(0)$ otherwise, and w is the rule weight

Inference The form of inference we need is MPE; finding the best assignment to the variables of the graphical model $G(V, R)$:

$$\arg\max_A P(G(V = A, R)) = \arg\max_A \frac{\prod_{r \in R} r(A)}{Z} \tag{7.7}$$

where A is a variable assignment to the varaibles V, $r(A)$ is $\exp(w)$ or $\exp(0)$ depending if the rule is satisfied or not under the assignment A, and Z is the normalization constant. Z is a constant and it can be dropped without affecting the optimization problem. Also all weights of the rules are exponentiated and it would be convenient to take the log of the product, again, without affecting the result of the optimization problem. We get:

$$\arg\max_A P(G(V = A, R)) = \arg\max_A \sum_{r \in R} \log(r(A)) \tag{7.8}$$

which reads as: find an assignment A to the variables V with the maximum sum of weights of satisfied rules.

The brute force solution for this optimization problem is to enumerate all possible assignments A, sum the weights of satisfied rules, and choose the assignment with the maximum sum. This of course is not a practical solution. A practical solution should utilize the fact that most assignments do not satisfy any rule, and that assignments that satisfy more than one rule are very sparse. It would be more efficient to have a search guided by the rules.

Before discussing the details of our inference algorithm, we see how it can satisfy the inference requirements mentioned in the previous section. Assuming that the set of rules is empty, all assignments will equally score zero, and in this case we choose the default assignment of assigning N (new entity) to all variables. With rules added, different assignments will have different scores, and we are looking for the set of rules that dictates the best assignment. This implicitly addresses the first requirement because finding the best assignment is equivalent to finding the set of rules that are relevant to this assignment.

The second requirement is to support entailing a part of H that is syntactically connected to the query variable x. Entailing a part of H means that an assignment does not necessarily have to have a value for every variable. The constraint of the syntactic connectivity simplifies our inference algorithm a lot. To enforce syntactic connectivity, our inference algorithm starts with assigning all possible values for the query variable, then iteratively exploring

Algorithm 7.3 Inference for QA

Input: `V`: list of variables `v`
Input: `x`: query variable (`x` $\in$ `V`)
Input: `R`: set of rules `r` (set of pairs (v_i, e_i), weight `w`)
Output: : (assignment, score, rules) of the best assignment for `V`

```
queue = priority queue containing the assignments being explored
for all a possible values of x do
    queue.insert (assignment = assign a to x, score = 0, rules = empty)
end for
while queue is not empty do
    A = queue.dequeue                          ▷ current partial assignment
    if A.score > bestA.score then              ▷ better than the best assignment
        bestA = A
    end if
    proposed = empty set of (assignment, score, rules)
    for all r ∈ R and r ∉ A.rules do
        if r is not compatible with A.assignment then
            Ignore
        else if none of r.v_i have an assigned value in A.assignment then
            Ignore
        else    ▷ only consider rules with some overlap with A.assignment
            tmpA = add all pairs (v_i, e_i) of r to A.assignment
            proposed += (tmpA, A.score + r.w, A.rules + r)
        end if
    end for
    For every entry in proposed, find leastVarIndx; the least index of a
  variable that is assigned in proposed.assignment but not in A.assignment
    minLeastVarIndx = minimum of all leastVarIndx
    Remove from proposed where leastVarIndx > minLeastVarIndx
    queue.insert remaining entries in proposed
end while
return bestA
```

the search space of adding rules until no more rules can be added. Adding a rule to an assignment means setting the variables of the assignment to the

values that satisfy the rule. A rule is added to an assignment *only if* there is an overlap between the variables of the rule and the assigned variables of the assignment. This condition guarantees that each assignment we explore satisfies the constraint of syntactic connectivity to the query variable. It is important to notice that syntactically connected assignments are not necessarily globally optimal, but this is what we need for the task. As shown in the example in section 7.4.2, is would be wrong to give high weights to entities of the second sentence because of evidence in the first sentence.

Algorithm 7.3 outlines our inference procedure. Given a set of variables V, a query variable x and a set of weighted rules R, it finds the best assignment of entities to the variables V. The rules R are weighted using the lexical entailment classifier trained using the expectation maximization procedure explained in Algorithm 7.2.

The inference algorithm uses a priority queue to keep track of the assignments we are currently exploring. The queue prioritizes exploring the assignments based on their score. Even though we explore all possible syntactically connected assignments anyway, exploring the promising ones first reduces queue size. The queue is initialized with assignments that correspond to assigning all possible values for the query variable x (line 3). The initial score of these assignments is zero because they do not satisfy any rule yet. Then, while there are more assignments in the queue to explore, we get one assignment A from the queue to be considered. Then we use the rules to extend A and generate a set of proposed assignments. We keep the rules that 1)

are compatible with A (the can be satisfied under the assignment A) (line 12), and 2) have some overlap with A (some of the assigned variables in A are variables of the rule) (line 14). For each one of the remaining rules, we generate a new proposed assignment by applying the rule to A. A rule is applied to A by assigning more variables in A so that the rule is satisfied (lines 17, 18). The score of the assignment A increases by the weight of the rule it just satisfied (line 18).

By the end of the loop (line 20), we have a list of proposed assignments. Our algorithm would work correctly if we just add all these proposed assignments to the queue. However, doing so will not be efficient because we will be repeating a lot of work as a result of generating the same assignment so many times. Here is an example: let's say we have an assignment A and we have two rules r_1, r_2 that can be added to this assignment, where r_1 assigns a value to the variable v_i and produces a new proposed assignment A_1 and r_2 assigns a value to a different variable v_j and produces another proposed assignment A_2. Notice that in later processing, the rule r_2 can be applied to A_1 to produce $A_{1,2}$ and r_1 can be applied to A_2 to produce the same assignment $A_{1,2}$. This is the same assignment generated twice through two different search paths. This happens every time we have rules that can be applied to an assignment where each rule assigns a value to a different variable. The solution is to only keep proposed assignments that all share the same changed variable, and ignore the rest of the proposed assignments. This does not leave any search space unexplored because we know that the rules that we ignore in this iteration will

be considered in later iterations, and that the order of applying rules does not matter (we will reach the same assignment if we apply rules r_1 followed by r_2 or r_2 followed by r_1). To decide which proposed assignments to keep, we use the index of the changed variable. We keep the ones with the least changed variable index. In the example above, if $i < j$, we keep A_1, if $i > j$, we keep A_2 and if $i = j$, we keep both.

This part of the algorithm is in lines 21 - 24. In line 21, we find the index of the changed variable for every proposed assignment. If more than one variable changed, we use the smallest value. Line 22 finds the minimum of these indices. The actual order of the variables and using the minimum or maximum does not really matter. It is just a way to set a specific order for the exploration of the assignments. Line 23 removes some of the proposed assignments (because we know if we keep them, we will be repeating the same processing so many times later), and line 23 inserts the remaining ones in the queue for further processing.

Finally, the algorithm keeps track of the best assignment found so far, and at the end it returns this best assignment. As assignment consists of value for each variable in the graphical model, the rules that are satisfied under this assignment (we keep track of them while building the assignment), and the score of the assignment (sum of weights of the rules).

	Train	Validate	Test
Full KB	43.2%	42.0%	43.1%
Full KB + weight learning	74.2%	46.2%	48.2%
Limited KB	47.2%	48.1%	47.2%
Limited KB + weight learning	78.7%	50.0%	51.6%
Alignment classifier	57.7%	54.5%	56.3%
Alignment classifier + additional features	64.7%	61.8%	62.9%
PSL	–	33.0%	–
State of the art (Chen et al., 2016)	–	72.4%	72.4%

Table 7.1: QA results on the CNN part of the dataset by Hermann et al. (2015). Results on the training set with weigh learning can be viewed as the results of an oracle lexical entailment.

	Average runtime per question
Our graphical model	9 milliseconds
PSL	4 seconds

Table 7.2: Runtime comparison of our graphical model inference and PSL on the CNN part of the QA dataset

7.5 Evaluation

Experimental setup We evaluated our implementation on the QA dataset by Hermann et al. (2015). We only use the CNN part of the dataset for evaluation, and we only use the first 10K training questions out of 380K. We did not find the additional training data to help. Sentences are translated to logical form using our translation of dependency parses (section 7.2). This logical form does not encode universal quantifiers and negations, so the only relevant logical form adaptation is the Skolemization (section 3.3.1.1). The KB is constructed using the graph-based alignment (section 7.3.1). We use

the preliminary PSL implementation for comparison.

Compared Systems We compared different configurations of our system. The configuration **Full KB** refers to constructing a KB as discussed in section 7.3.1, while **Limited KB** refers to following the same procedure but limiting the generated rules to relations between the query variable and all its immediate syntactic neighbors.

We also compared with and without **weight learning**. The weight learning is implemented following algorithm 7.2.Without weight learning, we weight rules simply based on the difference of the length of the LHS and RHS of the rule: $w = \frac{1}{1+abs(len(LHS)-len(RHS))}$. This equation considers only the length of the rule, not the actual content. If both sides of the rule have the same length, the rule is given weight 1 which is the highest weight, and as the difference of the length increases, the weight decreases.

We also compare with training the classifier to evaluate full alignments instead of individual lexical rules. The classifier follows the entity-centric classifier of (Chen et al., 2016). We use inference to find the best alignment for each entity, then extract features from the alignment of each entity. We evaluated two configurations, **Alignment classifier** which extract features only from the alignment, and **Alignment classifier + additional features** which adds a few more simple features. The alignment features are number of rules, sum of rule weights, number of aligned entities, number of unaligned entities of the question, number of entities in the question and number of new

entities introduced in the RHS of rules. The additional features are simple unigram and bigram features and a simple word alignment score from the original paper of the dataset (Hermann et al., 2015).

Finally, we compare with **PSL**, which is the preliminary PSL inference algorithm that supports partial entailment. We did not use any weight learning in this configuration.

Results and Discussion Table 7.1 and table 7.2 show the results of our experiments. Obviously, the results of our graphical model formulation is a lot better than PSL. It is more than two orders of magnitude faster than PSL. We found that the number of assignments that our inference explores is usually a few tens or a few hundreds (average 90 assignments), which justifies why it is fast, because number of assignments that worth exploring is small. PSL is a lot slower because ground atoms are not multi-valued, and because of a lot of overhead in grounding and processing the rules.

With weight learning accuracy improves. However, there are a few issues with the lexical entailment classifier. The first is that we found that our lexical entailment classifier gives incorrectly high weights to many rules. As a result, we have a high rate of false positives and that reduces the accuracy of our system. To confirm this hypothesis, we see that (counter intuitively) the results improve with the limited KB because it reduces false positives. The second is that the classifier obviously overfits the training set, and a better classifier can potentially improve the results. The results for the training set

with weight learning can be viewed as the result of our system with an "oracle" lexical entailment. The oracle results reach 7x% which is a large room for improvement. In addition, this 7x% agrees with the conclusion of Chen et al. (2016) (discussed in section 2.4.3) that the maximum achievable result in this dataset is around 75%.

Results also shows that training an alignment classifier is more accurate that training the classifier to evaluate individual lexical rules. This is expected because the classifier is trained to optimize the final end task, not an artificial intermediate one.

Finally, our current best result is a lot better than the result of what we get from one of the standard PL tools, but our results are still a bit lower than state of the art. However, we believe that there is still a big room for improvement on training the lexical entailment classifier and training the alignment classifier.

7.6 Chapter Summary

In this chapter, we discussed our implementation for the QA task. We revisited the implementation of our pipeline to adapt it to the QA task. The logical form is constructed by a rule-base translation of dependency parses, the knowledge base is collected using a graph-based alignment technique. For inference, we discussed that it needs to support partial entailment and allow for finding the KB rules that were relevant for the result. These requirements can be achieved to some extent on MLNs and PSL but they are not the most

appropriate tool. As a result, we formulated or own graphical model for the problem instead of relying on MLNs or PSL for that, and developed an inference algorithm for it. Our formulation is simpler than modifying MLNs and PSL, and it is faster. Our inference algorithm explores the space of syntactically connected alignments from the document to the question and finds the best one. We evaluated our implementation on a QA dataset and found that our graphical model formulation is more than two orders of magnitude faster than PSL and it is a lot more accurate. We also compared between training a lexical entailment classifier and training an alignment classifier and found that an alignment classifier is more accurate. Our results did not reach the state of the art, but we believe there is still a big room for improvement.

Chapter 8

Future Work

This section suggests several ideas for future work. We discuss ideas for logical form including support for generalized quantifiers and a more accurate translation of dependency trees to logical form which will enable applying our semantic representation to other languages. We also suggest developing a new PL formulation that is more appropriate for natural language understanding tasks than existing ones. Finally, we discuss the potential use of deep learning for integrating symbolic and continuous representations.

8.1 Logical Form

Translating dependency trees to logical forms In section 7.2 we described a rule-based approach to translate dependency trees to logical form. The resulting logical form is more robust than what we get from CCG parsers because dependency parsers are currently more mature and more accurate. However, the logical form is less expressive because we do not represent any linguistic phenomena that requires scope like negations, quantifiers and relative clauses.

For future work, it would be useful to explore how to recover scope from

dependency parses. One way to do so is, again, using manually constructed rules. A more attractive option would be to "learn" the translation from data. The training data can come from running Boxer on the CCG parses of the CCGbank and running dependency parser on the sentences. Learning the mapping can potentially be more accurate than our rule-based approach not only because of finding scope, but also by capturing more phenomena that are not supported in dependency parses. One example is possessives as in *A man is driving* ***his*** *car*. Boxer output signals that the "car" belongs to the "man" with the addition of a meta predicate connecting them. The connection between the "car" and the "man" is not encoded in dependency trees.

Theoretically, our semantic representation is language independent, but the lack of training data and tools (other than a Chinese CCGbank (Tse & Curran, 2010)) prevents us from trying it on languages other than English. However, relying on translating dependency structure to logical form can address this limitation. There has been recently a lot of focus on *Universal Dependencies*; dependency structures that are unified over languages. The model we learn to translate English dependencies can be a good start point to translate dependency trees of other languages.

Generalized Quantifiers Generalized quantifiers are words like Few, Most, Many, Only .. etc (Barwise & Cooper, 1981). First-order logic has native support for two of them, Every and Some. Also, some of them can be encoded relatively easy in first-order logic, like Exactly. However, some of them like

Few, Most and Many are not natively supported in first-order logic, and they can not be encoded easily. We would like to add support for these generalized quantifiers in our system. A generalized quantifier has a restrictor and body. In "Most birds fly", the generalized quantifier "Most" has the restrictor "birds" and the body "fly". One way to add support for generalized quantifiers in the RTE task, is to reason about the direction of entailments between the restrictors and the bodies of the sentences. For example, consider the RTE pair: "T: Most big birds fly high", "H: Some birds fly". We can conclude that T entails H because the restrictor "big birds" of T entails the restrictor "birds" of H, and the body "fly high" of T entails the body "fly" of H. This is a purely symbolic handling of the problem.

Some of the generalized quantifiers, especially Most and Few, can be represented in a different way that leverages the probabilistic capabilities of PL. The idea is to to replace Most or Few with "Every" then give the logical formula a low weight indicating that some worlds could be violating it. Setting the weight is a function of the generalized quantifier being represented.

8.2 Inference

This section suggests a few inference-related ideas for future work.

PL framework for natural language understanding We explored the use of MLNs and PSL for different natural language understanding tasks, but our QA experiments showed that developing our own formulation is a more

productive, more efficient and more promising direction. It is also easier to develop a new formulation than adapting exciting ones. For future work, we would like to develop a new PL formulation that follows the general structure of our PL formulation for the QA task. The new formulation will view inference as a form of alignment. It facilitates the construction of graphical models that encodes all possible alignments then chooses the best one. As in our QA implementation, random variables will be multi-valued variables encoding potential matches between the constants of the given and the variables of the query.

This formulation will require new inference algorithms that can reason efficiently with multi-valued variables. The inference algorithm we developed for the QA task was efficient because of the syntactic connectivity constraint which reduced the search space of promising alignments. It would be interesting to find out how to develop efficient inference algorithm for this formulation. It is also worth exploring how to efficiently support negation and universal quantifiers in this formulation.

Finally, and similar to the graph matching classifier proposed above, we can generalize this idea to other tasks. The new PL formulation views inference as form of alignment. Instead of just using the alignment with the highest score, we can extract features from the alignments and train a final classifier to evaluate them. This is more flexible and can potentially give more accurate results.

8.3 Deep learning

The main motivation for this thesis is to build a semantic representation that combines the advantages of symbolic representations with continuous representations of meaning, and we used probabilistic logic to do so. With the recent advancements in neural networks and deep learning, it would be interesting to explore ways of combining symbolic and continuous representations using deep learning. Using deep learning for natural language inference can provide a flexible form of inference with powerful end-to-end training.

Deep learning uses various neural network (NN) architectures to map an input to some latent continuous representation then use this continuous representation to perform an end task the neural network is trained to perform. In natural language understanding, a lot of work have used the NN architecture of Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) to translate raw text to a continuous representation. LSTM is a form of recurrent neural network that reads the input document word by word and produces the continuous encoding of the input. This can be viewed as automatic feature discovery, where the LSTM automatically finds which features of the input are relevant to the end task, extract them and encode them in a vector of real values. With large enough training data, it can automatically learn to perform complex tasks like Machine Translation.

Despite the power of deep learning architectures, they lack some of the basic and simple capabilities that we get from symbolic representations. Andreas, Rohrbach, Darrell, and Klein (2016) developed a deep learning archi-

tecture that leverages the compositionality of language. Our aim is to develop an architecture that leverages more of the symbolic representation capabilities. One of them is interpretability; being able to inspect the meaning representation and interpret how the end task is performed. This is easy in symbolic representations (e.g. Boxer's output is easy to read) but difficult in deep learning because the meaning is a vector of real values. Another missing capability is being able to use existing resources to assist inference. For example, if we are doing RTE and we already know from WordNet that $man \Rightarrow person$, we should be able to use this knowledge instead of having to train on hundreds of training examples with the words *man* and *person*.

We propose the following ideas to add some support for symbolic information within deep learning:

Logical form as input Tree-LSTM is a variation of LSTM that takes a tree structure as input, not just a sequence of words (Tai, Socher, & Manning, 2015). We want to explore similar architectures to input a logical form to the NN. Our logical form describes the entities in the sentence and the relations between them. Feeding this structured input to the NN is potentially more informative than raw text, and it facilitates interpretability and integrating existing resources (discussed next). Equivalent ideas have been introduced before for propositional logic, where propositional formulas are mapped to a neural network then refine the weights of the neural network on the training data (Towell & Shavlik, 1994).

Interpretability The tasks RTE, STS and QA can all be viewed as aligning a text and a hypothesis then processing the alignment to make a conclusion. This can be implemented in NN using an architecture called "attention" which produces a soft alignment (a 2d matrix of real values indicating how much each word of the text aligns with words of the hypothesis) (Sutskever, Vinyals, & Le, 2014). However, instead of searching for an alignment between words, we want an alignment between entities of the logical form (as we did in the QA task). An alignment between entities can be easier to understand from an alignment between words. We can also use the pair-wise alignment score between entities to find a single best discrete alignment that is easy to read and interpret. This can facilitate debugging the neural network because we can inspect the alignment to find common mistakes then fix them with additional training data. For example, if we find that the NN mistakenly aligns $word_1$ and $word_2$ frequently, we can train it with additional examples (maybe synthetic ones) that focus on these two words.

Existing resources Existing linguistic resources (like WordNet and PPDB) and tools (like parsing, word sense disambiguation and coreference resolution) are the results of a large body of research in NLP. They can be informative to the inference and can reduce the training data requirements because we are not learning everything from raw text. We need a way to provide the output of all NLP tools and resources to the NN.

Using logical form as input already covers parsing, and coreferences

(can be easily represented in the logical form). Other resources like WordNet can be provided as additional input with every token. Even if we do not use logical form, parsing and part of speech tagging can also be provided as addition input with every input token. For example,

```
Text:  # # A man # # is # driving # a car ####
Parse:  ( ( DT NN ) ( VBZ ( VBG ( DT NN ))))
WordNet:  - - - guy - - - - - - - - - - - -
```

Chapter 9

Conclusion

In this work, we presented a natural language understanding system that relies on probabilistic logic for meaning representation and inference. It combines the expressivity of logic-based representations with the ability to reason with uncertainty. It uses first-order logic as a basic representation and integrates that with a weighted knowledge base that encodes a graded representation for words and short phrases. To show the generality and effectiveness of our proposed semantic representation, we developed and evaluated this system on three language understanding tasks, namely Textual Entailment, Textual Similarity and Question Answering. These tasks can utilize the strengths of our representation and the integration of logical representation and uncertain knowledge. Our framework is three components, Logical Form, Knowledge Base and Inference. We developed and made new contributions in each of them.

The logical form component starts with translating input sentences to logical form, and we use Boxer, a rule-based semantic analysis tool that runs on top of a CCG parse, for this translation. We also developed a rule-based translation of dependency parses to logical forms that are less expressive

but more robust and we only use it for the QA experiments. The resulting logical forms are adapted to PL to take the assumption of a closed domain into account. We showed how to introduce enough constants in the domain and how to set the prior probability of ground atoms to make sure universal quantifiers and negations work as expected. We evaluated these adaptations on three entailment datasets including a synthetic dataset that exhaustively tests inference performance on sentences with two quantifiers.

The second component is the knowledge base which encodes relevant background knowledge in the form of weighted logical rules. Relevant rules are collected from existing lexical resources like WordNet and PPDB, however, these resources are never complete and inference usually requires more lexical information. Therefor we generated additional on-the-fly rules by aligning the text and hypothesis then use the alignment to extract rules for each pair of text and hypothesis. We developed two alignment techniques, one based on Robinson resolution and another based on graph matching. The rules we extract from the alignment are automatically annotated as positive and negative, then we use them to train a lexical entailment classifier which we use to weight unseen rules.

The third component is Inference. We use the logical form and the weighted knowledge base collected from the previous two steps to formulate a task specific PL inference problems, then use the appropriate PL tools to solve them. The RTE task is represented in terms of two PL inferences, one to decide between entail and neutral, and the other to decide between contradiction and

neutral. The STS task is represented in terms of two inferences, one form the first sentence to the second, and the other from the second to the first. The QA task is represented with the PL inference of finding the entity that maximizes probability of the question.

We used MLNs for the RTE task. Existing MLN implementations do not work very well for the type of problems we are solving, so we developed a new MLN inference algorithm that can calculate the probability of a query formula (not just a single ground atom) and it automatically identifies and removes parts of the ground network that do not significantly affect the result. The logical form adaptations, the Robinson resolution alignment and this inference algorithm (in addition to a few other smaller contributions) achieve a state of the art result on the SICK textual entailment dataset.

For the STS task, we showed how to adapt MLN and PSL inference to support partial entailments. For MLNs, we replaced the conjunctions in the hypothesis with an average combiner. For PSL, we used an averaging function to encode the hypothesis instead of conjunctions and developed a grounding algorithm that fits it. Our experiments show that PSL is a lot faster and more suitable for this task than MLNs.

For the QA task, we developed our own graphical model formulation instead of relying on MLNs or PSL to do so. Then, we developed an MPE inference procedure that answers the question encoded in the graphical model. Our inference algorithm supports partial entailment (as in STS) and finds the KB rules that were relevant to the result which we need to train the

lexical entailment classifier. Results show that our implementation is more than two order of magnitude faster than a preliminary PSL implementation, that training an alignment classifier is more accurate than a lexical entailment classifier, and that there is still a big room of improvement in training both classifiers.

Overall, we showed that probabilistic logic is a powerful representation that can effectively integrate symbolic and continuous aspects of meaning. It is also flexible and can be adapted to various natural language understanding tasks. For future work, it would be interesting to develop a new probabilistic logic formulation that is more suitable for natural language tasks, instead of adapting existing ones.

Appendices

Appendix A

PL program example

The following RTE examples encompasses examples of all adaptations mentioned in section 3.3. As we mention in section 5.2, an RTE problem is two entailments $P(H|T, KB, W_{T,H})$ and $P(\neg H|T, KB, W_{T,\neg H})$. Figure A.1 shows the two full PL programs of the example:

T: A grumpy ogre is not smiling.

H: A monster with a bad temper is not laughing.

Which in logic are:

T: $\exists x.\, ogre(x) \wedge grumpy(x) \wedge \neg\exists y.\, agent(y, x) \wedge smile(y)$

H: $\exists x, y.\, monster(x) \wedge with(x, y) \wedge bad(y) \wedge temper(y) \wedge \neg\exists z.\, agent(z, x) \wedge laugh(z)$.

This example has the following rules in the knowledge base KB:

r_1: laugh $\Rightarrow$ smile

r_2: ogre $\Rightarrow$ monster

r_3: grumpy $\Rightarrow$ with a bad temper

Figure A.1 shows the two PL programs representing the RTE task (Section 5.2). D is the set of constants in the domain. T and r_3 are skolemized and sk is the skolem function of r_3 (Section 3.3.1). G is the set of non-False (True or unknown) ground atoms as determined by the CWA (Section 3.3.2, 5.5.2). A is the CWA for the negated part of H (Section 3.3.2.3). D, G, A are the world assumptions $W_{T,H}$ (or $W_{T,\neg H}$). r_1, r_2, r_3 are the KB. r_1 and its weight w_1 are from PPDB (Section 4.2.2). r_2 is from WordNet (Section 4.2.1). r_3 is constructed using the Robinson resolution alignment (Section 4.3.1), and its weight w_3 is calculated using the lexical entailment classifier (Section 4.3.2). The resource specific weights $w_{ppdb}, w_{eclassif}$ are learned using weight learning (Section 5.5.3). Finally the two probabilities are calculated using PL inference where H (or $\neg H$) is the query formula (Section 5.5.1)

$D : \{O, L, C_o\}$

$G : \{ogre(O), grumpy(O), monster(O), agent(L, O), smile(L), laugh(L),$
$skolem_f(O, C_o), with(O, C_o), bad(C_o), temper(C_o)\}$

$T : ogre(O) \wedge grumpy(O) \wedge \neg\exists y.\, agent(y, O) \wedge smile(y) \mid \infty$

$r_1 : \forall x.\, laugh(x) \Rightarrow smile(x) \mid w_1 \times w_{ppdb}$

$r_2 : \forall x.\, ogre(x) \Rightarrow monster(x) \mid w_{wn} = \infty$

$r_3 : \forall x.\, grumpy(x) \Rightarrow \forall y.\, skolem_f(x, y) \Rightarrow with(x, y) \wedge bad(y) \wedge temper(y) \mid$
$w_3 \times w_{eclassif}$

$sk : skolem_f(O, C_o) \mid \infty$

$A : \forall x.\, agent(L, x) \wedge laugh(L) \mid 1.5$

$H : \exists x, y.\, monster(x) \wedge with(x, y) \wedge bad(y) \wedge temper(y) \wedge \neg\exists z.\, agent(z, x)$
$\wedge laugh(z)$

(a) PL program to calculate $P(H|T, KB, W_{T,H})$

$D : \{O, C_o, M, T\}$

$G : \{ogre(O), grumpy(O), monster(O), skolem_f(O, C_o), with(O, C_o),$
$bad(C_o), temper(C_o), monster(M), with(M, T), bad(T), temper(T)\}$

$T : ogre(O) \wedge grumpy(O) \wedge \neg\exists y.\, agent(y, O) \wedge smile(y) \mid \infty$

$r_1 : \forall x.\, laugh(x) \Rightarrow smile(x) \mid w_1 \times w_{ppdb}$

$r_2 : \forall x.\, ogre(x) \Rightarrow monster(x) \mid w_{wn} = \infty$

$r_3 : \forall x.\, grumpy(x) \Rightarrow \forall y.\, skolem_f(x, y) \Rightarrow with(x, y) \wedge bad(y) \wedge temper(y) \mid$
$w_3 \times w_{eclassif}$

$sk : skolem_f(O, C_o) \mid \infty$

$A : monster(M) \wedge with(M, T) \wedge bad(T) \wedge temper(T) \mid 1.5$

$\neg H : \neg\exists x, y.\, monster(x) \wedge with(x, y) \wedge bad(y) \wedge temper(y)$
$\wedge \neg\exists z.\, agent(z, x) \wedge laugh(z)$

(b) PL program to calculate $P(\neg H|T, KB, W_{T,\neg H})$

Figure A.1: PL programs representing an RTE example

Appendix B

PPDB rules templates

This appendix lists all templates used to translate PPDB rules to logical form (section 4.2.2). As mentioned before, a PPDB rule that does not match any of the templates is ignored because it is usually the result of a misparse. Adjectives are denoted with N like nouns. Variables that do not have an associated adjective, adverb, noun or verb are denoted with the letter X. POS tags of a template are listed following the topological order of the variables in the rule (viewing variables are nodes in a graph and relations as directed edges). If two variables have the same topological order, we order them following their POS tag order V, N, X.

In each template, subscripts of POS tags specify the mapping of variables from LHS to RHS.

- $N_1 \Rightarrow N_1$ e.g: boy $\Rightarrow$ little kid
- $N_1N_2 \Rightarrow N_1$ e.g: piece of paper $\Rightarrow$ paper
- $N_1N_2 \Rightarrow N_1N_2$ e.g: slice of pizza $\Rightarrow$ piece of pizza
- $N_1X_2 \Rightarrow N_1X_2$ e.g: slice of X $\Rightarrow$ piece of X

- $X_1N_2 \Rightarrow X_1N_2$ e.g: X into the water $\Rightarrow$ X in the sea
- $X_1N_2X_3 \Rightarrow X_1N_2X_3$ e.g: X in front of Y $\Rightarrow$ X with Y
- $X_1X_2N_3 \Rightarrow X_1X_2N_3$
- $V_1N_2 \Rightarrow V_1N_2$
- $X_2 \Rightarrow N_1X_2$
- $X_1N_2 \Rightarrow X_1$ e.g: X is in the air $\Rightarrow$ X is up
- $X_1 \Rightarrow X_1$ e.g: quickly $\Rightarrow$ very fast
- $X_1X_3 \Rightarrow X_1N_2X_3$ e.g: X at Y $\Rightarrow$ X in front of Y
- $X_1X_2 \Rightarrow X_1X_2$ e.g: X in Y $\Rightarrow$ X near Y
- $X_1X_2X_3 \Rightarrow X_2X_3$ e.g: X next to Y $\Rightarrow$ X outside of Y
- $V_1X_3 \Rightarrow V_1N_2X_3$ e.g: looking at X $\Rightarrow$ standing in front of X
- $V_1X_2 \Rightarrow V_1X_2$ e.g: looking at X $\Rightarrow$ staring at X
- $V_1N_2 \Rightarrow V_1$ e.g: drinking water $\Rightarrow$ drinking
- $V_1X_2 \Rightarrow V_1$ e.g: looking at X $\Rightarrow$ watching (change RHS to watching X)
- $V_1 \Rightarrow V_1$ e.g: carrying $\Rightarrow$ holding

Appendix C

Handcoded rules

This appendix lists all handcoded rules mentioned in section 4.2.3.

- time ⇒ lot of time
- a boy ⇒ one boy
- a girl one ⇒ girl
- a man ⇒ one man
- a person ⇒ one person
- a woman ⇒ one woman
- a cat ⇒ one cat
- a dog ⇒ one dog
- a herd ⇒ one herd
- the man ⇒ one person
- cutting ⇒ slicing
- cutting ⇒ chopping down

- jumping ⇒ bouncing
- climbing ⇒ climbing up
- man⇒ person
- someone ⇒ ¬ nobody
- nobody ⇒ ¬ someone
- nobody ⇒ ¬ people
- people ⇒ ¬ nobody
- nobody ⇒ ¬ man
- man ⇒ ¬ nobody
- nobody ⇒ ¬ men
- men ⇒ ¬ nobody
- nobody ⇒ ¬ person
- person ⇒ ¬ nobody
- nobody ⇒ ¬ woman
- woman ⇒ ¬ nobody
- nobody ⇒ ¬ women
- women ⇒ ¬ nobody

- nobody $\Rightarrow \neg$ boy
- boy $\Rightarrow \neg$ nobody
- nobody $\Rightarrow \neg$ boys
- boys $\Rightarrow \neg$ nobody
- nobody $\Rightarrow \neg$ girl
- girl $\Rightarrow \neg$ nobody
- nobody $\Rightarrow \neg$ girls
- girls $\Rightarrow \neg$ nobody
- empty $\Rightarrow \neg$ full
- full $\Rightarrow \neg$ empty

Appendix D

QA Example

This appendix is a complete example of how a KB is constructed for a QA task, and how inference proceeds. For simplicity, the example is shown in text not logical form.

Given:

T: ... The Arab League is expected to give its official blessing to the military operation on Saturday, which could clear the way for a ground invasion, Becky Anderson reported. The Arab League actions are ...

H(x): x blessing of military action may set the stage for a ground invasion

Answer: *The Arab League*

Figure D.1 shows the interesting parts of graphs of T and H. Nodes with the same color are potentially matching nodes. The resulting KB has the following rules, and the first three rules correspond to the best alignment.

r1: Arab League expected to give official blessing ⇒ X blessing

r2: official blessing to military operation ⇒ blessing of military action

r3: official blessing clear way for ground invasion ⇒ blessing set stage for ground invasion

r4: Arab League actions ⇒ X blessing of military action

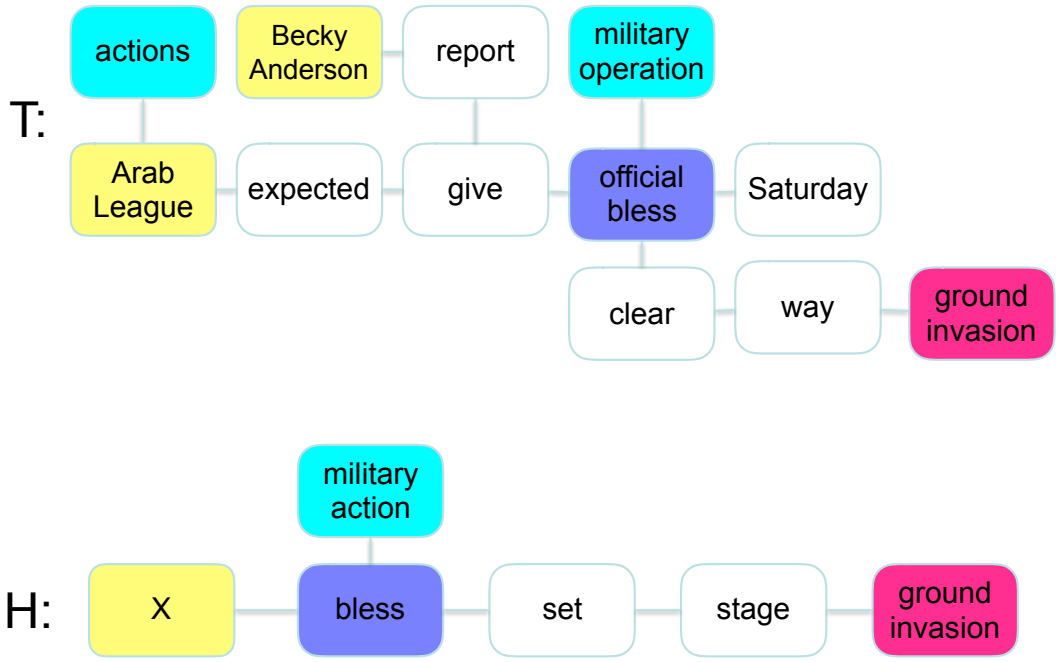


Figure D.1: Graphs representing T and H

r5: Becky Anderson reported give official blessing ⇒ X blessing

Figure D.2 shows the graphical model formulation. The top row are list of multivalued random variables and each one has a list of possible values. Rules are cliques between the values. Inference starts from possible values of X and explores the search space for the best assignment. The best assignment is specified by the values in the middle row.

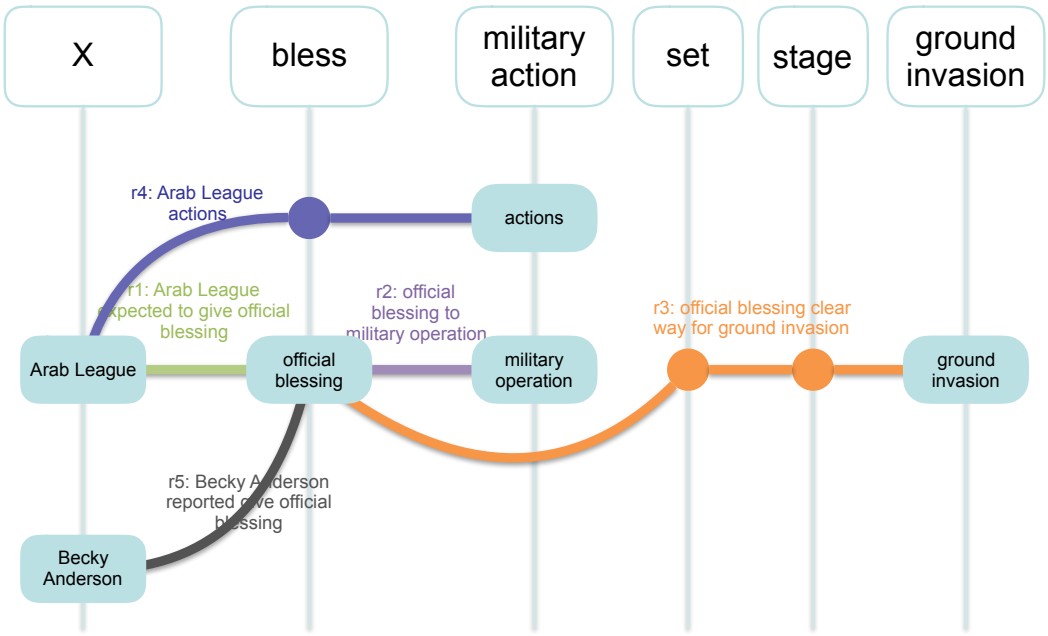


Figure D.2: Graphical model formulation for the QA task

Bibliography

Agirre, E., Cer, D., Diab, M., & Gonzalez-Agirre, A. (2012). SemEval-2012 task 6: A pilot on semantic textual similarity. In *Proceedings of the 6th International Workshop on Semantic Evaluation (SemEval-2012)*.

Alshawi, H. (1992). *The Core Language Engine*. MIT press.

Andreas, J., Rohrbach, M., Darrell, T., & Klein, D. (2016). Learning to compose neural networks for question answering. In *NAACL 2016*.

Bach, S. H., Huang, B., London, B., & Getoor, L. (2013). Hinge-loss Markov random fields: Convex inference for structured prediction. In *Proceedings of 29th Conference on Uncertainty in Artificial Intelligence (UAI-2013)*.

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

Beltagy, I., Chau, C., Boleda, G., Garrette, D., Erk, K., & Mooney, R. (2013). Montague meets Markov: Deep semantics with probabilistic logical form. In *Proceedings of the Second Joint Conference on Lexical and Computational Semantics (*SEM-2013)*.

Beltagy, I., & Erk, K. (2015). On the proper treatment of quantifiers in probabilistic logic semantics. In *Proceedings of the International Conference on Computational Semantics (IWCS-2015)*, London, Great Britain.

Beltagy, I., Erk, K., & Mooney, R. (2014). Probabilistic soft logic for semantic textual similarity. In *Proceedings of Association for Computational Linguistics (ACL-2014)*.

Beltagy, I., & Mooney, R. J. (2014). Efficient Markov logic inference for natural language semantics. In *Proceedings of AAAI 2014 Workshop on Statistical Relational AI (StarAI-2014)*.

Beltagy, I., Roller, S., Cheng, P., Erk, K., & Mooney, R. J. (2016). Representing meaning with a combination of logical and distributional models. *The special issue of Computational Linguistics on Formal Distributional Semantics*, *42*.

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

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

Bos, J. (2008). Wide-coverage semantic analysis with Boxer. In *Proceedings of Semantics in Text Processing (STEP-2008)*.

Bos, J. (2009). Applying automated deduction to natural language understanding. *Journal of Applied Logic*, *7*, 100–112.

Broecheler, M., Mihalkova, L., & Getoor, L. (2010). Probabilistic Similarity Logic. In *Proceedings of 26th Conference on Uncertainty in Artificial Intelligence (UAI-2010)*.

Brown, S. W. (2008). Choosing sense distinctions for WSD: Psycholinguistic evidence. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL 2008)*, Columbus, Ohio.

Chang, C.-C., & Lin, C.-J. (2001). LIBSVM: a library for support vector machines.. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chen, D., Bolton, J., & Manning, C. D. (2016). A thorough examination of the cnn / daily mail reading comprehension task. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*.

Chen, D., & Manning, C. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*.

Chen, D. L., & Dolan, W. B. (2011). Collecting highly parallel data for paraphrase evaluation. In *Proceedings of Association for Computational Linguistics (ACL-2011)*.

Clark, S. (2012). Vector space models of lexical meaning. In *Handbook of Contemporary Semantics* (2 edition). Wiley-Blackwell.

Clark, S., & Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. In *Proceedings of Association for Computational Linguistics (ACL-2004)*.

Cooper, R., Crouch, D., Van Eijck, J., Fox, C., Van Genabith, J., Jaspars, J., Kamp, H., Milward, D., Pinkal, M., Poesio, M., et al. (1996). Using the framework. Tech. rep., Technical Report LRE 62-051 D-16, The FraCaS Consortium.

Copestake, A., & Flickinger, D. (2000). An open-source grammar development environment and broad-coverage english grammar using HPSG. In *Proceedings of Language Resources and Evaluation Conference (LREC)*, Athens, Greece.

Curran, J., Clark, S., & Bos, J. (2009). Linguistically motivated large-scale nlp with c&c and boxer. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*.

Dagan, I., Roth, D., Sammons, M., & Zanzotto, F. M. (2013). Recognizing textual entailment: Models and applications. *Synthesis Lectures on Human Language Technologies*, *6*(4), 1–220.

Dolan, B., Quirk, C., & Brockett, C. (2004). Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *Proceedings of the Twentieth International Conference on Computational Linguistics (COLING-2004)*.

Dowty, D. R., Wall, R. E., & Peters, S. (1981). *Introduction to Montague Semantics*. D. Reidel, Dordrecht, Holland.

Edmonds, P., & Hirst, G. (2000). Reconciling fine-grained lexical knowledge and coarse-grained ontologies in the representation of near-synonyms. In *Proceedings of the Workshop on Semantic Approximation, Granularity, and Vagueness*.

Friedman, J. (1999). Stochastic gradient boosting. Tech. rep., Stanford University.

Ganitkevitch, J., Van Durme, B., & Callison-Burch, C. (2013). PPDB: The paraphrase database. In *Proceedings of North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*.

Genesereth, M. R., & Nilsson, N. J. (1987). *Logical foundations of artificial intelligence*. Morgan Kaufman, San Mateo, CA.

Getoor, L., & Taskar, B. (2007). *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA.

Geurts, B. (2007). Existential import. In Comorovski, I., & van Heusinger, K. (Eds.), *Existence: syntax and semantics*, pp. 253–271. Springer, Dordrecht.

Gogate, V. (2009). *Sampling Algorithms for Probabilistic Graphical Models with Determinism*. Ph.D. thesis, University of California, Irvine.

Gogate, V. (2014). IJGP-sampling and SampleSearch.. Software available at `http://www.hlt.utdallas.edu/~vgogate/ijgp-samplesearch.html`.

Gogate, V., & Domingos, P. (2011). Probabilistic theorem proving. In *Proceedings of 27th Conference on Uncertainty in Artificial Intelligence (UAI-2011)*.

Grefenstette, E. (2013). Towards a formal distributional semantics: Simulating logical calculi with tensors. In *Proceedings of Second Joint Conference on Lexical and Computational Semantics (*SEM-2013)*.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, *11*(1), 10–18.

Herbelot, A., & Vecchi, E. M. (2015). Building a shared world: Mapping distributional to model-theoretic semantic spaces. In *EMNLP 2015*.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., & Blunsom, P. (2015). Teaching machines to read and comprehend. In *Proceedings of Advances in Neural Information Processing Systems 28 (NIPS 2015)*.

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

Kamp, H., & Reyle, U. (1993). *From Discourse to Logic*. Kluwer.

Kimmig, A., Bach, S. H., Broecheler, M., Huang, B., & Getoor, L. (2012). A short introduction to Probabilistic Soft Logic. In *Proceedings of NIPS*

Workshop on Probabilistic Programming: Foundations and Applications (NIPS Workshop-2012).

Kok, S., Singla, P., Richardson, M., & Domingos, P. (2005). The Alchemy system for statistical relational AI.. `http://www.cs.washington.edu/ai/alchemy`.

Kotlerman, L., Dagan, I., Szpektor, I., & Zhitomirsky-Geffet, M. (2010). Directional distributional similarity for lexical inference.. *Natural Language Engineering*, *16*(4), 359–389.

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

Lai, A., & Hockenmaier, J. (2014). Illinois-lh: A denotational and distributional approach to semantics. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pp. 329–334, Dublin, Ireland. Association for Computational Linguistics and Dublin City University.

Landauer, T., & Dumais, S. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition, induction, and representation of knowledge. *Psychological Review*, *104*(2), 211.

Lewis, M., & Steedman, M. (2013). Combined distributional and logical semantics. *Transactions of the Association for Computational Linguistics (TACL-2013)*, *1*, 179–192.

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

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

Lund, K., & Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, and Computers*, *28*(2), 203–208.

MacCartney, B., & Manning, C. D. (2007). Natural logic for textual inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pp. 193–200.

MacCartney, B., & Manning, C. D. (2008). Modeling semantic containment and exclusion in natural language inference. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling-2008)*.

MacCartney, B., & Manning, C. D. (2009). An extended model of natural logic. In *IWCS 2009*.

Marelli, M., Menini, S., Baroni, M., Bentivogli, L., Bernardi, R., & Zamparelli, R. (2014). A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*.

Mateescu, R., Kask, K., Gogate, V., & Dechter, R. (2010). Join-graph propagation algorithms. *Journal of Artificial Intelligence Research*, *37*(1), 279–328.

Metzler, D. P., Noreault, T., Richey, L., & Heidorn, B. (1984). Dependency parsing for information retrieval. In *Proceedings of the 7th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 313–324.

Mitchell, J., & Lapata, M. (2010). Composition in distributional models of semantics. *Cognitive Science*, *34*(3), 1388–1429.

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

Natarajan, S., Khot, T., Lowd, D., Tadepalli, P., Kersting, K., & Shavlik, J. (2010). Exploiting causal independence in Markov logic networks: Combining undirected and directed models. In *Proceedings of European Conference in Machine Learning (ECML-2010)*.

Ng, D., & Curran, J. R. (2012). Dependency hashing for n-best ccg parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL-2012)*.

Parsons, T. (1990). *Events in the semantics of English*. MIT press, Cambridge, MA, USA.

Pasupat, P., & Liang, P. (2015). Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL 2015)*.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA.

Poon, H., & Domingos, P. (2006). Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, Boston, MA.

Poon, H., & Domingos, P. (2008). Joint unsupervised coreference resolution with markov logic. In *In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2008)*.

Princeton University (2010). About WordNet.. `http://wordnet.princeton.edu`.

Reddy, S., Täckström, O., Collins, M., Kwiatkowski, T., Das, D., Steedman, M., & Lapata, M. (2016). Transforming Dependency Structures to Logical Forms for Semantic Parsing. *Transactions of the Association for Computational Linguistics*, *4*, 127–140.

Richardson, M., Burges, C. J., & Renshaw, E. (2013). Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Pro-*

ceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2013).

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning*, *62*, 107–136.

Riedel, S., Chun, H.-W., Takagi, T., & Tsujii, J. (2009). A markov logic approach to bio-molecular event extraction. In *In Proceedings of the Natural Language Processing in Biomedicine NAACL 2009 Workshop (BioNLP 2009)*.

Riedel, S., & Meza-Ruiz, I. (2008). Collective semantic role labelling with Markov logic. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning (CoNLL'08)*, pp. 193–197.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM*, *12*(1), 23–41.

Roller, S., Erk, K., & Boleda, G. (2014). Inclusive yet selective: Supervised distributional hypernymy detection. In *Proceedings of the Twenty Fifth International Conference on Computational Linguistics (COLING-2014)*.

Sadrzadeh, M., Clark, S., & Coecke, B. (2013). The Frobenius anatomy of word meanings I: subject and object relative pronouns. *Journal of Logic and Computation*, *23*(6), 1293–1317.

Schoenick, C., Clark, P., Tafjord, O., Turney, P., & Etzioni, O. (2016). Moving beyond the turing test with the allen ai science challenge. In *CACM 2016*.

Skolem, T. (1920). Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze. *Skrifter utgit av Videnskapselskapet i Kristiania*, *4*, 4–36.

Strawson, P. F. (1950). On referring. *Mind*, *59*, 320–344.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112.

Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. In *ACL 2015*.

Tian, R., Miyao, Y., & Takuya, M. (2014). Logical inference on dependency-based compositional semantics. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, Baltimore, MD.

Towell, G. G., & Shavlik, J. W. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence*, *70*, 119–165.

Tse, D., & Curran, J. R. (2010). Chinese CCGbank: extracting CCG derivations from the Penn Chinese Treebank. In *Proceedings of the Twenty Third International Conference on Computational Linguistics (COLING-2010)*.

Turney, P., & Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, *37*(1), 141–188.

van Eijck, J., & Lappin, S. (2012). Probabilistic semantics for natural language. In *Logic and interactive rationality (LIRA) yearbook*. Amsterdam dynamics group.

van Eijck, J., & Unger, C. (2010). *Computational Semantics with Functional Programming*. Cambridge University Press.

Venugopal, D., & Gogate, V. (2013). GiSS: Combining SampleSearch and Importance Sampling for inference in mixed probabilistic and deterministic graphical models. In *Proceedings of Association for the Advancement of Artificial Intelligence(AAAI-13)*.

Zelle, J. M., & Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pp. 1050–1055, Portland, OR.

Zirn, C., Niepert, M., Stuckenschmidt, H., & Strube, M. (2011). Fine-grained sentiment analysis with structural features.. In *Proceedings of the The 5th International Joint Conference on Natural Language Processing (IJCNLP 2011)*, pp. 336–344.

Vita

Islam Beltagy was born in 1986 in Alexandria, Egypt. He studied computer engineering at Alexandria University. He received his Bachelor of Engineering in 2008 and was ranked the first among his batch. He obtained his Master of Science degree in Computer Science at Alexandria University in 2011 working on routing in cognitive networks. He was also working in an Alexandria-based software company. He obtained a Master of Science degree in Computer Science at the University of Texas at Austin in 2014. He then continued his doctoral study at the Department of Computer Science, the University of Texas at Austin, where he has been working in natural language understanding. After graduation, he will be joining Quora in Mountain View starting in September, 2016.

Permanent address: beltagy@cs.utexas.edu

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.