Knowledge Representation, Reasoning and Learning for Non-Extractive Reading Comprehension

by

Arindam Mitra

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved September 2019 by the
Graduate Supervisory Committee:

Chitta Baral, Chair
Joohyung Lee
Yezhou Yang
Murthy Devarakonda

ARIZONA STATE UNIVERSITY

December 2019

ABSTRACT

While in recent years deep learning (DL) based approaches have been the popular approach in developing end-to-end question answering (QA) systems, such systems lack several desired properties, such as the ability to do sophisticated reasoning with knowledge, the ability to learn using less resources and interpretability. In this thesis, I explore solutions that aim to address these drawbacks.

Towards this goal, I work with a specific family of reading comprehension tasks, normally referred to as the Non-Extractive Reading Comprehension (NRC), where the given passage does not contain enough information and to correctly answer sophisticated reasoning and "additional knowledge" is required. I have organized the NRC tasks into three categories. Here I present my solutions to the first two categories and some preliminary results on the third category.

Category 1 NRC tasks refer to the scenarios where the required "additional knowledge" is missing but there exists a decent natural language parser. For these tasks, I learn the missing "additional knowledge" with the help of the parser and a novel inductive logic programming. The learned knowledge is then used to answer new questions. Experiments on three NRC tasks show that this approach along with providing an interpretable solution achieves better or comparable accuracy to that of the state-of-the-art DL based approaches.

The category 2 NRC tasks refer to the alternate scenario where the "additional knowledge" is available but no natural language parser works well for the sentences of the target domain. To deal with these tasks, I present a novel hybrid reasoning approach which combines symbolic and natural language inference (neural reasoning) and ultimately allows symbolic modules to reason over raw text without requiring any translation. Experiments on two NRC tasks shows its effectiveness.

The category 3 neither provide the "missing knowledge" and nor a good parser. This

thesis does not provide an interpretable solution for this category but some preliminary results and analysis of a pure DL based approach. Nonetheless, the thesis shows beyond the world of pure DL based approaches, there are tools that can offer interpretable solutions for challenging tasks without using much resource and possibly with better accuracy.

DEDICATION

To Ma (Mitali Mitra), Bapi (Amaresh Mitra) and Choma (Priti Ghosh)

ACKNOWLEDGMENTS

Before starting my PhD I was probably good at solving algorithm or tricky Math problems. However, PhD was different. Most of the problems that most of us work on during our PhD probably won't be "solved" in a long time. How to deal with such "hard" problems, organize and make progress is something among many other things that I have learned from my advisor Dr.Chitta Baral to whom I am greatly thankful. Without his support and continuous faith in me I would not have enjoyed my PhD this much. I am also thankful to Dr.Peter Clark for giving me opportunities to work closely with a group of experts on my most favorite topic of *Science Question Answering* and helping me to think big.

I would also like to express my sincerest respect to my school teachers, specially Dulali didimuni, Tarun sir, Asim sir, Jayashree didimuni, Aniruddha sir, Subrata sir, Lakhi didimuni, Manu Didimuni, Rina Didimuni, Pramila didimuni, Malay Sir, Lekhashri Didimuni, Deepti Didimuni, Santosh sir, Ranjit sir, Jhuna sir, Bhupati sir, Bulu didimuni, Dulal sir, Nirumohon sir, Dulal mama and A.C. sir for the unbounded love and care that I have received from them.

The two people who have eagerly waited for this day and are more excited than me, are my parents. I do not have enough words to express how much indebted I am to them and the members of my big family, chhoma, boro mama, buchu bua, jhuma bua, bhai mama, fulda mama, manu mesho, babua mesho, raju mesho, mashimoni, mami, deepali mami, shuli di, dada bhai, appalu da, kaustav, bapan, bhai, mouri, munmun masi and the countless people of my birthplace, who throughout my entire life have shown utmost care and affection.

I have been very lucky to have a good number of best friends Abhiman, Paplu, Subho, Papa, Ani who are there for me at any point of time. They continuously teach me what not to do by making enormous mistakes themselves and I cannot be more grateful for that. The last one among them has requested (read forced) me to apply for PhD and deserve a special

thanks as I have really enjoyed this journey of five years of study, sports and long vacations and would have continued for some more time unless she again “requested” me to graduate soon.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

Table Page

Table Page

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

1.1 Motivation

The field of Natural Language Understanding is going through an important phase. In the last five years, a significant number of datasets have been developed at an unprecedented pace targeting different flavors of "question-answering" (QA) such as answering multiple-choice questions from 8^{th}-grade science textbooks (Clark and Etzioni 2016; Clark et al. 2018), math word problems (Hosseini et al. 2014; Koncel-Kedziorski et al. 2015), answering questions requiring qualitative reasoning (Tafjord et al. 2019), open-book question answering (Mihaylov et al. 2018a), commonsense question answering (Levesque, Davis, and Morgenstern 2012; Sakaguchi et al. 2019; Bhagavatula et al. 2019; Sap, Rashkin, et al. 2019), answering questions from text which describes a processes (Tandon et al. 2018) or answering simple look-up style or multi-hop questions over Wikipedia articles (Rajpurkar et al. 2016; Weston, Chopra, and Bordes 2014). Keeping up with the pace, numerous QA systems have been developed. However, unlike the diversity that exists in the datasets, the proposed QA systems are mostly deep neural nets that are designed specifically for the question-answering task at hand. Is it because the other approaches such as the ones that use knowledge representation and reasoning, are not good enough to handle the real world challenges? Is it worth practicing knowledge representation and reasoning when it comes to building question-answering systems? Or shall we keep aside the ideologies of knowledge representation and reasoning and keep exploring bigger and better engineered neural networks to produce the next generation QA systems?

Deep neural networks are the most popular solutions for many QA tasks mainly because of two reasons. First, they are good learners. They can produce surprisingly good results without any feature engineering. Even if the accuracy is not close to the human level accuracy, one can see a decent accuracy with comparatively little engineering effort if there exists a good amount of training data. Second, they can be easily deployed. For most QA tasks if not all, one can build and train a neural network quite quickly. However, even though the neural networks possess these important properties they lack several important other ones.

Arguably the most important feature that they lack is that of interpretability. It is almost impossible to answer in layman terms what has a deep neural net learned from the end task and how is it storing that knowledge or how is it using that learned knowledge to answer the questions. Interpretability of question-answering models is important not only for our curiosity, but also to address the models' failures. When a model fails to answer a question correctly, it is crucial to understand how is it trying to answer the question, what is going wrong and how to fix it. Is it the case that the representation and reasoning schemes are not flexible enough to deal with the target task? Is it the case that learning algorithm is discovering dataset specific ques that are not robust enough? Or is it the case that the training data or the available knowledge base is not enough, and it needs to more knowledge to answer the given question? If our models are not interpretable it is difficult to understand the root cause and make proper amendments. Another important drawback is their source of learning. Neural Networks cannot learn from testimony. One must gather annotated examples to teach a neural net something. However, testimony is a valuable source of knowledge. If one wants to teach the concept of "middle of a sequence" to a QA system, one should be able to do so just by describing its definition.

The knowledge representation and reasoning community even though have not provided

much effort to learning from data, they have devoted an enormous amount of attention towards interpretability and learning from testimony (McCarthy 1960; Daniel G Bobrow 1964; Green 1969; Simmons 1970; Charniak 1972; Winograd 1972; Bobrow and Winograd 1977; Perrault and Allen 1980; Balduccini, Baral, and Lierler 2008). Several tools and knowledge representation schemes have been developed over the years with which one can represent a question-answering problem, represent the required knowledge and the reasoning scheme and solve it in a declarative manner. One natural question that arises is that can we take these tools and modify and build more so that the knowledge representation and reasoning based QA systems without losing their interpretability or the ability to reason with knowledge 1) can also learn from data 2) can also be easily deployed for a broad variety of QA problems and 3) can also obtain comparable accuracy to that of state-of-the-art neural networks? If the answer to this question is yes, then certainly practicing knowledge representation and reasoning for question-answering is beneficial. In this thesis, I try to find an affirmative answer to this question while working on "Non-Extractive Reading Comprehension"- a family of QA tasks where world knowledge and reasoning play important roles.

1.2 Non-Extractive Reading Comprehension

Reading Comprehension (RC) in general is the task of answering questions with respect to a given passage. Figure 1 shows a sample reading comprehension problem. The answer to the question has been highlighted in red. Please see that to answer the question, one needs to understand only a small portion of the text (the highlighted text). This kind of reading comprehension where the answer is a span of the given passage and there is a small portion of the passage which directly answers the question is normally referred to as

The Normans (Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in the 10th and 11th centuries gave their name to Normandy, a region in France. They were descended from Norse ("Norman" comes from "Norseman") raiders and pirates from Denmark, Iceland and Norway who, under their leader Rollo, agreed to swear fealty to King Charles III of West Francia.

Q: In what country is Normandy located?
: France

Figure 1: An example of a (extractive) reading comprehension task that is taken from SQUAD 2.0 dataset.

Extractive Reading Comprehension. However not for all reading comprehension questions the answer is a span or there exists a small portion containing the answer. Figure 2 shows some examples of those more challenging kind.

In figure 2a, the answer i.e. "leaf" appears in the passage however to compute the answer one needs to reason with outside knowledge. For e.g., one needs to understand that all the participants which are undergoing a reaction to produce sugar are in the leaf. This requires the knowledge of event effects and inertia. Secondly, the commonsense knowledge that if all the materials that undergone a reaction to produce sugar were at leaf, the outcome sugar would normally be in the leaf is also needed. For the questions in 2b one needs to track states and reason with the effects of events such as "grabbing", "dropping" which is not given in the passage. For the questions in 2c one needs to understand the meaning of "indicates". Frog breathes with lungs in both the froglet and the adult stage. Thus knowing a frog has lungs does not allow us to determine which stage it is now. Option (B) is thus the correct answer. Problem in figure 2d requires the knowledge of arithmetic operators and formulas. Problem in figure 2e requires the knowledge of how increasing/decreasing one physical entity such as friction, speed, heat, affects others. I would be referring to this kind of problems where there might not be a small part of text which directly answers the question

Chloroplasts in the leaf of the plant trap light from the sun. The roots absorb water and minerals from the soil. This combination of water and minerals flows from the stem into the leaf. Carbon dioxide enters the **leaf**. Light, water and minerals, and the carbon dioxide all combine into a mixture. This mixture forms **sugar** (glucose) which is what the plant eats.

Q: Where is sugar produced?
A: in the leaf

(a) A paragraph from *ProPara* about photosynthesis. Processes are challenging because questions often require state tracking.

Mary grabbed the football.
Mary traveled to the office.
Mary took the apple there.
What is Mary carrying?
A:football,apple
Mary left the football.
Daniel moved to the bedroom.
What is Mary carrying?
A:apple

(b) A paragraph from Task 8 of *bAbI* corpus.

Life Cycle Of A Frog

egg - Tiny frog eggs are laid in masses in the water by a female frog. The eggs hatch into tadpoles.
tadpole - (also called the polliwog) This stage hatches from the egg. The tadpole spends its time swimming in the water, eating and growing. Tadpoles breathe using gills and have a tail.
tadpole with legs - In this stage the tadpole sprouts legs (and then arms), has a longer body, and has a more distinct head. It still breathes using gills and has a tail.
froglet - In this stage, the almost mature frog breathes with lungs and still has some of its tail.
adult - The adult frog breathes with lungs and has no tail (it has been absorbed by the body).

Q: What best indicates that a frog has reached the adult stage? (A) When it has lungs (B) When its tail has been absorbed by the body (Ans. **B**)

(c) Frog life-cycle and a sample question from it which requires the knowledge of *indicate*.

Carrie has 125 U.S. stamps. She has 3 times as many foreign stamps as U.S. stamps. How many stamps does she have altogether?
A: $x = 125 + 3 \times 125 = 500$

(d) An Arithmetic Word Problem

The propeller on Kate's boat moved slower in the ocean compared to the river. This means the propeller heated up less in the (A) ocean (B) river (Ans. **B**)

(e) A qualitative word problem

Figure 2: A variety of Non-Extractive Reading Comprehension questions from 5 different datasets. All of these questions require reasoning with knowledge which is missing in the passage and the answer cannot be looked up from a small part of the passage.

Which of these would let the most heat travel through?
A) a new pair of jeans.
B) a steel spoon in a cafeteria.
C) a cotton candy at a store.
D) a calvin klein cotton hat.

(a) A question from Mihaylov et al. 2018a. Along with the question a "book" containing a set of knowledge sentences is given which is not sufficient to answer the question. The task is to find out the useful knowledge from the book and missing knowledge from external sources to answer the question.

Obs1: It was a gorgeous day outside.
Obs2: She asked her neighbor for a jump-start.
Hyp1: Mary decided to drive to the beach, but her car would not start due to a dead battery.
Hyp2: It made a weird sound upon starting.

(b) A sample task from (Bhagavatula et al. 2019) requiring abductive reasoning. The task is to decide which of *hyp1* and *hyp2* fits between *obs1* and *obs2*, which in turn requires commonsense knowledge about world.

In the school play, Robin played a hero in the struggle to the death with the angry villain. How would others feel as a result?
a) sorry for the villain
b) hopeful that Robin will succeed
c) like Robin should lose the fight

(c) sample task from (Sap, Rashkin, et al. 2019) requiring reasoning about social interactions, which in turn requires commonsense knowledge about state of person executing the action and its effects on the executor or the other participants of the event.

You need to break a window. Which object would you rather use?
a) a metal stool
b) a giant bear
c) a bottle of water

(d) Question from a dataset containing problems requiring naive physics reasoning focusing on how we interact with everyday objects in everyday situations. This dataset focuses on what actions each physical object affords and what physical interactions a group of objects afford (e.g., it is possible to place an apple on top of a book, but not the other way around).

Figure 3: A variety of Non-Extractive Reading Comprehension (multiple choice) questions from 4 different datasets. All of these questions require reasoning with commonsense knowledge which is missing in the passage.

and one might need to use additional knowledge and sophisticated reasoning to derive the answer as *Non-Extractive Reading Comprehension* (NRC). In this thesis I aim to develop energy-efficient and interpretable solutions for *Non-Extractive Reading Comprehension* tasks.

1.3 Research Summary

1.3.1 Categorization of the Non-Extractive Reading Comprehension Tasks

In the quest to develop knowledge representation and reasoning based question-answering frameworks that also learns from data and is as easy to deploy as typical deep neural networks, I have categorized the Non-Extractive Reading comprehension tasks into three classes:

Category 1 Missing Knowledge In this category, we have a natural language parser such as AMR, QASRL or Dependency parser, which works well for the sentences pertaining to the task at hand. However, the knowledge which is required to answer the questions such as event effects is not present in any existing knowledge corpus and is very large in amount and thus cannot be handwritten. Based on the current state of research, the tasks in Figure 2a, 2b and 2d are examples of this kind.

Category 2 Missing Parser In this category, we do not have a natural language parser which works well for the sentences pertaining to the task at hand. However, the missing knowledge which is required to answer the questions is present in some existing knowledge corpus or is very small in amount and thus could be handwritten. The tasks in Figure 2c and 2e are examples of this kind.

Category 3 Missing both Knowledge & Parser In this category, we do not have a natural language parser which works well for the sentences pertaining to the task at hand. Also, We do not have most of the required knowledge in any existing knowledge corpus. The tasks in Figure 3a, 3b, 3c and 3d are examples of this kind.

1.3.2 Sketch of the Approach

To build interpretable solutions for Non-Extractive Reading Comprehension (NRC) tasks, I have mostly relied on the tools from the Knowledge Representation and Reasoning (KR) community and have developed new tools and introduced new modules to existing KR based question answering architectures. In this dissertation, I present an interpretable and energy-efficient solution for both Category 1 [Missing Knowledge] and Category 2 [Missing Parser] NRC tasks. I have started investigating interpretable solutions for Category 3 NRC tasks. My current findings and preliminary results are also part of this dissertation.

1.3.2.1 Approach For Category 1

For Category 1 NRC tasks, I have used pure Knowledge Representation and Reasoning based approaches. There are two key challenges that one needs to solve to employ a KR based approach. First challenge comes from the requirement that the passage and the question has to be given as a set of formal statements. Since, for category 1 NRC tasks we have a suitable parser, this challenge can be easily overcome. The second more critical challenge is known as "knowledge bottleneck". KR based approaches demonstrate strong reasoning ability which is suitable for NRC tasks. However, for the KR based approach to draw inferences, it needs inference enabling knowledge. For e.g. given the passage "John picked an apple. John went to kitchen." and the question "Where is the apple?" a KR based QA system would not be able to infer that the answer is "kitchen" unless it has access to the commonsense knowledge that "if someone moves to a new location while holding an object, the object also moves to the new location". Unfortunately for most of the tasks there does not exist a knowledge base which contains the suitable inference enabling

knowledge. The question that we ask then is: can we build an *efficient* machine learning algorithm that learns such knowledge from the question answering dataset at hand. It may be noted that algorithms of this kind are known as Inductive Logic Programming (ILP) (Muggleton 1991) algorithms and their existence goes way back to 1990. However, when comes to practice, existing algorithms hardly scale to modern QA datasets. In this thesis, I describe the issues of existing ILP formulation and propose a new ILP formulation which is better suited for supervised machine learning and finally describe a novel inductive logic programming algorithm that learns incrementally and in an iterative manner. The code for the learning algorithm is publicly available from: https://github.com/ari9dam/ILPME.

Figure 9 and 10 show the overall architecture of the question answering system developed to deal with the Category 1 NRC tasks. In the training phase, (Figure 9) some background knowledge is given (could be empty) along with some information about "what needs to be learned" (e.g. the name of the predicates). The system then converts the texts to some application specific representation and creates constraints (can be hard or weak) from the question and answer pairs in the training data. Finally the ILP algorithm learns the missing knowledge that when added to the system can infer the answers. The system then uses the learned knowledge along with the background knowledge, the text and the question parsers to answer an unknown question during test phase (Figure 10).

To differentiate the knowledge based Question Answering approach where the rules are learned from annotated dataset from the traditional approach where the rules are handwritten, I will refer to the approach in Figure 9 & 10 as LKR (the additional 'L' merely signifies the presence of a learning component). The LKR paradigm has been used to develop question answering systems on three recent Category 1 NRC tasks namely bAbI (Weston et al. 2015), word arithmetic problem solving (Hosseini et al. 2014; Koncel-Kedziorski et al. 2015; Roy,

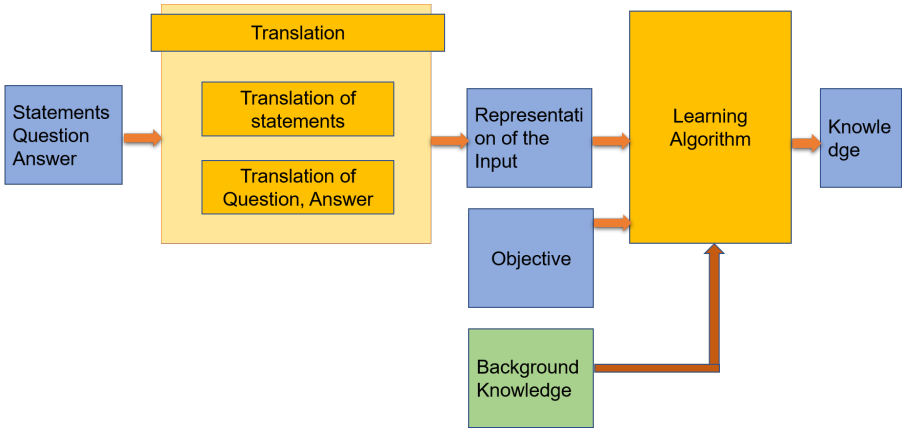

Figure 4: The modules that are involved in the training phase of LKR framework

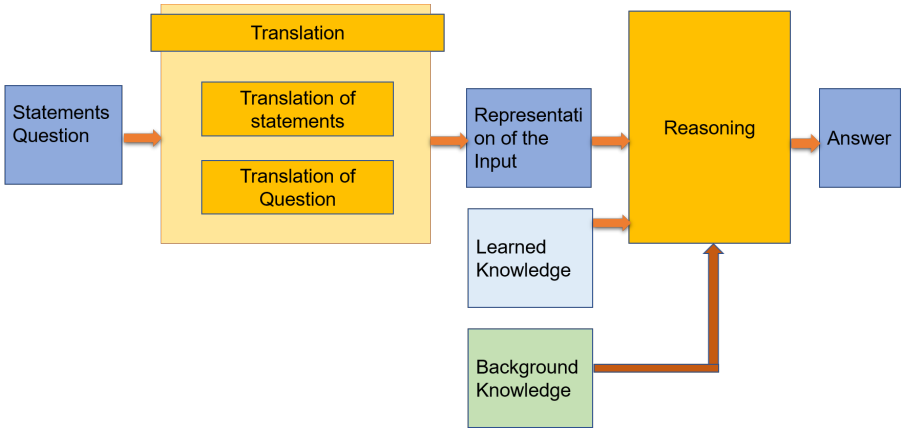

Figure 5: The modules that are involved in the test phase of LKR framework

Vieira, and Roth 2015; Roy and Roth 2015) and Process Paragraph Comprehension (Tandon et al. 2018). In the first two cases the system has achieved state-of-the-art performance beating the deep neural solutions and in the third dataset the difference with the state-of-the-art performance is statistically insignificant. This shows that it is worthwhile to investigate LKR paradigm based solutions for category 1 tasks where we have a suitable parser.

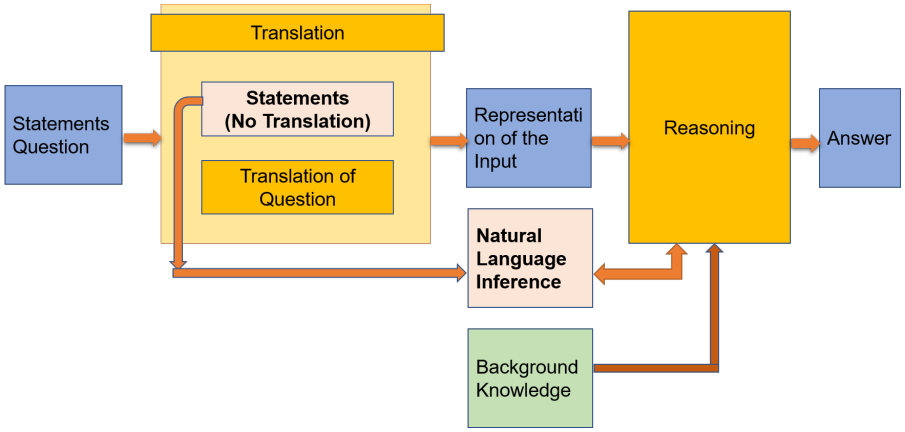

Figure 6: The modules that are involved in solving Category 2 NRC tasks

1.3.2.2 Approach For Category 2

For Category 2 NRC tasks, standard KR or the proposed LKR paradigms would not work, as both require the input passage and the question to be given as formal statements. However, fortunately for the Category 2 questions the missing inference enabling knowledge is available from some source. With that in mind, I have developed a novel architecture as shown in Figure 6 that integrates Natural Language Inference with Symbolic Inference to deal with the missing parser scenario of Category 2.

Given a passage and a question the system first parses the question with a semantic parser that has been trained for the task. The parsed question along with the original passage is given to the symbolic reasoning engine, which draws inference based on the available knowledge and makes calls to the Natural Language Inference (NLI) module, if any hypothesis (a statement in natural language) needs to be verified from the given passage. In my experiments I have used a neural Natural Language Inference system, which is trained with existing NLI datasets.

For example, for the question, *What best indicates that a frog has reached the adult*

stage? in Figure 2c, a semantic parser first parses the question into $qIndicator(frog, adult)$ to express that the question type is *indicator* and asks which of option A and option B indicates that *frog* is in the *adult* stage. The background knowledge contains the definition that an option is a better indicator of a stage if it is true in only that stage but not in other stages. Using this knowledge and the answer options the system will generate several hypothesis like, '*frog* in the *froglet* stage *breathes with lungs*', '*frog* in the *adult* stage *breathes with lungs*'. This hypothesis will then be given to a Natural Language Inference system which will return its confidence about the hypothesis being true. Those confidence scores will be used to compute a score for the options. The option with the highest score will be returned as the correct answer. In this way, the system will reason with knowledge without requiring the translation of the input passage. I also show that the need of a custom semantic parser to get the question representation can be avoided by using a Natural Language Inference system instead. The basic idea is similar, generate controlled natural language descriptions of the possible question predicates and see which one is better entailed with respect to the actual given question.

To differentiate the knowledge based Question Answering approach where the premises are given as natural language statements from the traditional approach where the premises are given as formal statements, I will refer to the approach in Figure 6 as TKR (the additional 'T' merely signifies the presence of text in the knowledge base).

1.3.2.3 Approach For Category 3

This dissertation does not provide a solution for Category 3 tasks. However, I have done some analysis to evaluate Natural Language Inference and the available benchmarks, which I describe here.

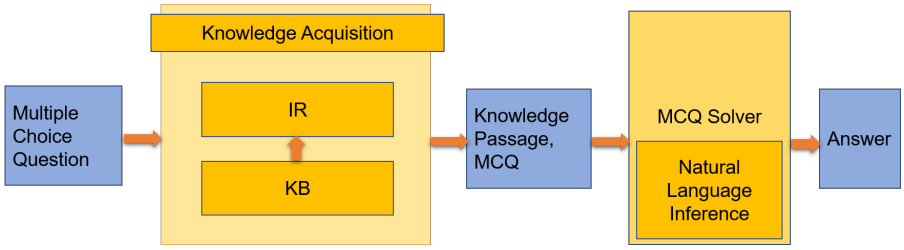

Figure 7: Standard approach for Category 3 NRC tasks using NLI

Natural Language Inference provides a bridge for symbolic reasoning in the absence of a natural language translation system as shown in Figure 6 and is going to play a key role for the more general setting of Category 3. To evaluate their performance further, I have selected four Category 3 question answering tasks (Figure 3) which require commonsense knowledge to answer the questions, which is hard to get from existing resources. Until now I have explored only the standard NLI based approach (Figure 7) that extracts some knowledge from existing corpora given a multiple choice question and then scores each of the answer choices with respect to the extracted knowledge using a Natural Language Inference system. The goal of this experiment is to see how useful existing knowledge corpora are and how effective NLI is for commonsense reasoning. It is part of my future work to introduce KR components in this naive opaque architecture to make it more transparent.

Below is the list of publications reflecting the contributions of this dissetation:

- Mitra, Arindam, and Chitta Baral. *Addressing a Question Answering Challenge by Combining Statistical Methods with Inductive Rule Learning and Reasoning*. AAAI. 2016.
- Mitra, Arindam, and Chitta Baral. *Incremental and Iterative Learning of Answer Set Programs from Mutually Distinct Examples.* Theory and Practice of Logic Programming, vol. 18, no. 3-4, 2018, pp. 623–637., doi:10.1017/S1471068418000248.
- Mitra, Arindam, and Chitta Baral. *Learning to use formulas to solve simple arith-*

metic problems. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vol. 1. 2016.

- Mitra, Arindam, and Chitta Baral. *Solving General Arithmetic Problems with Answer Set Programming and Inductive Logic Programming*.[Under Review]
- Mitra, Arindam, Bhattacharjee, Aurgo and Chitta Baral. *Learning Interpretable Models of Actions for Tracking State Changes in Procedural Text*. [Under Review]
- Mitra, Arindam, Peter Clark, Oyvind Tafjord, and Chitta Baral. *Declarative Question Answering over Knowledge Bases containing Natural Language Text with Answer Set Programming.* AAAI, 2019.
- Mitra, Arindam, Chitta Baral, Aurgho Bhattacharjee, and Ishan Shrivastava. *A Generate-Validate Approach to Answering Questions about Qualitative Relationships.* [Under Review]
- Arindam Mitra, Banerjee, Pratyay, Kuntal Kumar Pal, and Chitta Baral. *Careful Selection of Knowledge to solve Open Book Question Answering.* Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019.
- Arindam Mitra, Pratyay Banerjee, Kuntal Pal, Swaroop Mishra and Chitta Baral, *Exploring ways to incorporate additional knowledge to improve Natural Language Commonsense Question Answering*. [Under Review]

1.4 Contributions of the Research

In this dissertation I have shown that it is possible to build a knowledge representation and reasoning based question-answering system for a broad variety of domains which have the following properties: 1) it can learn from both annotated data and testimony, 2) it can reason with declarative knowledge 3) is interpretable and 4) achieves better or comparable

accuracy to that of a state-of-the-art neural network without much engineering effort. To achieve this,

- I have developed a novel Inductive Logic Programming algorithm that learns interpretable knowledge from the training data containing question-answer pairs. This has allowed me to build transparent solutions for the question-answering domains for which a reasonable natural language parser is available, Prior to my solution no Inductive Logic Programming algorithm could scale to the modern QA datasets. I have explained the reason and proposed an efficient solution. The implementation of the learning algorithm is also made publicly available at https://github.com/ari9dam/ILPME.
- I have proposed a lightweight solution for declarative programming with knowledge bases containing text. Prior to my solution declarative problem solving required translation of the input text into a formal meaning representation, which significantly limited its application, as for many reading comprehension tasks the associated passage cannot be translated properly with most of the existing parsers. However utilizing the recent advances in Natural Language Inference, I have been able to eliminate such a need.

I have additionally explored the performance of Natural Language Inference (NLI) models and the coverage of some suitable commonsense knowledge bases for a set of multiple-choice commonsense tasks for which neither the necessary knowledge nor a suitable natural language parser is available. This exercise has enabled me to observe how NLI systems perform on these datasets. Furthermore, this exercise has also helped me to identify that the benchmark datasets indeed contain a lot of problems for which the existing commonsense knowledge bases are not sufficient.

1.5 Organization of the Thesis

The rest of the dissertation is organized as follows:

Chapter 2 talks about the tools that I will be using in the proposed solutions and describes some parallel efforts for using external knowledge in deep neural nets.

Chapter 3 shows how to apply LKR paradigm to the bAbI question answering challenge (Figure 2b) with an existing ILP algorithm namely XHAIL (Ray 2009) and discusses the issues with the learning algorithm.

Chapter 4 addresses the scalability issue of the existing ILP algorithms and presents a sound and complete Inductive Logic Programming algorithm that can learn from large datasets.

Chapter 5 shows how to apply the LKR paradigm to solve word arithmetic problems with the proposed Inductive Logic Programming algorithm. Particularly, it shows how a machine can learn from data which operations or formulas to apply and in which order to incrementally construct an equation.

Chapter 6 shows how to apply the LKR paradigm for the task of Process Paragraph (ProPara) comprehension (Figure 2a) with the proposed Inductive Logic Programming algorithm. Understanding text describing a process that involves actions is particularly challenging for NLP, because knowledge about those actions, and how the world changes as a result of them, are often unstated in text. At the same time, the knowledge representation and reasoning (KR) community has developed effective techniques for modeling and reasoning about actions, but integrating them to understand realistic natural language text has remained elusive. This chapter shows how such an integration can be achieved with the LKR paradigm.

Chapter 7 describes the proposed solution for Category 2 NRC tasks, namely the TKR

paradigm with the life cycle question answering task (Figure 2c) where obtaining a good formal representation of the text is difficult.

Chapter 8 shows how the TKR paradigm can be applied to solve qualitative word problems (Figure 2e). This chapter also shows how the use of a semantic parser for question interpretation can be avoided with the help of Natural Language Inference and how it opens up the opportunity for transfer learning.

Chapter 9 & 10 describes the NLI based architecture for the commonsense question answering tasks in Figure 3.

chapter 11 describes some future directions and concludes the thesis.

Chapter 2

BACKGROUND

2.1 Reasoning

2.1.1 Answer Set Programming

In this thesis, I have used the language of Answer Set Programming (Gelfond and Lifschitz 1988; Baral 2003) to represent and reason with Knowledge. The decision to use Answer Set Programming as the primary choice for representing and reasoning knowledge is influenced by main Three reasons. First, most of our commonsense knowledge have defaults and exceptions. For e.g., we would normally assume that when someone is making something with some raw materials, the product at the end of the process will normally be at the same place to that of the raw materials. If you brought all the pieces of your bike to assemble in a place the final version of the bike will also be in that place. If tree gathered all water, mineral and light in its leaf to produce the food, then the produced food will probably be in the leaf. However this is not true in every scenario. For. e.g., if you are heating water, then the vapor leaves the container and moves to atmosphere as soon as gets created. A lot of the knowledge that we have or that our system learns by going through the question-answer pairs are of this kind. Answer Set Programming provides a straight-forward way to represent the defaults i.e. the normally scenarios and their exceptions, and thus is a suitable candidate my research work. Second, there exists efficient solvers for Answer Set Programs which makes it practical. An active body of researchers are using Answer Set Programming in industry-level applications and producing efficient and better tools which

makes it promising. Finally, Answer Set Programming supports calling external functions (for e.g. a neural network) which is useful while working over Text.

There exists very lucid tutorials and textbooks on Answer Set Programming. So I won't go through the syntax and semantics of the language here. Interested readers can go through any of the following resources (Gebser et al. 2012; Gelfond and Kahl 2014) to learn about Answer Set Programming. In each chapter, where I use ASP, I describe the relevant syntax and semantics to make the chapters self-contained.

2.1.2 Natural Language Inference

Natural language inference (NLI) (Bowman et al. 2015) is the task of determining the truth value of a natural language text, called "hypothesis" given another piece of text called "premise". The list of possible truth values include *entailment*, *contradiction* and *neutral*. *Entailment* means the hypothesis must be true as the premise is true. *Contradiction* indicates that the hypothesis can never be true if the premise is true. *Neutral* pertains to the scenario where the hypothesis can be both true and false as the premise does not provide enough information. Table 26 shows an example of each of the three cases.

Most of the existing NLI models are specially designed neural networks and normally assume that the premise and hypothesis contain a single sentence. These systems can take multi-sentence premise, but the underlying model is not suitable for multi-sentence premise.

2.2 Knowledge Infusion in Neural QA Systems

Knowledge is the key ingredient in question answering and often the given passage in the reading comprehension task does not contain enough information to answer the question.

premise: A soccer game with multiple males playing. **hypothesis:** Some men are playing a sport. **label:** Entailment.
premise: A black race car starts up in front of a crowd of people. **hypothesis:** A man is driving down a lonely road. **label:** Contradiction.
premise: A smiling costumed woman is holding an umbrella. **hypothesis:** A happy woman in a fairy costume holds an umbrella. **label:** Neutral.

Table 1: Example premise-hypothesis pairs from SNLI dataset with human-annotated labels.

As a result it is important for the QA systems to get the missing knowledge and use it while answering a question. In my work, while I try to build systems that explains and justify its answer, I have mostly relied on two options to deal with the missing knowledge 1) learn from the training data and 2) obtain the knowledge from existing knowledge base or manually write an Answer Set Program describing the knowledge. Researchers who are working on end-to-end deep neural QA systems have also developed several techniques to infuse existing knowledge. In this section I list those approaches. One can safely skip this section as I have not used this techniques in my research yet. But the hope is, while building systems that uses the good features from both the neural and symbolic approaches, knowledge about both the community will be helpful.

2.2.1 Appending Approach

When the missing knowledge is available as sentences or passages, a trivial way to add the knowledge is to add it to the original passage and hope the neural network will learn how to use it. This approach is probably the most popular one while it comes at providing knowledge to the pretrained models such as BERT (Devlin et al. 2018). However, not

all knowledge bases contain texts and knowledge exists in different formats for different purposes some of which I list here.

2.2.2 Data Augmentation

is one of the most popular knowledge infusion technique. Neural Networks are good at learning from data. Data augmentation relies on that property. The core idea here is to increase the size of annotated data by **automatically** producing more labelled data. For e.g., if you are learning a function for Natural Language Inference, you might want to add the knowledge that "X gave Y an object" should not entail that "Y gave X an object" even though both the premise and hypothesis has same set of words. The original training data might not have such examples and one can teach such phenomenon by adding more labelled data. The work of Mitra, Shrivastava, and Baral 2019 shows how to automatically get the labelled data for the NLI example.

2.2.3 Constraint-Based Learning

This is one of the recent most addition for knowledge infusion. In this case instead of learning only from the labelled question-answer $< x, y >$ pairs the function is also trained using a modified loss function that captures some constraints which encodes background knowledge. This technique has been applied to tracking an object in a video Stewart and Ermon 2017. Given a video (sequence of images i.e $< x_i^1, ..., x_i^n >$) the task is to continuously predict the height of an falling object (i.e. another sequence of numbers $< y_i^1, ..., y_i^n >$). Since any falling object follows the rule of free fall i.e $y_t = y_0 - gt^2$, the

numbers $< y_i^1, ..., y_i^n >$ cannot be random and a loss function is defined which penalizes the output of the neural network $< y_i^1, ..., y_i^n >$ if the relation $y_t = y_0 - gt^2$ is not satisfied.

2.2.4 Constraint-Based Decision Making

This is another recent most addition to the approaches that utilizes background knowledge with neural-network based decision making which was applied in the work of Tandon et al. 2018. The core idea is that instead of computing the probability of $p(y = c_i|x)$ using only a neural net, use a function that linearly combines the output of the neural network to that of a knowledge based predictor i.e $p(y = c_i|x) = \lambda p_{NN}(y = c_i|x) + (1 - lambda)p_{KB}(y = c_i|x)$.

Apart from these general techniques there has been a decent amount of work which tries to infuse specific knowledge triplets such as the ones that are present in knowledge graphs for specific architectures. Here we describe those works.

2.2.5 Producing Knowledge Aware Embedding

Every neural architecture in NLP has an embedding layer which converts the "tokens" (think of words) of the sentences to vectors, commonly known as word embeddings. These embeddings are then used by the subsequent layers which predicts the answer. The work of Yang et al. 2019 answers extractive question and the work of Mihaylov and Frank 2018 answers multiple-choice fill-in-the-blank questions, however they are quite similar in the sense that both uses vector embeddings of knowledge triples and add it to the word-embeddings to produce knowledge-aware word embeddings which are then used by the subsequent layers designed for the respective task. The work of Yang et al. 2019 uses triplets describing general knowledge such as (Donald Trump, person-leads-organization, US)

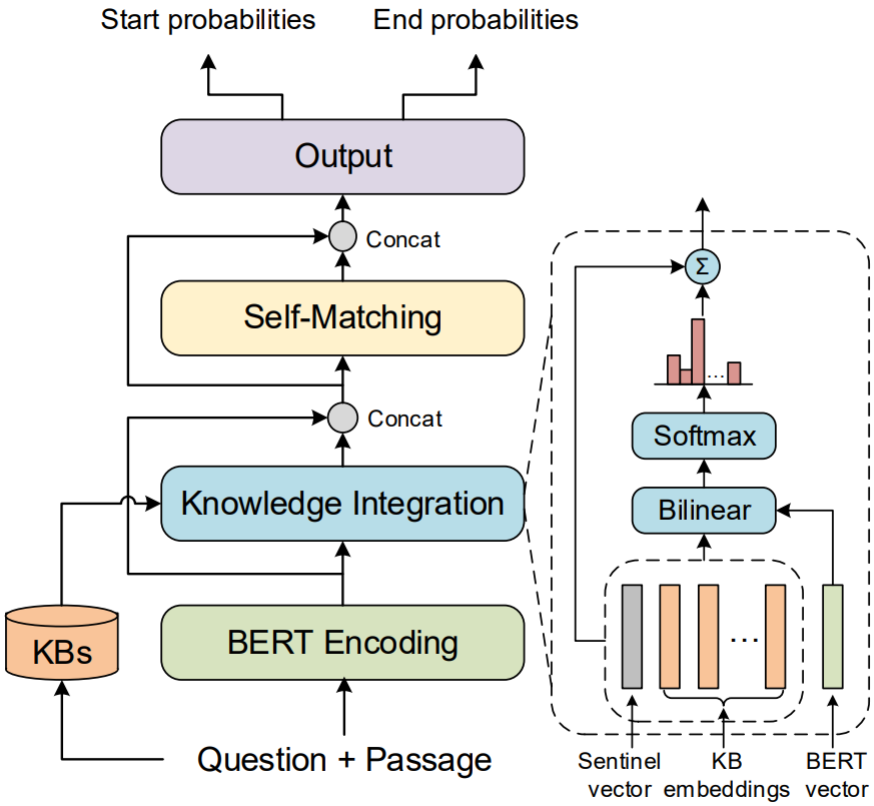

Figure 8: The architecture for producing knowledge-aware embeddings in Yang et al. 2019

whereas the work in Mihaylov and Frank 2018 uses the CenceptNet (Liu and Singh 2004) facts describing relation between two dictionary words such as (horse, isUsedFor, riding). to the QA-System. Figure 8 shows the architecture of Yang et al. 2019. Given a passage and a question, first BERT is used to obtain the embeddings of the tokens. The passage and the question is also used to get a set of knowledge triplets from the relevant Knowledge Graph and their embeddings. For each token vector, the Knowledge Integration module then computes an attention score with each of the retrieved knowledge triplet embeddings and a sentinel ("no match") vector. It then computes a weighted sum of the knowledge triplets and the sentinel vector using the attention scores as weights. The weighted-sum is appended

to the original token embedding. The enlarged token embedding is then used by the upper layers to predict the final output.

2.2.6 Knowledge-Based Matching

In NLP several tasks such as Natural Language Inference or Textual Similarity, require matching two sentences. To compute the sentence similarity most-systems first do word level matching and then aggregate those information to compute the sentence level matching. The work of (Chen et al. 2017; Wang et al. 2019) uses ConceptNet relations between word pairs in computing the word-level matching. Particularly, they define word-similarity as a function of word-embeddings based similarity and and an indicator variable which describes if the two words are related in the ConceptNet (either directly or indirectly within a certain distance) or not.

Chapter 3

LKR PARADIGM: LEARNING INFERENCE ENABLING KNOWLEDGE AND USING THEM TO ANSWER QUESTIONS

Developing intelligent agents is one of the long term goals of Artificial Intelligence. To track the progress towards this goal, several challenges have been recently proposed that employs a Question-Answering (QA) based strategy to test an agent's understanding. The *Allen Institute for AI*'s flagship project ARISTO, Richardson, Burges, and Renshaw 2013's *MCTest* and the *Winograd Schema Challenge* Levesque, Davis, and Morgenstern 2012 are all examples of this. As mentioned in the work of Weston et al. 2015, even though these tasks are promising and provide real world challenges, successfully answering their questions require competence on many sub-tasks (deduction, use of common-sense, abduction, coreference etc.); which makes it difficult to interpret the results on these benchmarks. Often the state-of-the-art systems are highly domain specific. In this light, they Weston et al. 2015 have proposed a new dataset (Facebook bAbl dataset) that put together several question-answering tasks where solving each task develops a new skill set into an agent.

In the following paragraph, I provide some examples of the tasks from Weston et al. 2015. A detailed description of all the tasks can be found there. Each task is noiseless, provides a set of training and test data and a human can potentially achieve 100% accuracy.

Example 1. Task 8: List/Sets

Mary grabbed the football.
Mary traveled to the office.

Mary took the apple there.

What is Mary carrying? A:football,apple

Mary left the football.

Daniel went back to the bedroom.

What is Mary carrying? A:apple

Example 2. Task 19: Path Finding

The office is east of the hallway.

The kitchen is north of the office.

The garden is west of the bedroom.

The office is west of the garden.

The bathroom is north of the garden.

How do you go from the kitchen to the garden? A:s,e

In this work, I describe an agent architecture, which I will refer to as the LKR paradigm, that simultaneously works with a formal reasoning model and a statistical inference based model to address the task of question-answering (Fig 9 10). Human beings in their lifetime learn to perform various tasks. For some tasks they may have a clear reasoning behind their actions. For example, the knowledge needed to answer the previous question "What is Mary carrying?" is clear and can be described formally. On the other hand, there are tasks such as Named Entity Recognition that we can do easily, however, we may not be able to describe it well enough for anyone else to use the description for recognition. In these cases, a statistical inference model that allows to learn by observing a distribution may be a better fit. In this research, thus, I work with a heterogeneous agent model. In our current implementation, the agent model contains three components.

Translation Module This component normally contains statistical NLP models. In this case study, it contains an Abstract Meaning Representation Parser (AMR) Banarescu et al. 2013; Flanigan et al. 2014 and an additional formatting module that the the AMR parser output to the syntax of Event calculus with some naive deterministic algorithm.

Formal Reasoning Module This module is responsible for formal reasoning. It uses the Answer Set Programming (ASP) Gelfond and Lifschitz 1988 language as the knowledge representation and reasoning language. The knowledge required for reasoning is learned with an Inductive Logic Programming algorithm XHAIL Ray 2009. The reasoning module takes sentences represented in the logical language of Event calculus which is a temporal logic for reasoning about the events and their efforts. The ontology of the Event calculus comprises of *time points*, *fluent* (i.e. properties which have certain values in time) and *event* (i.e. occurrences in time that may affect fluents and alter their value). The formalism also contains two domain-independent axioms to incorporate the commonsense *law of inertia*, according to which fluents persist over time unless they are affected by an event. The building blocks of Event calculus and its domain independent axioms are presented in Table 2.

Learning Module The Learning module takes as input the formal representation of the passage, the question and the answer and some additional information regarding the learning objective and outputs a set of rules, which are then used by the reasoning module to answer new questions.

Given a question-answer text such as the one shown in Example 1 (Task 8), the translation module first converts the natural language sentences to the syntax of Event calculus. While

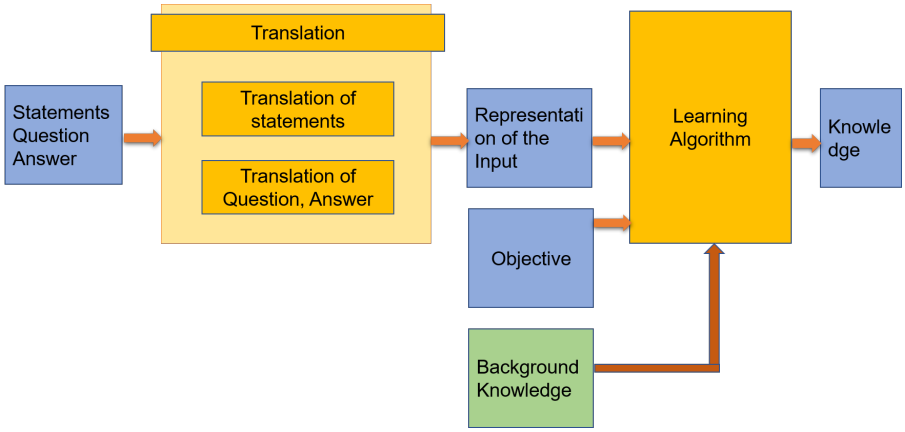

Figure 9: The modules that are involved in the training phase of LKR framework

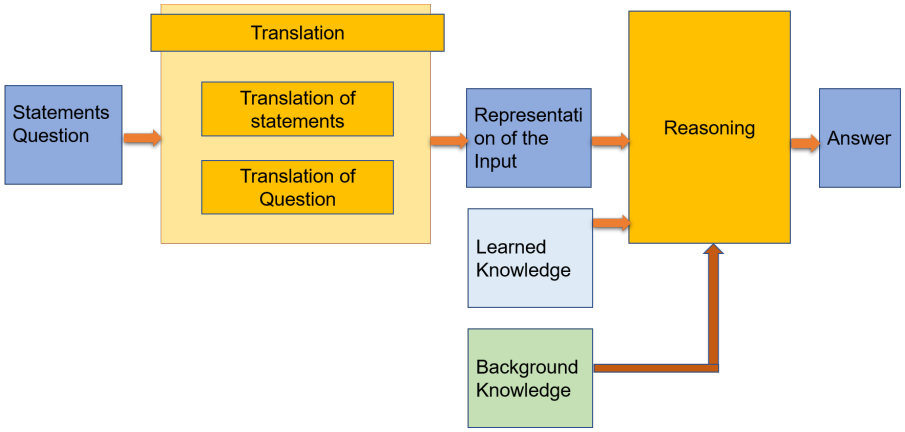

Figure 10: The modules that are involved in the test phase of LKR framework

doing so, it first obtains the Abstract Meaning Representation (AMR) of the sentence from the AMR parser in the statistical NLP layer and then applies a rule-based procedure to convert the AMR graph to the syntax of Event calculus. Figure 1 & 2 show two AMR representations for the sentence "Mary grabbed the football." and the question "What is Marry carrying?". The representation of the sentences (*narratives*) and the question-answer pairs (*annotation*) of Example 1 in Event calculus is shown in Table 3. The narratives in Table 3 describe that the event of grabbing a football by Mary has happened at time point 1, then another event named *travel* has happened at time point 2 and so on. The first two annotations state that both the fluents specifying Mary is carrying an apple and Mary is

Predicate	**Meaning**
happensAt(F,T)	Event E occurs at time T
initiatedAt(F,T)	At time T a period of time for which fluent F holds is initiated
terminatedAt(F,T)	At time T a period of time for which fluent F holds is terminated
holdsAt(F,T)	Fluent F holds at time T
Axioms	
$holdsAt(F,T+1) \leftarrow initiatedAt(F,T).$	$holdsAt(F,T+1) \leftarrow holdsAt(F,T),$ $not\ terminatedAt(F,T).$

Table 2: The basic predicates and axioms of Simple Discrete Event Calculus (SDEC)

carrying a football holds at time point 3. The *not holdsAt* annotation states that at time point 7 Mary is not carrying a football. Given such a set of narratives and annotations the reasoning module employs an Inductive Logic Programming algorithm to derive a Hypothesis $\mathcal{H}$, that can explain all the annotations.

The rest of the chapter is organized as follows: in section 4.2, I provide a brief overview of Answer Set Programming and Inductive Logic Programming; In section 3.2, I describe the way the task specific ASP reasoning rules are learned. Section 3.2.3 presents training of the coreference resolution system with reasoning. In section 4.4, I describe the related works. In section 4.5.1, I present a detailed experimental evaluation of our system. Finally, section 3.5 concludes our paper. Further details are available at *http://goo.gl/JMzHbG*.

```
(g / grab
    :ARG0 (p / person
        :name (m / name :op1 Mary ))
    :ARG1 (f / football ))
```

```
(c / carry
    :ARG0 (m / Marry )
    :ARG1 (a / amr-unknown)))
```

Figure 12: AMR representation of “What is Marry carrying?”

Narrative
happensAt(grab(mary,football),1).
happensAt(travel(mary,office),2).
happensAt(take(mary,apple),3).
happensAt(leave(mary,footbal;),5).
happensAt(go_back(daniel,bedroom),6).
Annotation
holdsAt(carry(mary,football),4).
holdsAt(carry(mary,apple),4).
holdsAt(carry(mary,apple),7).
not holdsAt(carry(mary,football),7).

Table 3: Representation of the Example 1 in Event Calculus

3.1 Background

3.1.1 Answer Set Programming

An answer set program is a collection of rules of the form,

$$L_0 \leftarrow L_1, ..., L_m, \textbf{not } L_{m+1}, ..., \textbf{not } L_n$$

where each of the L_i's is a literal in the sense of a classical logic. Intuitively, the above rule means that if $L_1, ..., L_m$ are true and if $L_{m+1}, ..., L_n$ can be safely assumed to be false then L_0 must be true Baral 2003 . The left-hand side of an ASP rule is called the *head* and the right-hand side is called the *body*. The semantics of ASP is based on the stable model (answer set) semantics of logic programming Gelfond and Lifschitz 1988.

3.1.1.1 Example

$$initiatedAt(carry(A, O), T) \leftarrow$$
$$happensAt(take(A, O), T). \tag{3.1}$$

The above rule represents the knowledge that the fluent $carry(A, O)$, denoting A is carrying O, gets initiated at time point T if the event $take(A, O)$ occurs at T. Following Prolog's convention, throughout this chapter, predicates and ground terms in logical formulae start with a lower case letter, while variable terms start with a capital letter. A rule with no *head* is often referred to as a *constraint*. A rule with empty *body* is referred to as a $fact$. An answer set program $\mathcal{P}$ containing the above rule (Rule 5.1) and the axioms of Event calculus (from Table 2) along with the fact $happensAt(take(mary, football), 1)$ logically entails ($\models$) that *mary* is *carrying* a *football* at time point 2 i.e. $holdsAt(carry(mary, football), 2)$. Since it can be safely assumed that *mary* is not *carrying* a *football* at time point 1, $\mathcal{P} \models not\ holdsAt(carry(mary, football), 1)$ or equivalently $\mathcal{P} \not\models holdsAt(carry(mary, football), 1)$.

It should be noted that it is also true that $\mathcal{P} \models holdsAt(carry(mary, football), 3)$, due to the axioms in Table 2. However, if we add the following two rules in the program $\mathcal{P}$:

$$terminatedAt(carry(A, O), T) \leftarrow$$
$$happensAt(drop(A, O), T). \tag{3.2}$$
$$happensAt(drop(marry, football), 2). \tag{3.3}$$

then the new program $\mathcal{P}$ will no longer entail $holdsAt(carry(mary, football), 3)$ due the axioms of Event calculus. This is an example of non-monotonic reasoning when adding more knowledge changes one's previous beliefs and such thing is omnipresent in human

reasoning. First Order Logic does not allow non-monotonic reasoning and this is one of the reasons why I have used the Answer Set Programming language as the formal reasoning language.

3.1.2 Inductive Logic Programming

Inductive Logic Programming (ILP) Muggleton 1991 is a subfield of Machine learning that is focused on learning logic programs. Given a set of positive examples $\mathcal{E}^+$, negative examples $\mathcal{E}^-$ and some background knowledge $\mathcal{B}$, an ILP algorithm finds an Hypothesis $\mathcal{H}$ (answer set program) such that $\mathcal{B} \cup \mathcal{H} \models \mathcal{E}^+$ and $\mathcal{B} \cup \mathcal{H} \not\models \mathcal{E}^-$.

The possible hypothesis space is often restricted with a language bias that is specified by a series of mode declarations $\mathcal{M}$ Muggleton 1995. A *modeh(s)* declaration denotes a literal *s* that can appear as the head of a rule (Table 12). A *modeb(s)* declaration denote a literal *s* that can appear in the body of a rule (Table 12). The argument *s* is called *schema* and comprises of two parts: 1) an identifier for the literal and 2) a list of *placemakers* for each argument of that literal. A *placemaker* is either +*type* (input), -*type* (output) or #*type* (constant), where *type* denotes the type of the argument. An answer set rule is in the hypothesis space defined by $\mathcal{L}$ (call it $\mathcal{L}(\mathcal{M})$) iff its head (resp. each of its body literals) is constructed from the schema *s* in a *modeh(s)* (resp. in a *modeb(s)*) in $\mathcal{L}(\mathcal{M})$) as follows:

- By replacing an output (-) placemaker by a new variable.
- By replacing an input (+) placemaker by a variable that appears in the head or in a previous body literal.
- By replacing a ground (#) placemaker by a ground term.

*modeh(*initiatedAt*(carrying(+arg*$_1$*,+arg*$_3$*),+time))*
*modeh(*terminatedAt*(carrying(+arg*$_1$*,+arg*$_3$*),+time))*
*modeb(*happensAt*(grab(+arg*$_1$*,+arg*$_2$*),+time))*
*modeb(*happensAt*(take(+arg*$_1$*,+arg*$_3$*),+time))*
*modeb(*happensAt*(go_back(+arg*$_1$*,+arg*$_2$*),+time))*
*modeb(*happensAt*(leave(+arg*$_1$*,+arg*$_3$*),+time))*

Table 4: Mode declarations for the problem of Task 8

Table 12 shows a set of mode declarations $\mathcal{M}$ for the example problem of Task 8. The Rule 5.1 of the previous section is in this $\mathcal{L}(\mathcal{M})$ and so is the fact,

$$initiated(carrying(A, O), T).$$

However the following rule is not in $\mathcal{L}(\mathcal{M})$).

$$initiated(carrying(A, O), T) \leftarrow$$
$$happensAt(take(A, O), T').$$

The set $\mathcal{E}^-$ is required to restrain $\mathcal{H}$ from being over generalized. Informally, given a ILP task, an ILP algorithm finds a hypothesis $\mathcal{H}$ that is general enough to cover all the examples in $\mathcal{E}^+$ and also specific enough so that it does not cover any example in $\mathcal{E}^-$. Without $\mathcal{E}^-$, the learned $\mathcal{H}$ will contain only facts. In this case study, negative examples are automatically generated from the positive examples by assuming the answers are complete, i.e. if a question-answer pair says that at a certain time point *mary* is carrying a *football* we assume that *mary* is not carrying anything else at that time stamp.

3.2 Learning Answer Set Programs for QA

In this section, I illustrate the formulation of an ILP task for a QA task and the way the answer set programs are learned. I explain the approach with the XHAIL Ray 2009

algorithm and specify why a better learning algorithm is needed. I continue with the example of Task 8 and conclude with path finding.

3.2.1 Task 8: Lists/Sets

Given an ILP task $ILP(\mathcal{B}, \mathcal{E} = \{\mathcal{E}^+ \cup \mathcal{E}^-\}, \mathcal{M})$, XHAIL derives the hypothesis in a three step process. For the example of task 8, $\mathcal{B}$ contains both the axioms of SDEC and the narratives from Table 1. The set $\mathcal{E}$ comprises of the annotations from Table 1 which contains three positive and one negative examples. $\mathcal{M}$ is the set of mode declarations in Table 2.

3.2.1.1 Step 1

In the first step the XHAIL algorithm finds a set of ground (variable free) atoms $\triangle = \cup_{i=1}^{n} \alpha_i$ such that $\mathcal{B} \cup \triangle \models \mathcal{E}$ where each α_i is a ground instance of the *modeh(s)* declaration atoms. For the example ILP problem of task 8 there are two *modeh* declarations. Thus the set $\triangle$ can contain ground instances of only those two atoms described in two *modeh* declarations. In the following I show one possible $\triangle$ that meets the above requirements for the ILP task of Example 1.

$$\triangle = \left\{ \begin{array}{l} initiatedAt(carry(mary, football), 1) \\ initiatedAt(carry(mary, apple), 3) \\ terminatedAt(carry(mary, football), 5) \end{array} \right\}$$

3.2.1.2 Step 2

In the second step, XHAIL computes a clause $\alpha_i \leftarrow \delta_i^1 ... \delta_i^{m_i}$ for each α_i in $\triangle$, where $\mathcal{B} \cup \triangle \models \delta_i^j, \forall 1 \leq i \leq n, 1 \leq j \leq m_i$ and each clause $\alpha_i \leftarrow \delta_i^1 ... \delta_i^{m_i}$ is a ground instance of a rule in $\mathcal{L}(\mathcal{M})$. In the running example, $\triangle$ contains three atoms that each must lead to a clause $k_i, i = 1, 2, 3$. The first atom $\alpha_1 = initiatedAt(carry(mary, football), 1)$ is initialized to the head of the clause k_1. The body of k_1 is saturated by adding all possible ground instances of the literals in *modeb(s)* declarations that satisfy the constraints mentioned above. There are six ground instances (all the narratives) of the literals in the *modeb(s)* declarations; however only one of them, i.e. $happensAt(grab(mary, football), 1)$ can be added to the body due to restrictions enforced by $\mathcal{L}(\mathcal{M})$. In the following I show the set of all the ground clauses $\mathcal{K}$ constructed in this step and their variabilized version $\mathcal{K}_v$ that is obtained by replacing all input and output terms by variables.

$$K = \left\{ \begin{array}{l} initiatedAt(carry(mary, football), 1) \\ \quad \leftarrow happensAt(grab(mary, football), 1). \\ initiatedAt(carry(mary, apple), 3) \\ \quad \leftarrow happensAt(take(mary, apple), 3). \\ terminatedAt(carry(mary, football), 6) \\ \quad \leftarrow happensAt(leave(mary, apple), 6). \end{array} \right\}$$

$$K_v = \left\{ \begin{array}{l} initiatedAt(carry(X,Y),T) \\ \quad \leftarrow happensAt(grab(X,Y),T). \\ initiatedAt(carry(X,Y),T) \\ \quad \leftarrow happensAt(take(X,Y),T). \\ terminatedAt(carry(X,Y),T) \\ \quad \leftarrow happensAt(leave(X,Y),T). \end{array} \right\}$$

3.2.1.3 Step 3

In this step XHAIL tries to find a compressive theory $\mathcal{H}$ by deleting from K_v as many literals (and clauses) as possible while ensuring that $\mathcal{B} \cup \mathcal{H} \models \mathcal{E}$. In the running example, working out this problem will lead to $\mathcal{H} = \mathcal{K}_v$.

3.2.1.3.1 Scalability of the Learning Algorithm

The discovery of a hypothesis $\mathcal{H}$ depends on the choice of $\triangle$. Since the value of $\triangle$ that satisfies the constraints described in Step 1 is not unique, I employ an iterative deepening strategy to select $\triangle$ of progressively increasing size until a solution is found. Furthermore, in Step 2 of XHAIL I restricted the algorithm to consider only those ground instances of *modeb* declarations that are not from the future time points. This method works when the size of the example is small. However, the dataset of Task 8 like other tasks contains 1000 examples, where each example comprises of a set of narrative and annotations (as I have shown before) and the choice of $\triangle$ will be numerous. This issue is addressed by learning rules from each example and then using the learned rules to learn new rules from yet unsolved examples.

Even though this strategy works for this dataset, in general it is not a sound strategy, as the learned rules might not be consistent with the next example.

3.2.2 Task 19 : Path Finding

In this task (Example 2), each example first describes the relative positions of several places and then asks a question about moving from one place to another. The answer to the question is then a sequence of directions. For the question "How do you go from the kitchen to the garden?" in Example 2, the answer "s,e" tells that to reach garden from kitchen, you should first head south and then head east.

Given such an example, an agent learns how moving towards a direction changes its current location with respect to the particular orientation of the places. Let us say, $mt(X, Y)$ denotes the event of X moving towards the direction Y. Similar to the earlier problem, the natural language text is first translated to the syntax of ASP (Table 5). However, in this task the background knowledge $\mathcal{B}$ also contains the rules learned from the task 4. In the following I show an example of such rules:

$$holdsAt(relative\_position(X, Y, east), T) \leftarrow$$
$$holdsAt(relative\_position(Y, X, west), T).$$

The above rule says that if Y is to the west of X at time point T then X is to the east of Y at T. Similar rules were learned for each direction pair from the Task 4 which were used in the process of hypothesis generation for the task of path finding. Table 5 shows the corresponding ILP task for the example of path finding and the hypothesis generated by the XHAIL algorithm. This example illustrates how the task of path finding can be easily learned when a formal representation is used. While the state-of-the-art neural network

based systems have achieved 36% accuracy on this task with an average of 93% on all tasks, our system is able to achieve 100% with the two compact rules shown in Table 5.

Input
Narrative
holdsAt(relative_position(office,hallway,east),1).
holdsAt(relative_position(kitchen,office,north),2).
holdsAt(relative_position(garden,bedroom,west),3).
holdsAt(relative_position(office,west,garden),4).
holdsAt(relative_position(bathroom,garden,north),5).
holdsAt(location(you,kitchen),6). happensAt(mt(you,south),6).
happensAt(mt(you,east),7).
Annotation
not holdsAt(*location*(*you*, *garden*), 6).
holdsAt(*location*(*you*, *garden*), 8).
not holdsAt(*location*(*you*, *kitchen*), 8).
Mode declarations
*modeh(*initiatedAt*(location(+arg$_1$,+arg$_2$),+time))*
*modeh(*terminatedAt*(carrying(+arg$_1$,+arg$_2$),+time))*
*modeb(*happensAt*(mt(+arg$_1$,+direction),+time))*
*modeb(*holdsAt*(location(+arg$_1$,+arg$_2$),+time))*
*modeb(*holdsAt*(relative_position(+arg$_2$,+arg$_2$, +direction),+time))*
Background Knowledge
Axioms of SDEC (Table 1)
Output
$initiatedAt(location(X, Y), T) \leftarrow happensAt(mt(X, D), T)$,

$holdsAt(relative\_position(Y,Z,D),T), holdsAt(location(X,Z),T).$

$terminatedAt(location(X,Y),T) \leftarrow happensAt(mt(X,D),T).$

Table 5: Hypothesis Generation For Path Finding

3.2.3 Learning Coreference Resolution with Reasoning

The dataset contains contains two tasks related to coreference resolution : 1) task of basic coreference resolution and 2) task of compound coreference resolution. Examples of the tasks are shown below :

Task 11: Basic Coreference

Mary went back to the bathroom.
After that she went to the bedroom.
Daniel moved to the office.
Where is Mary? bedroom

Task 13: Compound Coreference

Daniel and Sandra journeyed to the office.
Then they went to the garden.
Sandra and John travelled to the kitchen.
The office is west of the garden.
After that they moved to the hallway.
Where is Daniel? A:garden

I formulate both the coreference resolution tasks as ILP problems and surprisingly it learns answer set rules that can fully explain the test data. For the task of basic coreference, it learns a total of five rules one for each of the five different events *go, travel, go back, move, journey* that appeared in the training data. The rule corresponding to the event *go* (Table 6) states that if a narrative at time point $T + 1$ contains a pronoun, then the pronoun is referring to the arg_1 (agent) of the event *go* that happened at time point T. Similar rules were learned for the other four events. Here, $corefId(X, T)$ denotes that the pronoun with id X has appeared in a narrative at time point at $T + 1$.

$$initiatedAt(coref(X, Y), T) \leftarrow corefId(X, T), \\ happensAt(go(Y, Z), T).$$

Table 6: One rule for coreference resolution

One drawback of the learned rules is, they are event dependent, i.e. if a coreference resolution text contains a pronoun which is referring to an argument of an previously unseen event, these rules will not be able to resolve the coreference. In spite of that, these rules reflect one of the basic intuitions behind coreference resolution and all of them are learned from data.

3.3 Related Works

In this section, I briefly describe the two other attempts on this challenge. The attempt using Memory Network (MemNN) Weston, Chopra, and Bordes 2014 formulates the QA task as a search procedure over the set of narratives. This model takes as input the Question-Answering samples and the set of facts required to answer each question. It then learns to find 1) the supporting facts for a given question and 2) the word or set of words from

the supporting facts which are given as answer. Even though this model performs well on average, the performance on the tasks of positional reasoning (65%) and path finding (36%) are far below from the average (93%).

The attempt using Dynamic Memory Network (DMN) Kumar et al. 2015 also models the the QA task as a search procedure over the set of narratives. The major difference being the way supporting facts are retrieved. In the case of the Memory Networks, given a question, the search algorithm scans the narratives in the reverse order of time and finds the most relevant hypothesis. It then tries to find the next most relevant narrative and the process continues until a special marker narrative is chosen to be the most relevant one in which case the procedure terminates. In the case of Dynamic Memory Networks the algorithm first identifies a set of useful narratives conditioning on the question and updates the agent's current state. The process then iterates and in each iterations it finds more useful facts that were thought to be irrelevant in the previous iterations. After several passes the module finally summarizes its knowledge and provides the answer. Both the models rely only on the given narratives to answer a question. However, for many QA tasks (such as task of Path finding) it requires additional knowledge that is not present in the text (for path finding, knowledge from Task 4), to successfully answer a question. Both MemNN and DMN models suffer in this case whereas our method can swiftly combine knowledge learned from various tasks to handle more complex QA tasks.

3.4 Experiments

Table 14 shows the performance of our method on the set of 20 tasks. For each task, there are 1000 questions for training and 1000 for testing. Our method was able to answer all the question correctly except the ones testing basic induction. In the following I provide

TASK	MemNN	DMN	Our Method
1: Single Supporting Fact	100	100	100
2: Two Supporting Facts	100	98.2	100
3: Three Supporting facts	100	95.2	100
4: Two Argument Relations	100	100	100
5: Three Argument Relations	98	99.3	100
6: Yes/No Questions	100	100	100
7: Counting	85	96.9	100
8: Lists/Sets	91	96.5	100
9: Simple Negation	100	100	100
10: Indefinite Knowledge	98	97.5	100
11: Basic Coreference	100	99.9	100
12: Conjunction	100	100	100
13: Compound Coreference	100	99.8	100
14: Time Reasoning	99	100	100
15: Basic Deduction	100	100	100
16: Basic Induction	100	99.4	93.6
17: Positional Reasoning*	65	59.6	100
18: Size Reasoning	95	95.3	100
19: Path Finding	36	34.5	100
20: Agent's Motivations*	100	100	100
Mean Accuracy(%)	93.3	93.6	99.68

Table 7: Performance on the set of 20 tasks

a detail error analysis for the task of Induction. For each task the *modeh* and the *modeb* declarations were manually defined and can be found at the project website. The test set of Task 5 (Three argument relations) contains 2 questions that have incorrect answers. The result is reported on the corrected version of that test set. The details on the error can be found on the project website. Training of the tasks that are marked with (*) used the annotation of supporting facts present in the training dataset.

3.4.1 Error Analysis for Basic Induction

This task tests basic induction via potential inheritance of properties. The dataset contains a series of examples like the one described below:

Lily is a frog.
Julius is a swan.
Julius is green.
Lily is grey.
Greg is a swan.
What color is Greg? green

The learning algorithm could not find a hypothesis that can characterize the entire training data with the given set of mode declarations. So, I took the hypothesis that partially explained the data. This was obtained by ignoring the examples in the training data which resulted in a failure. The resulted hypothesis then contained the following single rule:

$$\begin{aligned} holdsAt(color(X,C),T) \leftarrow\ & holdsAt(domain(Z,D),T), \\ & holdsAt(color(Z,C),T), \\ & holdsAt(domain(X,D),T). \end{aligned}$$

The above rule says that X has color C at time T if there exists a Z which is of type D and has color C at time point T, where X is also of type D. This rule was able to achieve 93.6% accuracy on the test set. However it failed for the examples of following kind where there are two different entity of type D having two different colors:

For the error case 1, the learned rule will produce two answers stating that Bernhard has the color grey and yellow. Since, the more number of frogs are grey it may seem like the correct rule should produce the color that has appeared maximum number of times for that

Error Case 1	Error Case 2
Lily is a frog.	Lily is a rhino.
Brian is a frog.	Lily is yellow.
Greg is frog.	Bernhard is a frog.
Lily is yellow.	Bernhard is white.
Julius is a frog.	Brian is a rhino.
Brian is grey.	Greg is a rhino.
Julius is grey.	Greg is yellow.
Greg is grey.	Julius is a rhino.
Bernhard is a frog.	Julius is green.
What color is Bernhard? A:grey	What color is Brian? A:green

Table 8: Failure cases for Induction

type (here, frog). However, error case 2 describes a complete opposite hypothesis. There are two yellow rhino and one grey rhino and the color of Brian which is a rhino is grey. The actual rule as it appears is the one that determines the color on the basis of the latest evidence. Since, Memory Networks scans the facts in the deceasing order of time, it always concludes from the recent most narratives and thus has achieved a 100% accuracy.

3.5 Conclusion

This chapter presents the LKR approach for the task of Question-Answering that benefits from the field of knowledge representation and reasoning, inductive logic programming and statistical natural language processing. Ihave shown that to employ knowledge representation and reasoning, one does not have to write the rules but the rules can be learned and the resulting system not only is explainable but also performs better than the deep learning models due to the addition of a formal reasoning layer significantly increases the reasoning capability of an agent.

Chapter 4

SCALING LEARNING OF INFERENCE ENABLING KNOWLEDGE: AN EFFICIENT INDUCTIVE LOGIC PROGRAMMING ALGORITHM

4.1 Introduction

Answer Set Programming has emerged as a powerful tool for knowledge representation and reasoning. To use this tool for an application, however, one needs application specific knowledge. For E.g., if a system uses answer set programming to answer the question from column 1 in Table 9 the system needs to know that "X is to the right of Y IF Y is to the left of Z and Z is above X". Inductive Logic Programming algorithms aim to learn these kinds of knowledge from a dataset. However, as we have seen in previous chapter, existing ILP algorithms often have limited scalability. This often leads to manual construction of a knowledge base which can be very time consuming and may not be practical sometimes. For E.g., for applications where an effective representation of the rules is unknown, such as for the case of handwritten digit recognition (Fig. 13), one may need to try several representations before settling down for a winner. However, this may be unrealistic given that MNIST dataset (Fig. 13) contains $50,000$ examples and writing down the rules that explain all these examples for a particular choice of representation will take significant amount of time.

In this chapter, I consider this scalability issue. I observe that one major obstruction in scalability arises from the discrepancy between the definition of Inductive Logic Programming and the structure of a machine learning dataset. The learning problem in Inductive Logic Programming (ILP) is defined as follows Muggleton 1991:

Definition 1 (Inductive Logic Programming) *Given a set of positive examples E^+, negative examples E^- and some background knowledge B, an ILP algorithm finds an Hypothesis H such that,*

$$B \cup H \models E^+, B \cup H \not\models E^-$$

The hypothesis space is restricted with a language bias that is specified by a series of mode declarations M.

A machine learning dataset on the other hand contains a series of $\langle x, y \rangle$ pairs, x being the input and y being the desired output (Table 9). To work with an ILP algorithm, one needs to first convert the $\langle x, y \rangle$ pairs in the format of $\langle B, E^+, E^- \rangle$. The conversion process is carried out by the user and so there might be some variations. However, normally the sets E^+ and E^- are created using y's and the x's go inside B. Extra care is taken so that different $\langle x, y \rangle$ pairs do not interfere with each other. Table 2(a) shows one example of this process. Since the number of $\langle x, y \rangle$ pairs are usually large, the problem instance becomes too big for the ILP solvers to handle . For example, consider someone wants to employ an ILP algorithm to learn from a question answering task from bAbI dataset Weston et al. 2015, which contains $1,000$ comprehension examples similar to the ones in Table 9. The resulting background knowledge B will contain about $10,000$ facts and E^+ will contain $1,000$ positive annotations pertaining to answers and E^- will contain a total of $1,000$ negative examples describing what is not an answer for each question. An ILP solver such as XHAIL Ray 2009 will throw memory errors when given an input of this size. The question that I ask here is "can we find a solution to the ILP problem without considering all the $\langle x, y \rangle$ pairs together ?" I show that the answer is yes. In fact it is possible to find a solution considering only one $\langle x, y \rangle$ pair at a time. To achieve this I model the learning task as follows:

x	The square is above the rectangle. The triangle is to the left of the square. Is the rectangle to the right of the triangle?	The square is below the rectangle. The triangle is to the right of the square. Is the rectangle to the right of the triangle?	The square is below the rectangle. The triangle is to the right of the square. Is the triangle below the rectangle?
y	Yes	No	Yes

Table 9: A set of examples taken from the Task 17 of bAbI question answering dataset.

Figure 13: A set of images from the MNIST dataset.

Definition 2 (Inductive Logic Programming for Distinct Examples) *An ILP task for Distinct Examples (denoted as ILP^{DE}) is a tuple $\langle B, M, D\rangle$, where B is an Answer Set Program, called the background knowledge, M defines the set of rules allowed in hypotheses (the hypothesis space) and D is the dataset containing a series of context dependent examples $\langle E_1, E_2, ..., E_n\rangle$. Here each E_i is a tuple $\langle O_i, E_i^+, E_i^-\rangle$ where, O_i is a logic program, called observation , E^+ is a set of positive ground literals and E^- is a set of negative ground literals. A hypothesis H is an inductive solution of T (written as $H \in ILP^{DE}(B, M, D)$) iff,*

$$H \cup B \cup O_i \vdash E_i^+, \ \forall i = 1...n$$

$$H \cup B \cup O_i \nvdash E_i^-, \ \forall i = 1...n$$

In this formulation, each example $\langle O_i, E_i^+, E_i^-\rangle$ directly corresponds to an $\langle x, y\rangle$ pair and it takes into consideration that there are several distinct examples in a dataset, so there is no need to explicitly isolate them from each other. Table 2(b) shows the encoding of the running example in the format of ILP^{DE}. It turns out that the ILP^{DE} task described here is a

simplification of the *Context-dependent Learning from Ordered Answer Sets task* proposed in Law, Russo, and Broda 2016. However, to solve the *Context-dependent Learning from Ordered Answer Sets task* the authors in Law, Russo, and Broda 2016 convert it to a standard ILP problem which creates the same scalability issue.

	$ans(X, no) \leftarrow not\ ans(X, yes), id(X).$
B	$sample(1, holdsAt(rp(sq, rec, above), 1)).$ $sample(1, holdsAt(rp(tri, sq, left), 1)).$ $ans(1, yes) \leftarrow$ $\quad sample(1, holdsAt(rp(rec, tri, right), 1)).$
	$sample(2, holdsAt(rp(sq, rec, below), 1)).$ $sample(2, holdsAt(rp(tri, sq, right), 1)).$ $ans2(yes) \leftarrow$ $\quad sample(2, holdsAt(rp(rec, tri, right), 1)).$
	$sample(3, holdsAt(rp(tri, sq, left), 1)).$ $sample(3, holdsAt(rp(tri, sq, left), 1)).$ $ans(3, yes) \leftarrow$ $\quad sample(3, holdsAt(rp(tri, rec, below), 1)).$
E^+	$\{ans(1, yes), ans(2, no), ans(3, yes).\}$
E^-	$\{ans(1, no), ans(2, yes), ans(3, no).\}$

(a) ILP encoding of the problem in Table 9

E_1	O_1	$holdsAt(rp(sq, rec, above), 1).$ $holdsAt(rp(tri, sq, left), 1).$ $ans(yes) \leftarrow holdsAt(rp(rec, tri, right), 1).$
	E_1^+	$\{ans(yes)\}$
	E_1^-	$\{ans(no)\}$
E_2	O_2	$holdsAt(rp(sq, rec, below), 1).$ $holdsAt(rp(tri, sq, right), 1).$ $ans(yes) \leftarrow holdsAt(rp(rec, tri, right), 1).$
	E_2^+	$\{ans(no)\}$
	E_2^-	$\{ans(yes)\}$
E_3	O_3	$holdsAt(rp(tri, sq, left), 1).$ $holdsAt(rp(tri, sq, left), 1).$ $ans(yes) \leftarrow holdsAt(rp(tri, rec, below), 1).$
	E_3^+	$\{ans(yes)\}$
	E_3^-	$\{ans(no)\}$

(b) ILP^{DE} encoding of the problem in Table 9

Table 10: The *sample* predicate is used to separate different examples. The constants *tri*, *rec*, *sq* respectively denote triangle, rectangle and square. $holdsAt(rp(sq, rec, above), 1)$ says that the square is above the rectangle at time point 1.

It should be noted that any standard ILP problem $\langle B, M, E^+, E^- \rangle$ can be thought of as an ILP^{DE} problem with only one example, $\langle \{\}, M, \langle (B, E^+, E^-) \rangle \rangle$. Similarly any ILP^{DE} task can be converted to an ILP task. However, utilizing the 'distinctness' property of the examples we can do better. The algorithm that I propose here roughly works as follows: Given an instance of the ILP^{DE} task, it first finds a solution H_1 of E_1. Then it expands H_1 minimally to solve only E_2 and obtains H_2 . In the next iteration it again expands H_2 minimally to solve E_1 and it continues expanding until it finds a hypothesis that solves both E_1 and E_2. Next it starts with a solution of $\langle E_1, E_2 \rangle$ and tries to expand it iteratively until it solves all of E_1, E_2 and E_3. The process continues until a hypothesis is found that explains all the examples.

Section 4.3 describes the algorithm. I show that the algorithm is sound and complete when $H \cup B \cup O_i$ is *stratified* for all $i = 1, ..., n$.

The proposed algorithm allows more control over the mode declarations (Section 4.2) which can lead to noticeable speed up in the search process. I evaluate our algorithm on two popular datasets: 1) a question answering dataset published by Facebook AI Research Weston et al. 2015 and 2) a handwritten digit recognition database LeCun 1998. To the best of my knowledge, no sound and complete ILP algorithm previously could learn from these two datasets.

4.2 Background

In this section, I describe the type of rules that the algorithm can deal with, the syntax of the mode declarations and the XHAIL algorithm which plays a crucial role in the proposed algorithm.

4.2.1 Answer Set Programming

An answer set program is a collection of rules of the form,

$$L_0 \leftarrow L_1, ..., L_m, \textbf{not } L_{m+1}, ..., \textbf{not } L_n$$

where each of the L_i's is a literal in the sense of a classical logic. Intuitively, the above rule means that if $L_1, ..., L_m$ are true and if $L_{m+1}, ..., L_n$ can be safely assumed to be false then L_0 must be true. The left-hand side of an ASP rule is called the *head* and the right-hand side is called the *body*. Predicates and ground terms in a rule start with a lower case letter, while variable terms start with a capital letter. I will follow this convention throughout the paper. A rule with no *head* is called a *constraint*. A rule with empty *body* is referred to as a $fact$.

The semantics of ASP is based on the stable model semantics of logic programming Gelfond and Lifschitz 1988. In this work, both the background knowledge B and the solution H are a collection of such ASP rules.

4.2.2 Mode Declarations

Given a set of positive examples E^+, negative examples E^- and some background knowledge B, an ILP algorithm computes a set of rules H so that $B \cup H \models E$. The rules in H are often restricted with a language bias that is specified by a series of mode declarations M Muggleton 1995. One can think of this as a way of injecting expert knowledge for the learning task.

There are two types of mode declarations, namely *modeh* declarations and *modeb* declarations. A *modeh(s)* declaration (Table 12) specifies a literal s that can appear as the head of a rule in H. A *modeb(s)* declaration (Table 12) specifies a literal s that can appear in the body of a rule. The argument s is called *schema* and comprises of two parts: 1) an *identifier* for the literal and 2) a list of *placemakers* for each argument of that literal. A *placemaker* is either +*type* (input), -*type* (output) or *$ type* (constant), where *type* denotes the type of the argument. An answer set rule is in the hypothesis space defined by M (call it $L(M)$) if and only if its head (resp. each of its body literals) is constructed from the schema s in a *modeh(s)* (resp. in a *modeb(s)*) in $L(M)$) as follows:

- by replacing an output (-) placemaker by a new variable.
- by replacing an input (+) placemaker by a variable that appears in the head or in a previous body literal and
- by replacing a ground ($) placemaker by a ground term.

Table 12 shows a set of mode declarations M_{sample} that one can use to solve the example

problem in Table 9.There is only one *modeh(s)* declaration in M_{sample}, where the schema is *holdsAt(relativeposition(+op1,+op1, $ direction), +time)*. Assuming that there are only four constants of type *directions*, the set of possible head literals are:

$$\left\{\begin{array}{l} holdsAt(relativeposition(X,Y,left),T), \\ holdsAt(relativeposition(X,Y,right),T), \\ holdsAt(relativeposition(X,Y,above),T), \\ holdsAt(relativeposition(X,Y,below),T) \end{array}\right\}$$

Where X and Y are variables of type $op1$ and T has type *time*. There are three *modeb* declarations and they restrict additions of literals to the body as directed by their individual schema. Note that the following rule,

$$holdsAt(relativeposition(X,Y,left),T) \leftarrow holdsAt(relativeposition(Z,X,above),T), \\ holdsAt(relativeposition(Y,Z,right),T).$$

is in $L(M_{sample})$, as the head is allowed by the *modeh* (Table 12) and the third *modeb* (Table 12) allows the addition of $holdsAt(relativeposition(Z,X,above),T)$ with Z being an output (new) variable and the first *modeb* allows the addition of $holdsAt(relativeposition(Y,Z,right),T)$, as all the associated variables Y, Z and T have appeared before.

#modeh holdsAt(relativeposition(+op1,+op1,$ direction),+time).
#modeb holdsAt(relativeposition(+op1,+op1,$ direction),+time).
#modeb holdsAt(relativeposition(+op1,-op1,$ direction),+time).
#modeb holdsAt(relativeposition(-op1,+op1,$ direction),+time).

Table 12: Mode declarations for the problem of Table 9

Additionally, *weights* can be assigned to *modeh* and *modeb* (written as *#modeh(s)=W*) and they express the cost that is involved when a mode declaration is used. The default

weight for mode declarations is 1. Existing implementations of the ILP algorithms, take only one set of mode declarations and thus all the *modeh* declarations share the same set of *modebs*. Our algorithm allows the user to provide *modeh* specific *modeb* declarations. This additional feature allows the user to provide more supervision in the search procedure and makes the search faster.

4.2.3 XHAIL

The XHAIL Ray 2009 algorithm plays a crucial role in the algorithm that I present here. In this section, I describe various concepts and notations associated with the XHAIL algorithm. Given an ILP task $ILP(B, M, E = \{E^+ \cup E^-\})$, XHAIL Ray 2009 derives the hypothesis in three steps, namely the *abductive* step, the *deductive* step and the *inductive* step. I will explain these steps with respect to the example E_1 from Table 2(b). The set B contains the representation of x_1, denoted by O_1 and the set E the contains annotations derived from y_1. M is the set of mode declarations described in Table 12.

4.2.3.1 Abductive Step

In the first step XHAIL finds a set of ground (variable free) atoms $\triangle = \{\alpha_1, ..., \alpha_n\}$ such that $B \cup \triangle \models E$, where each α_i is a ground instance of the *modeh(s)* declaration atoms. For the running example there is only one *modeh* declaration. Thus the set $\triangle$ can contain ground instances of only $holdsAt(relativeposition(X, Y, Z), T)$. In the following I show one possible $\triangle$ that meets the above requirement.

$$\triangle = \Big\{ holdsAt(relativeposition(rectangle, triangle, right), 1) \quad \Big\}$$

4.2.3.2 Deductive Step

In the second step, XHAIL computes a clause $\alpha_i \leftarrow \delta_i^1 ... \delta_i^{m_i}$ for each α_i in $\triangle$, where $B \cup \triangle \models \delta_i^j, \forall 1 \leq i \leq n, 1 \leq j \leq m_i$ and each clause $\alpha_i \leftarrow \delta_i^1 ... \delta_i^{m_i}$ is a ground instance of a rule in $L(M)$. In the running example, $\triangle$ contains only one atom, $\alpha_1 = holdsAt(relativeposition(rectangle, triangle,$
$right), 1)$ which is initialized to the head of the clause k_1. The body of k_1 is saturated by adding all possible ground instances of the literals in *modeb(s)* declarations that satisfy the constraints mentioned above. There are two ground instances, $holdsAt(relativeposition(square, rectangle, above), 1)$ and $holdsAt(relativeposition(triangle, square, left), 1)$, of the literals in the *modeb(s)* declarations and both of them can be added to the body as specified by M. In the following I show the set of ground clauses K (called *kernel*) constructed in this step and their variabilized version K_v (called *generalization*) that is obtained by replacing all input and output terms by variables.

$$K = \left\{ \begin{array}{l} holdsAt(relativeposition(rectangle, triangle, right), 1) \\ \quad \leftarrow holdsAt(relativeposition(square, rectangle, above), 1), \\ \quad holdsAt(relativeposition(triangle, square, left), 1). \end{array} \right\}$$

$$K_v = \left\{ \begin{array}{l} holdsAt(relativeposition(X, Y, right), T) \\ \quad \leftarrow holdsAt(relativeposition(Z, X, above), T), \\ \quad holdsAt(relativeposition(Y, Z, left), T). \end{array} \right\}$$

4.2.3.3 Inductive Step

In this step XHAIL tries to find a compressive theory H by selecting from K_v as few literals as possible while ensuring that $B \cup H \models E$. For this example, working out this problem will lead to a unique solution,

$$H = \left\{ holdsAt(relativeposition(X, Y, right), T). \quad \right\}$$

which contains a single rule with empty body. In general, the compression process may lead to multiple options for H.

Let $\langle H_I, H_G, \triangle \rangle$ denote a solution returned by $XHAIL(B, M, E)$, where H_G is the generalization computed from $\triangle$ and H_I is a compressed version of H_G that solves E. It should be noted that there might be many choices for $\triangle$ and correspondingly there might be many possible solutions $\langle H_I, H_G, \triangle \rangle$. In the following table, I define few notations which will be useful later.

Notations	
XHAIL(B,M,E)	The set of all the solutions $\langle H_I, H_G, \triangle \rangle$ to the problem $P = ILP(B, M, E)$, where H_I is minimal i.e. no compressed version of H_I can solve P.
$\triangle(B, M, E)$	$\{\triangle \mid \langle H_I, H_G, \triangle \rangle \in XHAIL(B, M, E)$ for some $H_I, H_G\}$.
$H_G(B, M, E)$	$\{H_G \mid \langle H_I, H_G, \triangle \rangle \in XHAIL(B, M, E)$ for some $\triangle, H_I\}$.
$H_G(\triangle)$	The generalization computed from $\triangle$.

4.3 Algorithm

XHAIL can compute the solutions of $ILP(B_{E_1}, M, \{E^+, E^-\}_{E_1})$. However how to compute the solutions of $ILP^{DE}(B, M, \langle E_1, E_2 \rangle)$ without solving the standard Inductive Logic

Programming task constructed from E_1 and E_2 (denoted by $ILP(B_{E_1,E_2}, M, \{E^+, E^-\}_{E_1,E_2})$) ? This section addresses this question. Before that I define the following terms which will be needed for the discussion.

Definition 3 ***$H_1 \leq H_2$*** *Two answer set programs H_1 and H_2 are related by "$\leq$" (denoted as $H_1 \leq H_2$) if and only if H_1 can be transformed into H_2 by either adding new rules to H_1 or by adding new literals in the body of the existing rules.*

Definition 4 ***Minimality*** *A solution H of $ILP(B, M, E)$ is **minimal** iff $\nexists H' < H$ in $L(M)$ that solves $ILP(B, M, E)$.*

Definition 5 ***Distinctness*** *A series of examples $E_i\langle O_i, E_i^+, E_i^-\rangle, i = 1...n$ are said to be distinct iff, $\Delta(B \cup O_1 \cup ... \cup O_n, M, \cup_{i=1}^n E_i^+, \cup_{i=1}^n E_i^-) = \{\cup_{i=1}^n \triangle_i | (\triangle_1, ..., \triangle_n) \in \Delta(B \cup O_1, M, E_1^+, E_1^-) \times ... \times \Delta(B \cup O_n, M, E_n^+, E_n^-)\}$. A series of examples $E_i\langle O_i, E_i^+, E_i^-\rangle, i = 1...n$ are said to be mutually distinct iff all subsets of the examples are distinct.*

Now consider the two examples E_1 and E_2 . Since E_1 and E_2 are *distinct* examples constructed from two different $\langle x, y\rangle$ pairs, by definition, $\Delta(B \cup O_1 \cup O_2, M, \cup_{i=1}^2 E_i^+, \cup_{i=1}^2 E_i^-) = \{\triangle_1 \cup \triangle_2 | (\triangle_1, \triangle_2) \in \Delta(B \cup O_1, M, E_1^+, E_1^-) \times \Delta(B \cup O_2, M, E_2^+, E_2^-)\}$. Thus, for any solution $\langle H_I, H_G, \triangle\rangle$ of $ILP(B \cup O_1 \cup O_2, M, \cup_{i=1}^2 E_i^+, \cup_{i=1}^2 E_i^-)$, $\exists \triangle_1 \in \Delta(B \cup O_1, M, E_1^+ \cup E_1^-)$ and $\exists \triangle_2 \in \Delta(B \cup O_2, M, E_2^+ \cup E_2^-)$ such that,

$$H_G(\triangle) = H_G(\triangle_1) \cup H_G(\triangle_2) \geq H_I$$

This property allows us to search for H_I's without solving $ILP(B \cup O_1 \cup O_2, M, \cup_{i=1}^2 E_i^+, \cup_{i=1}^2 E_i^-)$ directly. The search procedure can be briefly described as follows: For any choice of $(\triangle_1, \triangle_2)$ pair, first find all the minimal $H \leq H_G(\triangle_1) \cup H_G(\triangle_2)$ that solves E_1 and then expand those minimally, with respect to E_2 and E_1 alternatively, until all the minimal

H_I's that solves both E_1 and E_2 are found. To find all the H_I one simply needs to iterate over all possible $(\triangle_1, \triangle_2)$ pairs which can be computed from $ILP(B \cup O_1, M, E_1^+, E_1^-)$ and $ILP(B \cup O_2, M, E_2^+, E_2^-)$ individually.

It should be noted that it is possible to have $H_G(\triangle') = H_G(\triangle'')$, even though $\triangle' \neq \triangle''$. Thus, the above search procedure can be optimized by iterating over pairs of generalizations instead of iterating over the abducibles. Another drawback of the above search procedure is that the search results of $(H_G^1(\triangle_1), H_G^2(\triangle_2))$ do not give any information for the search initiated on $(H_G^1(\triangle_1'), H_G^2(\triangle_2'))$. In every iteration it starts from scratch. However, if we remember the solutions of $ILP^{DE}(B, M, E_1)$, we can use those as lower bounds for finding the solutions of $ILP^{DE}(B, M, \langle E_1, E_2 \rangle)$. This is because, if H_I is a minimal solution of $ILP^{DE}(B, M, \langle E_1, E_2 \rangle)$, then H_I also solves $ILP^{DE}(B, M, E_1)$ and there exists a $\langle H_I^1, H_G^1, \triangle_1 \rangle \in ILP^{DE}(B, M, E_1)$ such that $H_I^1 \leq H_I$. Thus, for the iteration $(H_G^1(\triangle_1), H_G^2(\triangle_2))$, one can search if some $H_I^1 \leq H_G^1(\triangle_1)$ can be expanded by either expanding some rules in H_I^1 or by adding new rules from the remainder of $H_G^1(\triangle_1) \cup H_G^2(\triangle_2)$ or both to solve E_2 along with E_1. Theorem 1 formalizes this idea.

Theorem 1 *For any solution $\langle H_I, H_G, \triangle \rangle$ of $ILP^{DE}(B, M, \langle E_1, ..., E_n \rangle)$ there exists a solution $\langle H_I', H_G', \triangle' \rangle$ of $ILP^{DE}(B, M, \langle E_1, ..., E_{n-1} \rangle)$ and a generalization H_G'' in $ILP^{DE}(B, M, E_n)$ such that, $H_I' \leq H_I \leq H_G' \cup H_G''$, when $H \cup B \cup O_i$ is stratified for any choice of $i \in \{1, ..., n\}$ and $H \in \{H_G, H_G', H_G''\}$. Here, O_i is the observation from E_i.* ■

With this in mind, the algorithm for finding the solutions of $ILP^{DE}(B, M, \{E_1, E_2, ..., E_n\})$ is described in Algorithm 1. The proof of the theorem is in Appendix A.

Algorithm 1: I^2XHAIL

1 1.0 **Data:** An instance of $ILP^{DE}(B, M, \{E_1, \ldots, E_n\})$
Result: A solution to the problem
/* initialize a stack with the solutions of $ILP(B, M, E_1)$ */
2 $stack = XHAIL(ILP(B, M, E_1))$;
3 **while** *stack is not empty* **do**
/* pop the hypothesis from the top */
4 $\langle H_I, H_G \rangle = stack.pop()$;
/* get an example E_i such that $B \cup H_I \cup O_i \nvdash E_i^+$ or $B \cup H_I \cup O_i \vdash E_i^-$ */
5 $E_i = nextUncoveredExample(H_I)$;
/* No such example exists */
6 **if** E_i *is null* **then**
/* found a solution */
7 return H_I.
8 **else**
/* Find expansions of H_I that also solves E_i */
9 $refinementsStack = <>$;
/* support set denotes the set of examples from which $< H_i, H_G >$ is created */
10 $supports = supportSet(H_I) \cup \{E_i\}$;
/* compute a set of lower bound-upper bound pairs for the search space. */
11 $H_G(E_i) = findGeneralizatons(B, M, E_i)$;
12 **foreach** H *in* $H_G(E_i)$ **do**
13 *push* $\langle H_I, H_G \cup H \rangle$ to $refinementsStack$
14 **while** *refinementsStack is not empty* **do**
/* get a candidate lower bound-upper bound pair */
15 $\langle H'_I, H'_G \rangle = refinementsStack.pop()$;
/* get an example from *supports* that is not covered by H'_I */
16 $E_j = nextUncoveredExampleFromS(H'_I, supports)$;
17 **if** E_j *is null* **then**
/* if no such example exists then we found a solution to the subproblem. Push it to the *stack*. */
18 *push* $\langle H'_I, H'_G \rangle$ to *stack*;
19 **else**
/* Expand H'_I minimaly along H'_G so that it covers E_j */
20 $expansions = expandMinimal(\langle H'_I, H'_G \rangle, B, E_j)$;
/* Push all expansions in the $refinementsStack$ for further updates. */
21 **foreach** $\langle H''_I, H''_G \rangle$ *in expansions* **do**
22 $refinementsStack.push(\langle H''_I, H''_G \rangle)$

4.3.1 Example

In this subsection I describe how our algorithm computes a solution to the running example $ILP^{DE}(B, M, \langle E_1, E_2, E_3\rangle)$ from Table 9. Here B contains all the constants of type $op1$, *direction* and *time* and M is the one described in Table 12 .

Initialization: First the *stack* is filled with the output from $XHAIL(B, M, E_1)$. In section 1, we have seen that the output contains only one tuple. The following block shows the content of the stack after initialization.The underlined part denotes H_I, where H_G is the entire program.

$$\underline{holdsAt(relativeposition(X, Y, right), T)}$$
$$\leftarrow holdsAt(relativeposition(Z, X, above), T), holdsAt(relativeposition(Y, Z, left), T).$$

Iteration 1: In iteration 1, the hypothesis on the top (denoted as $Top\langle H_I^{Top}, H_G^{Top}\rangle$) of the stack is popped. One can see that the hypothesis H_I^{Top} does not cover E_2. So, the algorithm tries to find an expansion of it which solves E_2 and E_1 both. For that it first finds $H_G(B, M, E_2)$ and creates a new *refinement stack* with lower bound (H_I^{Top}) - upper bound ($H_G^{Top} \cup H_G^{Top}$) pairs as shown below:

$$\underline{holdsAt(relativeposition(X, Y, right), T)}$$
$$\leftarrow holdsAt(relativeposition(Z, X, above), T),$$
$$holdsAt(relativeposition(Y, Z, left), T).$$

It may be noted that $H_G(B, M, E_2)$ is empty as E_2 does not contain any positive example, so the stack contains only and exactly the *Top*. Next it pops the *refinement stack* and tries to find the minimal extensions of the *Top* that covers E_2. There are two such minimal extensions , H', H'' and both of them are pushed to the *refinement stack*.

$$H' = \left\{ \begin{array}{l} \underline{holdsAt(relativeposition(X, Y, right), T)} \\ \quad \underline{\leftarrow holdsAt(relativeposition(Z, X, above), T),} \\ \quad holdsAt(relativeposition(Y, Z, left), T). \end{array} \right\}$$

$$H'' = \left\{ \begin{array}{l} \underline{holdsAt(relativeposition(X, Y, right), T)} \\ \quad \leftarrow holdsAt(relativeposition(Z, X, above), T), \\ \quad \underline{holdsAt(relativeposition(Y, Z, left), T).} \end{array} \right\}$$

The algorithm then goes on popping the top of the *refinement stack*, say H'. Since H' solves both E_1 and E_2 the condition on line 16 of Algorithm 1 is satisfied and H' is pushed into the main stack. Similarly, H'' is popped next and pushed to the main stack. At this point *refinement stack* becomes empty and iteration 1 exits as it has discovered all the minimal extensions of Top. The stack now contains H'' on top of H'.

Iteration 2: In the next iteration the algorithm pops $\langle H''_I, H''_G \rangle$ which is currently at the top of the stack. The next problem that it does not solve is E_3. It then computes $H_G(B, M, E_3)$ which contain only one element,

$$H''' = \left\{ \begin{array}{l} holdsAt(relativeposition(X, Y, below), T) \\ \quad \leftarrow holdsAt(relativeposition(Z, Y, below), T), \\ \quad holdsAt(relativeposition(X, Z, right), T). \end{array} \right\}$$

It then pushes $\langle H''_I, H''_G \cup H''' \rangle$ to the refinement stack and finds the minimal expansions of H''_I within the bound of $H''_G \cup H'''$. There will be only one such expansion, H^{final} which

will then be pushed into the refinement stack and finally into the main stack. Since H^{final} solves all three examples, the algorithms terminates returning H^{final} as the solution.

$$H^{final} = \begin{Bmatrix} holdsAt(relativeposition(X, Y, right), T) \\ \qquad \leftarrow holdsAt(relativeposition(Y, Z, left), T). \\ holdsAt(relativeposition(X, Y, below), T) \leftarrow . \end{Bmatrix}$$

4.3.2 On the Minimality of the Solution

The solution returned by algorithm 1 may not be minimal. This is because if H_I is expanded minimally to H'_I to solve a new example E, it does not ensure that H'_I is minimal with respect to the relevant subproblem. An example of this is the following: $B = \{\}$, $E_1 = \langle\{p., b., c.\}, \{a\}, \{\}\rangle$, $E_2 = \langle\{b.\}, \{\}, \{a\}\rangle$, $E_3 = \langle\{c.\}, \{a\}, \{\}\rangle$, and $M = \{\#modeh\ a, \#modeb\ b, \#modeb\ c, \#modeb\ p\}$. There are two solutions in $ILP^{DE}(B, M, \langle E_1, E_2\rangle)$: $H_1 = \{a \leftarrow c.\}$ and $H2 = \{a \leftarrow p.\}$. If H_2 is expanded first, it will produce $\{a \leftarrow p., a \leftarrow c.\}$ as the solution of $ILP^{DE}(B, M, \langle E_1, E_2, E_3\rangle)$ and since it covers all the examples, it will be returned as the solution. However, only $\{a \leftarrow c.\}$ is sufficient to cover E_1, E_2, E_3. Thus the output is not minimal. The minimal solution can be found by computing all the solutions to $ILP^{DE}(B, M, \langle E_1, E_2, E_3\rangle)$ and then discarding the ones which have a compressed version of it already in $ILP^{DE}(B, M, \langle E_1, E_2, E_3\rangle)$. However, algorithm 1 prefers efficiency over minimality and returns the first solution found.

4.4 Related Work

In recent years the field of Inductive logic programming has seen major advancements in many of its areas. Different ILP algorithms have been proposed Ray 2009; Athakravi

et al. 2013; Law, Russo, and Broda 2014; Athakravi et al. 2015; Katzouris, Artikis, and Paliouras 2015; Kazmi, Schüller, and Saygın 2017; Schüller and Kazmi 2017. Researchers have analyzed various kinds of "good" rules that cannot be learned with the current definition of entailment (called "cautious inference") and proposed an alternative to that, named as "brave inference". ILP Algorithms have thus been proposed that can do only "brave inference" Otero 2001 or both Sakama 2005; Sakama and Inoue 2009; Law, Russo, and Broda 2015. Efforts have also been made to learn answer set programs that not only contain Horn clauses but also choice rules and constraints Law, Russo, and Broda 2015. With these developments and the various systems that have been produced with these researches, people have successfully applied the paradigm of Inductive logic programming to various areas Gulwani et al. 2015; Arindam Mitra and Baral 2016a. And with these exposures to different applications, several changes are being made to the paradigm of ILP.

Recently Law, Russo, and Broda 2016 proposed context dependent learning for *ordered* answer set programs. Interested readers can refer to Law, Russo, and Broda 2016. The definition of context dependent learning in this work is an adaptation of their definition for standard ILP setting. It should be noted that even though the concept of context depending learning was proposed in Law, Russo, and Broda 2016, to solve the problem their method converts it to a standard ILP problem using choice rules. Here, I have made the first attempt to solve the problem in its original form.

In this work, I deal with the situation where there are many small distinct examples $\{(x_1, y_1), ..., (x_n, y_n)\}$. Another situation where scalability is needed, is when there is a single but large example. Works in Katzouris, Artikis, and Paliouras 2015, 2017 talk about this situation. Our work is also related to the work in logical vision Dai, Muggleton, and Zhou 2015 that aims to learn symbolic representation of simple geometric concepts.

4.5 Experiments

I have applied the proposed algorithm on two datasets. They are discussed below:

Task 6: Lists/Sets	**Task 17: Path finding**	**Task 10: Indefinite reasoning**
Sandra picked up the football there.	The office is east of the hallway.	Fred is either in the school or the park.
Sandra journeyed to the office.	The kitchen is north of the office.	Mary went back to the office.
Sandra took the apple there.	The garden is west of the bedroom.	Bill is either in the kitchen or the park.
Sandra discarded the apple.	The office is west of the garden.	Fred moved to the cinema.
What is Sandra carrying?	How do you go from the kitchen to the garden?	Is Bill in the office?

Table 13: Example question answering tasks from bAbI dataset

4.5.1 Question Answering

Recently a group of researchers from Facebook has proposed a question answering challenge Weston et al. 2015 containing 20 different tasks. Table 9 and 13 shows examples of such tasks. Each task contains 1000 or more such stories in the training data. The goal is to build a system that uniformly solves all the tasks.

The previous chapter has shown how Inductive logic programming can be used to solve the tasks. The overall method can be summarized as follows: Given the input containing a story and a question, first translate it to an Answer Set Program using a natural language parser and some handwritten rules, then use some knowledge to answer the question. In the training phase, learn the necessary knowledge. In the previous chapter, I have used *XHAIL* system to learn the knowledge. However, *XHAIL* could not scale to the entire

dataset. So I have divided the dataset. For each task then I have taken a bunch of examples together. *XHIAL* then learns from that bunch. I then add the learned hypothesis back to the background knowledge and *XHAIL* then takes the next bunch to learn from. Since knowledge learned from a group of examples is never updated again, I had to manually find a group size that will work for this dataset. The group size depended on the task and clearly it might happen that for some new task there does not exist a group size to which *xhail* can scale. In this work, I reuse the mode declarations from the previous chapter and have found that the proposed algorithm can learn all the knowledge given the input $ILP^{DE}(B, M, D_{task})$, where D_{task} contains all the 1000 examples of a task. Table 14 shows the time it has taken, the number of rules learned for each task and the accuracy for each task. The new system has achieved the same accuracy as that of the previous chapter.

4.5.1.0.1 Semantic Parsing

I have done further experiments with the task of semantic parsing. I took all the unique sentences in the training dataset of Weston et al. 2015 and the corresponding parse tree of the sentences and then trained an ILP system to do the conversion from scratch. Table 15 shows an example of this task. The training dataset contains 5458 such examples. The developed system learned a collection of 165 rules in 128 minutes from the training data which accurately parsed all the sentences in the test data.

4.5.2 Handwritten Digit Recognition

The MNIST dataset LeCun 1998 contains images of handwritten digits. Each image is a 28×28 matrix and is labeled with a number between 0 to 9 denoting the digit it represents. The value of a cell (pixel) in the matrix (image) ranges between 0 (black) to 255 (white)

#	TASK	Time	Rules	Acc
1	Single Supporting Fact	3	10	100
2	Two Supporting Facts	3	2	100
3	Three Supporting facts	_	_	100
4	Two Argument Relations	2	8	100
5	Three Argument Relations	6	20	100
6	Yes/No Questions	_	_	100
7	Counting	5	14	100
8	Lists/Sets	4	8	100
9	Simple Negation	4	13	100
10	Indefinite Knowledge	9	21	100
11	Basic Coreference	4	5	100
12	Conjunction	_	_	100
13	Compound Coreference	_	_	100
14	Time Reasoning	4	4	100
15	Basic Deduction	4	1	100
16	Basic Induction	4	1	93.6
17	Positional Reasoning	4	26	100
18	Size Reasoning	4	4	100
19	Path Finding	17	2	100
20	Agent's Motivations	2	6	100

Table 14: Performance on the set of 20 tasks. The tasks for which training is not required is marked with '-'. *Running time* is measured in *minutes*.

Sentence
Daniel journeyed to the bathroom.
ASP Representation O_i
index(1..5). lemma(1,daniel). pos(1,nn). lemma(2,journey). pos(2,vbd). lemma(3,to). pos(3,to). lemma(4,the). pos(4,dt). lemma(5,bathroom). pos(5,nn).
Positive Examples E_i^+
arg1(journey01,daniel), arg2(journey01,bathroom) .
Positive Examples E_i^-
any possible output that is not in E^+.

Table 15: An example from the semantic parsing task. For each word in the sentence the representation contains its lemma and pos tag, which are obtained using Stanford parser .

capturing the darkness at that point. In this experiment we use our ILP algorithm to learn rules that identifies digits. For that I represent the images in the following way:

1. First, I divide all cell value by 255 so that the value of each cell is in the range of $[0, 1]$.
2. For each 4×4 non-overlapping submatrix I create a super-pixel whose value is the sum of the all the pixels in that region. This gives a 7×7 size matrix representation of the original image. Note that in this reduced matrix, each cell value ranges between 0 to 16.
3. If the value of a super-pixel from the 7×7 matrix is less than 2 I consider it to be in the *off* state. If the value is more than or equal to 5 I consider it be in the *on* state. The original image is then described as two disjoint sets: 1) a set of positions where the state of the super-pixel is *off* and 2) another set where all the super-pixel are *on*.

The system learn rules on this representation. Each learned rule for a digit d simply says, if the super-pixels in certain positions are *off* and are *on* for some other positions then the image represents the digit d. The training data in the MNIST dataset contains a total of $60,000$ images with approximately $6,000$ images for each digit. To learn the rules for each digit I take all the examples of that digit and take equal amount of images that represent other digits and pass that to our algorithm. Table 16 shows the number of rules learned for each digit and the performance on the test data. Except for the digit 1, it takes 160 hours to learn the rules for each digit.

As the Table 16 suggests the performance on handwritten digit recognition is quite poor in comparison to the state-of-the-art neural network classifier Wan et al. 2013 that achieves 99.79% accuracy on this dataset. The number of rules column in Table provides insights on this high error rates. Consider the example of digit 0. If there are 5000 instances of digit 0 and the algorithm outputs $3,021$ rules that means the representation that I have chosen does

Digit	#Rules	#Test Examples	Acc(%)
0	3,021	980	60.91
1	444	1134	95.85
2	4,606	1032	32.95
3	3,661	1010	49.80
4	3,416	982	49.59
5	3,459	891	42.65
6	2,621	958	65.03
7	2,430	1028	63.52
8	3,237	978	54.50
9	2,382	1009	69.18

Table 16: Performance on handwritten digit recognition tasks. For each digit, column 2 shows the numbers of rules learned, the number instances of that digit in the test set and the percentage of instances correctly classified.

not allow good generalization. However, the representation seems to work quite well for the digit 1.

An important lesson learned from this experiment is that even though it takes a small amount of time to perform a hypothesis refinement when finding a solution H for $\langle E_1, ..., E_i \rangle$ from a solution of $\langle E_1, ..., E_{i-1} \rangle$, the algorithm needs to verify if H explains all of $\{E_1, ..., E_i\}$ before it can proceed to the next iteration. If the size of H is big (such as the case for digit recognition) and too many refinements are taking place then the algorithm spends a lot of time in the verification phase. An important future work will be to optimize this step by identifying which examples could have been affected if a hypothesis goes through refinement. Nevertheless, the algorithm is able to output a solution and does not blow up when a problem of this size is given as input. The dataset associated with all the experiments and the learned rules are available at https://goo.gl/k6AEEz. All experiments were performed on an intel i7 machine with 12 GB RAM.

Chapter 5

APPLICATION OF LKR: LEARNING TO SOLVE GENERAL ARITHMETIC PROBLEMS

In this chapter, I describe how to apply LKR paradigm for solving general arithmetic problems (Table 17). Solving math problems require applying theories related to math and reasoning with such knowledge. This makes knowledge representation and reasoning an important aspect of any automated word problem solver. However, only knowing the theory is not sufficient, we also want to learn how to apply them. This makes LKR a suitable paradigm for word math problem solving.

---

1. *Ned bought 14 boxes of chocolate candy and gave 5 to his little brother. If each box has 6 pieces inside it, how many pieces did Ned still have?*
2. *Carrie has 125 U.S. stamps. She has 3 times as many foreign stamps as U.S. stamps. How many stamps does she have altogether?*
3. *Sam, Dan, Tom, and Keith each have 15 Pokemon cards. How many Pokemon cards do they have in all?*
4. *Bert runs 2 miles every day. How many miles will Bert run in 3 weeks?*

Table 17: Sample General Arithmetic Problems

To solve an arithmetic word problem one can express the problem in terms of an equation or some equivalent representation such as expression tree or a formula from where the equation can be generated and then an equation solver can be employed to compute the solution. Existing methods following this approach generally work in two steps. In the first step, given an input problem it generates a set possible equations. In the second step, it ranks all these possibilities and picks the best one. It normally learns to do the second part

from the data. A natural question that arises here is what should the set of possible equations contain. As an answer to this question, existing systems only consider the equations that can be generated using the operators $+, -, \div, \times$ and the numbers from the text with no repetition allowed. A quick look over the problems in Table 17 shows that this restriction does not work well in practice. For example, the correct equation ($x = 125 + 3 * 125$) for the problem 2 requires the use of the number 125 twice. To overcome this issue with the existing approach one may think of allowing repetition of numbers to a certain limit such as 2, however such a measure will drastically increase the number of possibilities and introduce another limitation that the approach can solve only problems with at most one repetition. Problem 3 and 4 requiring the use of counting and additional knowledge of unit conversion pose further challenges to this approach.

In this work, I present an approach that constructs a correct equation incrementally without iterating and ranking. The general idea is as follows: given an input problem P, represent P in the knowledge representation and reasoning language of Answer set programming. Add some "domain knowledge" to the representation and pass the entire program to the answer set solver. If there is sufficient knowledge to generate the equation, the output of the solver will contain the equation.

The domain knowledge here contains information about the theories of arithmetic, including its set of operations, formulas, unit conversion knowledge and contains information about how to use those theory in problem solving. The latter is learned from data using the Inductive Logic Programming algorithm from the previous chapter. The output equation has no limit on the repetition of a number, can use as many unit conversion knowledge as necessary and can use information obtained from counting. For the problem 4 from Table 17, the output of our system will contain the following:

Here the symbol k_1 represents the fact that *there are 7 days in a week* and $q1, q2$ and

$perform(mult(k_1, q_2), 1)$
$perform(mult(q_1, k_1), 1)$
$apply(unitaryConcept(q_1, mult(k_1, q_2), x), 2)$
$apply(unitaryConcept(mult(q_1, k_1), q_2, x), 2)$
$equation(\text{“}x = 2 * (7 * 3)\text{”}, 2)$,
$equation(\text{“}x = (2 * 7) * 3\text{”}, 2)$

x respectively stand for '*Bert runs 2 miles every day*', '*in 3 weeks*' and the *unknown* in the question. The system at time step 1 decides that it needs to use an unit conversion knowledge and it can convert either q_1 or q_2. As a result of which it will know that '*Bert runs 14 miles every week*' and '*in 3 weeks*' is same as '*in 21 days*'. Here the knowledge that it has used to decide that it needs to multiply q_1 with k_1 or k_2 with q_2 is learned from the data. The knowledge that helps it to understand the meaning of multiplication, i.e. $q_1 \times k_1$ denotes '*Bert runs 14 miles every week*' is provided as part of domain knowledge. The system continues reasoning and in the next time step, using the knowledge it has gained from problem solving, decides that it can apply the *unitary formula*. The *unitary formula* says that if one item costs r unit and you get m number of items and the total cost incurred is t, then $t = m \times r$. In this case, the system finds that both $unitaryConcept(mult(q_1, k_1), q_2, x)$ and $unitaryConcept(q_1, mult(k_1, q_2), x), 2)$ are possible i.e. the value of the *unknown* can be found by multiplying the "amount of miles Bert runs per week" by the "total number of weeks" ($unitaryConcept(mult(q_1, k_1), q_2, x)$) or by multiplying the "amount of miles Bert runs per day" with the "total number of days" ($unitaryConcept(q_1, mult(k_1, q_2), x), 2)$. Once a formula is applied, the equation connecting the *unknown* with the other numbers is generated using the meaning of the formula.

In this work, the formulas are treated as relations which when extracted generates an equation. The operators on the other hand helps to infer new information from known ones. The rest of the paper is organized as follows: in section 5.1, I describe the language of Answer Set Programming; In section 5.2, I describe the representation of a word problem.

Section 5.3 describe the representation of the theories. In section 5.4, I describe the learning model. Section 5.5 describes the related works. In section 5.6, I present a detailed experimental evaluation of our system. Finally, section 5.7 concludes our paper. The code and the data has been uploaded with the submission and will be made publicly available.

5.1 Answer Set Programming

An answer set program (ASP) is a collection of rules of the form,

$$L_0 \text{ :- } L_1, ..., L_m, \textbf{not } L_{m+1}, ..., \textbf{not } L_n$$

where each of the L_i's is a literal in the sense of a classical logic. Intuitively, the above rule means that if $L_1, ..., L_m$ are true and if $L_{m+1}, ..., L_n$ can be assumed to be false then L_0 must be true Gelfond and Lifschitz 1988. The left-hand side of an ASP rule is called the *head* and the right-hand side is called the *body*. Predicates and constants in a rule start with a lower case letter or a digit, while variable terms start with a capital letter. I will follow this convention throughout the paper. A rule with empty *body* is referred to as a $fact$.

Example

$time(1)$. $time(2)$.

$multiply(X, Y, T)$:- $time(T), goodRateMultiplier(X, Y, T), not\ exists(mult(X, Y), T)$.

$exists(mult(X, Y), T + 1)$:- $multiply(X, Y, T), time(T)$.

$goodRateMultiplier(q_1, q_2, 1)$. $goodRateMultiplier(q_1, q_2, 2)$.

Consider the above program containing 6 rules. The first line of the program contains two facts saying that 1 and 2 are the possible values of *time*. The rule in the second line says that the quantity X and Y can be multiplied at time T if the relation *goodRateMultiplier* holds

between them at time T and $mult(X, Y)$ does not exist at time T. The next rule says that if X and Y are multiplied at time T then $mult(X, Y)$ exists at time $T + 1$. The last two lines describe that the relation *goodRateMultiplier* holds between the q_1 and q_2 at both time points. The output of this program will contain two additional facts, namely $multiply(q_1, q_2, 1)$ and $exists(mult(q_1, q_2), 2)$ along with the four given facts.

5.2 Problem Representation

In this work, an arithmetic word problem is represented as an ASP program that contains only facts. There are mainly two types of facts: 1) facts that denote that a quantity q has a property p, written as '$holdsAt(p(q), 1)$.'. 2) facts that describe that a relation r holds between two quantities q_1 and q_2, written as '$holdsAt(r(q_1, q_2), 1)$.'. The $holdsAt(F, T)$ predicate denotes that F holds at time point T. Here a quantity can be a number from the text, an unit conversion knowledge, number obtained from counting or the unknown corresponding to the question. Three additional predicates namely $exp(q_1, n)$, $verb(q_1, v)$ and $order(q_1, k)$ describe that the number value of q_1 is n, the verb associated with the quantity is v and the appearance order of the quantity from left to right is k. There are 24 possibilities for properties and 36 possible relations.

To compute these facts, a quantity object is created for each occurrence of a number in the text and for the question representing the unknown. For each of these quantities a list of attributes is extracted using Stanford Core NLP Manning et al. 2014. The value of each attribute is a set of words from the sentence that contains the quantity. The list of attributes includes the *verb* attribute i.e. the verb attached to the number, and attributes corresponding to Stanford dependency relations De Marneffe and Manning 2008, such as *nsubj*, *tmod*, *prep in*, that spans from the associated *verb*. A special attribute *type* denotes the kind of object

the quantity refers to. Another special attribute *rate* denotes the denominator of a rate type. For example, for the quantity "20 balls per box" or "each box contains 20 balls" the *type* is "ball" and the *rate* refers to "box". For the quantity, "Carrie has 125 US stamps" , the *type* is {US,stamps} but *rate* is ϕ. The relations correspond to whether a pair of attributes matches with each other or is a subset of another or disjoint. The set of properties include whether an attribute value is empty, whether the quantity has a non-empty rate, whether it is unknown, and the type of the verb as defined in Arindam Mitra and Baral 2016b. The following table shows the properties and relations for the quantity "2 miles" in problem 4 of table 17. The complete representation will contain additional facts regarding the remaining quantities.

$exp(q_1, \text{“2”})$. $exp(q_2, \text{“3”})$. $exp(x, \text{“}x\text{”})$.
$holdsAt(hasEmptyPrepon(q_1), 1)$.
$holdsAt(hasEmptyPrepin(q_1), 1)$.
$holdsAt(hasEmptyPrepof(q_1), 1)$.
$holdsAt(verb(q_1, run), 1)$.
$holdsAt(denotesRate(q_1), 1)$.
$order(q_1, 1)$.
$holdsAt(hasPresentTense(q_1), 1)$.
$holdsAt(subjectMatch(q_1, q_2), 1)$.
$holdsAt(exactVerbMatch(q_1, q_2), 1)$.
$holdsAt(typeMatch(q_1, x), 1)$.

5.3 Representation of Theories

Akin to any other field, the field of arithmetic has its own set of theories. In this section I show how those theories can be represented and passed to an AI system as background knowledge.

5.3.1 Formulas

Formulas are relations that when applied produces an equation. The predicate $apply(F, T)$ denotes that the formula F should be applied at time T. There are five formulas that are relevant to general arithmetic problems. Here I describe the meaning of each of those formulas along with the rules that captures that meaning.

5.3.1.1 PartWhole

The *part-whole* formula says that the value of the *whole* (W) is equal to the sum of its *parts* ($P_1, ..., P_n$). The predicate $partWhole($
$W, P_1, ..., P_n)$ represents this relation. To capture its meaning the following rules are written:

$$
\begin{array}{l}
apply(partWhole(W, P_1, ..., P_n), T) \text{ :- } partsOf(W, P_1, ..., P_n, T), \\
\qquad\qquad not\ hasMoreParts(W, P_1, ..., P_n, T). \\
equation(\text{“}V_w = V_1 + ... + V_n\text{”}, T) \text{ :- } apply(partWhole(W, P_1, ..., P_n), T), \\
\qquad\qquad exp(W, V_w), exp(P_1, V_1), ..., exp(P_n, V_n).
\end{array}
$$

Here, the first rule says that one can apply $partWhole(W, P_1, ..., P_n)$ at time T if $partsOf($
$W, P_1, ..., P_n, T)$ holds i.e. $P_1, ..., P_n$ are the parts of W at time T and W does not have any more parts at time T. $partsOf(W, P_1, ..., P_n, T)$ holds if each P_i is a part of W (represented as $partOf(P_i, W)$) and all pairs of Pi, P_j are *joinable* (represented as $joinable(P_i, P_j)$). The rules for theses two predicates, $partOf$ and *joinable* will be learned from data. The second rule shows that an equation gets created if a *part whole* formula is applied at time T. Similar rule for equation generation is written for all other formulas.

5.3.1.2 Gain

The *gain* formula says that if the value of a quantity increases by G from S and the final value is E, then $E = S + G$. The predicate $gainFormula(S, G, E)$ represents this relation. The executability condition of this formula is written as follows:

```
apply(gainFormula(Q_1,Q_2,Q_3),T):-
      goodStartGain(Q_1,Q_2,T), goodGainEnd(Q_2,Q_3,T),
      goodStartEnd(Q_1,Q_3), holdsAt(before(Q_1,Q_3),T),
      holdsAt(before(Q2,Q3),T).
```

The *before* predicate captures the order of the quantities. The definition of *goodStartGain , goodGainEnd, goodStartEnd* is learned from data.

5.3.1.3 Loss

The *loss* formula says that if the value of a quantity decreases by L from S and the final value is E, then $S - L = E$. The predicate $lossFormula(S, L, E)$ represents this relation. The executability condition of this formula is written as follows:

```
apply(lossFormula(Q_1,Q_2,Q_3),T) :-
      goodStartLoss(Q_1,Q_2,T), goodLossEnd(Q_2,Q_3,T),
      goodStartEnd(Q_1,Q_3), holdsAt(before(Q_1,Q_3),T),
      holdsAt(before(Q2,Q3),T).
```

Here, the definition of *goodStartLoss, goodLossEnd, goodStartEnd* is learned from data.

5.3.1.4 Comparison

The *comparison* formula says that if D represents the difference between two quantities B and S, then $value(B) = value(S) + value(S)$. The predicate $comparison(B, S, D)$ represents this relation. The executability condition is written as follows:

```
apply(coparison(Q_1,Q_2,Q_3),T) :- goodBigDiff(Q_1,Q_3,T),
      goodSmallDiff(Q_2,Q_3,T), goodBigSmall(Q_1,Q_2).
```

Here, the definition of *goodBigDiff, goodSmallDiff, goodBigSmall* is learned from data.

5.3.1.5 Unitary

The *unitary* formula says that if one item costs R unit and you get M number of items and the total cost incurred is T, then $T = R \times M$. The predicate $unitaryConcept(R, M, T)$ represents this relation. The following rules shows the executability condition:

```
apply(unitaryConcept(Q_1,Q_2,Q_3),T) :-
      goodRateMultiplier(Q_1,Q_2,T),
      goodRateTotal(Q_1,Q_3,T),
      goodMultiplierTotal(Q_2,Q_3).
```

The definition of *goodRateMultiplier, goodRate- Total, goodMultiplierTotal* is learned from data.

5.3.2 Operations

Operations produces new numeric quantities from existing ones. To reason with these new quantities it is important for a machine to understand their meaning. Rules are written

1.0

Operation	Properties of Output D	Relations of D
join(A, B)	$\forall p \in S_p \setminus \{isUnknown\}$, $p(D) \iff p(A) \wedge p(B)$.	Attributes value of D is the union of the attribute values of A and B.
increase(A, B)	$\forall p \in S_p \setminus \{isUnknown\}$, $p(D) \iff p(A)$.	$\forall r \in S_p \setminus \{before\}, r(D, Q) \iff r(A, Q)$. $before(Q, D) \iff before(Q, B)$. $before(D, Q) \iff before(B, Q)$.
separate(A, B)	$\forall p \in S_p \setminus \{isUnknown\}$, $p(D) \iff p(A)$.	Attributes of D is the intersection of the attributes of A and B plus their set difference.
decrease(A, B)	$\forall p \in S_p \setminus \{isUnknown\}$, $p(D) \iff p(A)$.	$\forall r \in S_p \setminus \{before\}, r(D, Q) \iff r(A, Q)$. The *before* relations follow the use case of *increase*.
multiply(A, B)	$\forall p \in S_p \setminus \{isUnknown\}$, $p(D) \iff p(A) \wedge p(B)$. $denotesRate(D) \iff denotesRate(B)$	$rateTypeMatch(D, Q) \iff rateTypeMatch(B, Q)$. $typeMatch(D, Q) \iff typeMatch(A, Q)$. $\forall r \in S_p \setminus \{before\}, r(D, Q) \iff r(A, Q) \vee r(B, Q)$.
divide(A, B)	$\forall p \in S_p \setminus \{isUnknown\}$, $p(D) \iff p(A)$. $denotesRate(D)$.	$rateTypeMatch(D, Q) \iff typeMatch(B, Q)$. $typeMatch(D, Q) \iff typeMatch(A, Q)$. $\forall r \in S_p \setminus \{before\}, r(D, Q) \iff r(A, Q)$.

Table 18: This table shows relations and properties of the derived quantity. The derived quantity D is *unknown* if any of A or B is an unknown. The *before* relations of D is determined by the *before* relations of A if A occurs after B otherwise it is determined by B. S_P and S_R respectively denotes the set of all properties and the set of all relations.

for this purpose that capture the semantics of these operations and assign name and meaning to these derived quantities. I have used a total of 7 operations and I briefly describe them here. Table 18 shows the meaning of the derived quantity for each of these operations.

5.3.2.1 Join & Increase

Both *join* and *increase* operations correspond to addition. The *increase* operation represents an increase to the value of an existing quantity such as the case in problem 6 (Table 23). The *join* action on the other hand combines two quantities to produce a new quantity with a value equal to the sum of those two quantities. An example of this action is shown in Table 23. The following rule shows the result of the *join* action:

Problem	Output
1. *Amy had 4 music files and 21 video files on her flash drive. If she deleted 23 of the files, how many files were still on her flash drive?*	*perform(join(*4, 21*),1)* *apply(LossFormula(joined(4,21),*23, *x),2)*
2. *Benny bought a soft drink for 2 dollars and 5 candy bars. He spent a total of 27 dollars. How much did each candy bar cost?*	*perform(multiply(*5, *x),1)* *apply(partWhole(*27, {2, *mult*(5, *x*)}*),2)*
3. *A company invited 18 people to a luncheon, but 12 of them didn't show up. If the tables they had held 3 people each, how many tables do they need?*	*perform(separate(*18, 12*),1)* *apply(unitary(x*, 3,*separated(18,12)),2)*
4. *Oscar's bus ride to school is 0.75 of a mile and Charlie's bus ride is 0.25 of a mile. How much longer is Oscar's bus ride than Charlie's?*	*apply(comparison(*0.75, 0.25, *x),1)*
5. *After eating at the restaurant, Sally, Sam and Alyssa decided to divide the bill evenly. If each person paid 45 dollars, what was the total of the bill?*	*perform(count(*c_1 *≡{Sally,Sam,Alyssa}),1)* *apply(unitary(*c_1, 45, *x),2)*
6. *Mika had 20 stickers. She bought 26 stickers from a store in the mall and gave 6 of the stickers to her sister. Then Mika got 20 stickers for her birthday. How many stickers does Mika have now?*	*perform(increase(*20, 26*),1)* *perform(decrease(increased(20,26),*6*),2)* *apply(gainFormula(decreased(increased(20,26),6)*, 20, *x),3)*
7. *Fred has 90 cents in his bank. How many dimes does he have?*	*apply(unitary(*90, $k_{dime-cent}$, *x),1)*

Table 19: shows how the different formulas and operations can be used to solve arithmetic word problems.

$$exists(joined(Q_1, Q_2), T + 1) :-$$
$$perform(join(Q1, Q2), T).$$

Here, $exists(joined(Q_1, Q_2), T + 1)$ denotes that the quantity $joined(Q_1, Q_2)$ exists at time $T + 1$ and thus can be used in an operation or a formula from $T + 1$ onwards. $joined(Q_1, Q_2)$ is the symbolic name that is assigned to the derived quantity. Similar rules are written for all remaining operations. The *increase* action creates a quantity named $increased(Q_1, Q_2)$.

The following rules show how the *typeMatch* relation is computed for the quantity $joined(Q_1, Q_2)$.

$tm(joined(Q_1, Q_2), Q_1)$:- $tm(Q_1, Q_2)$.
$stm(Q1, joined(Q_1, Q_2))$:- *not* $tm(Q_1, Q_2)$.
$stm(Q2, joined(Q_1, Q_2))$:- *not* $tm(Q_1, Q_2)$.
$tm(joined(Q_1, Q_2), Q)$:- $tm(joined(Q_1, Q_2), Q_2), tm(joined(Q_1, Q_2), Q_1)$.

Here *tm, stm* is used as a shorthand for *typeMatch, subTypeMatch*.The first rule says that the *type* of $joined(Q_1, Q_2)$ matches with Q_1 if Q_1 and Q_2 has the same *type*. The second and third rules say that if Q_1 and Q_2 does not have the same *type* then their *type* must be a *sub type* of the new quantity. The fourth rule computes the type match using the type match relations of Q_1 when Q_1 has the same type of the joined quantity. When *type* of Q_1 and Q_2 does not match, an additional rule checks if Q_1 and Q_2 are *sub type* of some quantity Q. If the answer is yes and there are no more subtypes of Q, then the type of Q and $joined(Q_1, Q_2)$ is declared to be same. Similar rules are written to compute the other relations and properties of the derived quantity for all the operations. Due to space limitation I briefly summarize them in Table 18.

The executability conditions of these operations are defined in terms of the same predicates that are used to define the applicability of formulas and are learned together. The following rule show the executability conditions of the *increase* operation.

$perform(increase(Q_1, Q_2), T)$:-
goodStartGain(Q_1, Q_2, T), *not*
holdsAt(canApplyGainFormula(Q_1, Q_2), T).

The $holdsAt(canApplyGainFormula(Q_1, Q_2), T)$ predicate is true if at any time T there exists another quantity Q_3 such that the gain formula is applicable to Q_1, Q_2, Q_3. This extra condition in the body eliminates unnecessary execution of *increase* operation. The executability condition of *join* is defined in terms of the *joinable* predicate. Recall that the definition of both *goodStartGain* and *join* are learned from data.

5.3.2.2 Separate & Decrease

Both *separate* and *decrease* operations correspond to subtraction. The *decrease* operation represents an 'decrease' to the value of an existing quantity (problem 6 Table 23). The *separate* operation on the other hand separates a part from the whole to produce a new part with a value equal to their difference (problem 3 Table 23). The *separate* operation creates a quantity named $separated(Q_1, Q_2)$ and execution of it is defined in terms of the *partOf* predicate. The *decrease* operation creates a quantity named $decreased(Q_1, Q_2)$ and execution of it is defined in terms of the *goodStartLoss* predicate.

5.3.2.3 Multiply & Divide

These two operations correspond to standard multiplication and division. They produce quantities named $mult(Q_1, Q_2)$ and $div(Q_1, Q_2)$ respectively and the execution of these two operations is defined using *goodRateMultiplier* and *goodMultiplierTotal* respectively.

5.3.2.4 Count

For each input problem I compute a list of countable quantities, $c_1, ..., c_m$. Each countable quantity is associated with a set of words. The value of the quantity is set to the size of that set and the *type* being equal the common WordNet Miller 1995 or NER (e.g. person) class of those words. The *count* operation decides if those quantities should be used in reasoning, which is written as $exists(c_i, T + 1)$:- $perform(count(c_i), T)$. The executability of this operation is defined using *canCount* and is learned from data.

5.3.3 Unit Change Knowledge

Along with the definitions of *formulas* and *operations* the background knowledge also contains a set of unit conversion knowledge. To provide uniformity, an unit conversion knowledge saying 1 fromUnit = y toUnit (e.g. 1 dollar = 100 cents) is represented as a quantity with value equal to *y*, *type* attribute equal to *toUnit* and the *rate* attribute being equal to *fromUnit*. Properties of an unit conversion quantity and its relations with other quantities for are computed based on the *type* and the *rate* attribute.

5.4 Training

The previous section has defined the effect of each formula and operation. However, to apply a formula or perform an operation one needs to learn the definitions of the 13 predicates, *partOf, joinable, goodStartGain, goodStartEnd, goodStartLoss, goodGainEnd, goodLossEnd, goodRateMultiplier, goodMultiplierTotal, goodRateTotal goodBigDiff, goodBigSmall, goodDiffSmall, canCount* that defines the executability conditions of the operations and formulas. In this section I present the task of learning these predicates.

Inductive Logic Programming Muggleton 1991 is a subfield of machine learning that aims to learn rules from data. To learn the rules for the 13 predicates I use the Inductive Logic Programming algorithm from Arinam Mitra and Baral 2018. The input to the algorithm is a tuple $\langle B, M, \{E_1, ..., E_n\}\rangle$. B is normally called the background knowledge. In this work B contains all the rules from the previous section that defines the theories of arithmetic and the set of unit change knowledge. The set M contains mode declarations which describe what to learn and in terms of what. In this work, it will contain the name of the 13 predicates under the category of what to learn and the name of all the properties

and all the relations as the predicates that it can use in the body of the learned rules. $E_1, ..., E_n$ are the set of examples from which the algorithm will learn. Each E_i is a tuple $< O_i, E_i^+, E_i^- >$, where O_i is called the context, E_i^+ is the set of facts that follows from the context and E_i^- is the set of facts that should not follow from the context. In this work, O_i is the logical representation of the i^{th} problem in the training dataset. The set E_i^+ contains $perform(O, T)$ and $apply(F, T)$ predicates denoting the set of operations and formulas that should be used to solve the problem. E^- set contains the operations and formulas that should not be performed. I use the meaning of the operations and formulas to populate this set. For example, if $perform(increase(Q_1, Q_2), T)$ holds that means no other operation should be done on Q_1 and Q_2. Similarly if $apply(gainFornula(Q_1, Q_2, Q_3), T) \in E_i^+$ holds I add $apply(partWhole(Q_1, Q_2, Q_3), T)$ to E_i^- but not $apply(partWhole(Q_3, Q_1, Q_2), T)$. The output of the algorithm is a collection of rules, called the hypothesis H, *s.t.*

$$H \cup B \cup O_i \vdash E_i^+, \ \forall i = 1...n$$

$$H \cup B \cup O_i \nvdash E_i^-, \ \forall i = 1...n$$

Here, $\vdash$ represents logical entailment. The above conditions describe that the output of the program containing the rules from H and B and facts from the problem O_i contains all the actions and formulas from E_i^+ and does not contain any of the formulas or actions from E_i^-.

I have annotated the dataset in Koncel-Kedziorski et al. 2015 with the $perform(O, T)$ and $apply(F, T)$ predicates. When trained on this dataset the algorithm learns a total of 134 rules. The following shows an example of a learned rule:

$$
\begin{aligned}
partOf(Q_1, Q_2, T) :- \\
&holdsAt(hasEmptySubject(Q_1), T), \\
&holdsAt(verb(Q_2, make), T), \\
&holdsAt(verb(Q_1, leaveover), T).
\end{aligned} \tag{5.1}
$$

The rule 5.1 intuitively says that the amount of left over is a subset of the amount of items made.

5.5 Related Work

Developing algorithms to solve arithmetic word problems is a long standing challenge in NLP Feigenbaum and Feldman 1963. Early years saw systems that solve the word problems in a constrained domain by either limiting the input sentences to a fixed set of patterns Daniel G. Bobrow 1964; Daniel G Bobrow 1964; Hinsley, Hayes, and Simon 1977 or by directly operating on a propositional representation Kintsch and Greeno 1985; Fletcher 1985. Mukherjee and Garain 2008 survey these works.

Among the recent algorithms, the most general ones are the work in Kushman et al. 2014; Zhou, Dai, and Chen 2015; Upadhyay et al. 2016; Huang et al. 2017; Wang, Liu, and Shi 2017; Huang et al. 2017 which can solve both arithmetic and algebraic word problems. All these algorithms try to map a word math problem to one of the n possible 'equation template's, such as $ax + b = c$, by filing the empty slots a, b, c with numbers from the text. These n templates are collected from the training data. They implicitly assume that these templates will reoccur in the new examples which is a major drawback of these algorithms.

Also none of these algorithms properly handle the use of missing unit conversion knowledge or counting.

The closest to our work are the ones in Koncel-Kedziorski et al. 2015; Roy and Roth 2015, 2016, 2017; L. Wang et al. 2018 that try find the best expression tree for the input word problem. However, the expression trees considered contain only the numbers specifically mentioned in the text and do not allow any repetition of numbers. Thus cannot solve the last three problems in Table 17. Also these algorithms explicitly assume that a single equation is needed to solve the problem. Our approach put no such restriction.

Work of Arindam Mitra and Baral 2016b is also close to our work in the sense that they have used formulas. However they have used formulas as a replacement of equation templates Kushman et al. 2014 and their method follows the generate and rank approach. Thus it has the same drawbacks to that of any other generate and rank approach. Moreover their method can only solve addition-subtraction problems.

Also there has been some work on very specific types of word problems Hosseini et al. 2014; Shi et al. 2015; Matsuzaki et al. 2017. Finally, this work is also related to *semantic parsing* Zelle and Mooney 1996; Zettlemoyer and Collins 2012, a task of mapping sentences to formal expressions. However most of the semantic parsers process single sentences whereas arithmetic problem solving requires the entire narrative to be considered together.

5.6 Experimental Evaluation

5.6.1 Dataset & Results

I evaluate our system on 3 standard datasets.

5.6.1.1 SingleEQ Dataset

This dataset Koncel-Kedziorski et al. 2015 contains a total of 508 *general* arithmetic problems requiring multiple steps. I have annotated the problems of this dataset manually to train our system. The authors have performed 5-cross validation and have reported the average. I follow the same setting.

5.6.1.2 AddSub Dataset

This dataset released by Hosseini et al. 2014 consists of a total of 395 addition-subtraction arithmetic problems for third, fourth, and fifth graders. They have reported 3-fold cross validation. Due to lack of suitable annotation, I use this dataset as test data and report the accuracy of problem solving using the *SingleEQ* dataset as the training data.

5.6.1.3 IL Dataset

This dataset Roy, Vieira, and Roth 2015 contains a total of 562 arithmetic problems involving all the four arithmetic operators. Each problem from this dataset can be solved in a single step and does not require any use of counting or outside knowledge. Due to lack of suitable annotation I have used this dataset only as test data. When trained on SingleEq dataset our system solves 369 problems giving an accuracy of 65.66%. The state-of-the-art performance Roy and Roth 2015 on this dataset is 74% (average of 5 cross validation). In this dataset each problem is repeated four times on average, thus the difference of 8.34% which corresponds to 46 problems is actually equivalent to 12 problems.

Table 20 compares the performance of our system on AddSub and SingleEq dataset.

Method	Add Sub	Single Eq
Hosseini et al. 2014	77.7	48.0
Kushman et al. 2014	64.0	67.0
Koncel-Kedziorski et al. 2015	77.0	72.0
Roy and Roth 2015	78.0	-
Arindam Mitra and Baral 2016b	86.07	-
L. Wang et al. 2018	78.5	-
Our System	75.7	80.3

Table 20: Comparison with existing systems on the accuracy of solving arithmetic problems on the Add Sub and Single Eq datasets.

There is an increase of 8.3% in the accuracy of solving problems in SingleEq dataset. One important factor behind this improvement is that the existing systems cannot solve problems where the equation uses numbers that are not mentioned in the text. The accuracy on the *AddSub* dataset is within a range of 2.8% from the accuracy of all the systems except the one in Arindam Mitra and Baral 2016b. Note that I did not train our system on the *AddSub* on contrary to the other systems that reported 3-fold cross-validation accuracy (Table 20). However, as the result shows our system generalizes quite well.

5.6.2 Error Analysis

Among all the problems 38% of the error occurs in the application of *Unitary* formula. Our system uses a set of simple patterns to extract the *rate* attribute of a numeric quantity; however, the extraction fails sometimes resulting in an error. A majority of the error (45%) occurs in the application of the *Part Whole* formula. There are several ways to describe a part whole relationship which presents rigorous challenges for part whole relation extraction. One example of a part whole problem which our system fails to solve is "*There were 3409*

pieces of candy in a jar. If 145 pieces were red and the rest were blue, how many were blue?". Since no quantity schema captures the information that "the rest were blue" it fails to identify the correct relationship. Also, unit conversion rates are not the only types of missing information. To solve the problem, "*532 people are watching a movie in a theater. The theater has 750 seats. How many seats are empty in the theater?*", it is important to know that one person normally acquires one seat in a theater. Our system does not have this knowledge and fails to solve this problem. Some problems do not specify 'has/have' verbs properly which creates issues for change problems. For the problem, "*There are 9 crayons in the drawer. Benny placed 3 more crayons in the drawer. How many crayons are now there in total*", using the tense information our system assumes that Benny placed the crayons before there were 9 crayons, which results in error. Also creating a single quantity for a number does not work always. The following problem shows an example of this type of error, "*When Joan was visited by the toothfairy, she received 14 each of quarters, half-dollars, and dimes. How much did the toothfairy leave Joan?*".

5.7 Conclusion

While a human being solves a math problem, she considers various missing knowledge that are necessary to solve the problem. Also, she is never preoccupied with the thought that each number can be used only once in the equation. It is part of the problem-solving process to decide what additional knowledge is needed and what should be the structure of the equation. In this chapter I how the LKR paradigm can be used to learn to generate the equations in such free-form manner. It is part of my future work to apply this method to word algebra problems and to analyse the additional challenges that it would create.

Chapter 6

APPLICATION OF LKR: LEARNING INTERPRETABLE MODELS OF ACTIONS FOR TRACKING STATE CHANGES IN PROCEDURAL TEXT

6.1 Introduction

With success in some reading comprehension aspects such as factoid question answering (QA) Rajpurkar et al. 2016; Joshi et al. 2017 and QA with respect to machine generated text Weston et al. 2015, newer QA challenges are being proposed that try to take the understanding to a higher level. One such dataset is ProPara Dalvi et al. 2018 where paragraphs are natural texts about processes that describe a changing world and answering the questions requires reasoning with commonsense knowledge that is implicit and not given. As of now several learning systems have been proposed for this task, all of which are neural. However a significant amount of technology has been developed in the knowledge representation and reasoning (KR) community which is well suited for the task of reasoning about dynamic world. In this chapter, I show how to effectively use such technology for reading comprehension of procedural text through thr LKR. The resulting system is interpretable and (potentially) more transferable, and it could also support more complex reasoning as needed.

Fig 14 shows an annotated paragraph from the ProPara dataset which describes “erosion by ocean”. The paragraph comprises of a sequence of 6 events each occurring in a distinct time point. The annotation tracks the states (location and existence) of three given participants:“waves”, “rocks” and “tiny parts of rocks”. For e.g., the state at time point 1, shows that the “waves” exist and the location is “ocean’; the ‘rocks’ also exist and is located on

	Participants			Time
Paragraph (Seq. of steps)	waves	rocks	tiny parts of rocks	
Water from the ocean washes onto beaches in waves.	**ocean**	beach	-	1
The waves contain sediment from the ocean.	**beach**	beach	-	2
The water and particles in it hit rocks and sand on the beach.	beach	**beach**	-	3
Tiny parts of the rocks come off the larger rocks.	beach	beach	-	4
The waves pick up sand and small rocks.	beach	beach	**beach**	5
The waves go back out into the ocean.	beach	beach	**wave**	6
	ocean	beach	ocean	7

Figure 14: An annotated paragraph from ProPara. Each filled row shows the existence and location of participants at each time point ("-" denotes "does not exist"). For example in time point 1, waves are located in the ocean.

"beach"; the "tiny pieces of rocks" however "does not exist". Here, the symbol "-" denotes "does not exist". Another special value "?" denotes "exists but location is unknown". The state at time point T describes the location and the existence of the participants before the event that starts at T. The state at T + 1 describe the state that follows the event at T and precedes the event starting at T + 1. The training dataset of ProPara contains 395 such annotated paragraphs. The goal is to develop a natural language understanding system , which given a new paragraph and a set of participants, predicts the states at each time point.

These predictions can lead to the answer of a wide variety of questions such as: (1) Where is the tiny parts of rocks located at time point 5? (2) What participants existed before the process began, but not afterwards and vice versa? (3) Which participants were converted to which other participants? and (4) Which participants moved ?

All these questions requires more than mere look up. For e.g., to answer question 1, one needs to understand that "tiny parts of rocks" came off the larger rocks during the event that started at time point 4 and during that period "larger rocks" was on "beach". So "tiny parts of rocks" should be on "beach" at time point 5. Similarly, answering questions of the

type 2, 3 and 4 require precise knowledge of the entire state sequence. As a result existing machine comprehension systems face several challenges while answering questions that require tracking states Dalvi et al. 2018.

Predicting the states at each time point mainly requires two types of knowledge. The first type of knowledge helps one to understand the deeper meaning of the events: *does it describe a location change? does it create or destroy anything?* This type of knowledge provides "explicit information" about the state of a participant. For e.g., knowledge about "washes onto" tells us that the location of "waves" is "ocean" at time point 1 and will change to "beach" after its completion (time point 2) and none of the given participants has been created or destroyed during this event (Fig 14). Similarly, knowledge about "comes off" tells us that the "tiny parts of rocks" is created during the event starting at time point 4 and will exist from time 5 and again nothing is destroyed during time point 4. From now on, we will refer to this type of knowledge as *event-centric knowledge*. All the knowledge of this type are learned from the annotated paragraphs in the training data.

The second type of knowledge helps to predict the existence or location of a participant in the absence of any "explicit information". For e.g., the event at time point 4 does not provide any information about the location of "waves". However, it can be assumed that the "waves" are still at "beach" since the last known location was "beach" and the event that started at 3 did not change its location. Similarly, "rocks" should be on "beach" since the beginning of the process as before the event at time point 3 it was on "beach" ("explicit information") and none of the events during time point 1 or 2 have changed its location. This type of knowledge is popularly known as "inertia" knowledge. The "inertia" knowledge is provided to the system as background knowledge.

Over the years the KR community has developed several formalisms describing how to represent these two types of knowledge as rules so that an automated reasoner can track

the state changes. In this work we use one such knowledge representation and reasoning language, namely Answer Set Programming Gelfond and Lifschitz 1988; Gelfond and Kahl 2014; Brewka, Eiter, and Truszczyński 2011.

The proposed system, has the three main components from the LKR paradigm. A **translation layer** which takes as input a paragraph and the participants and outputs a predicate logic representation of the text; a **reasoning layer** which takes in the formal representation of the text, the learned *event-centric* rules and the rules describing inertia and outputs the state sequence; and finally, a **learning component** which takes as input the formal representation of the paragraphs, the annotation (Fig 14), the inertia rules and outputs a set of *event-centric* rules.

This work has two key contributions: (1) It shows that with the recent advancement in question answering based meaning representation of sentences FitzGerald et al. 2018; Michael et al. 2017; He, Lewis, and Zettlemoyer 2015 and Inductive Logic Programming Arindam Mitra and Baral 2018, it is possible to learn good quality rules. To the best of my knowledge, this is the first work that uses question answering based meaning representation to learn the effect of events from a noisy dataset. The developed system, which integrates symbolic and machine learning approach, matches the state of the art performance while also providing interpretable reasoning for its predictions. (2) I analyze the learned rules and describe what additional knowledge could help the systems to generalize and perform better for state tracking, which also has gone unseen by the previous methods.

6.2 Representation

6.2.1 Paragraph & Participants

To perform symbolic reasoning or rule learning, it is important to first translate the text into a predicate logic. In this work we use a question-answering based meaning representation, namely QA-SRL FitzGerald et al. 2018 to obtain the predicate logic representation of the sentence(s). Our choice is motivated by the fact that the QA-SRL parser is trained on a much larger corpus (~250K sentences) than the ones for AMR Banarescu et al. 2013 or semantic role labelling Palmer, Gildea, and Xue 2010. The formalism of QA-SRL represents the predicate-argument structure of a sentence in terms of (question, answer) pairs where a (question, answer) pair is created for each argument of a verb in the sentence. Fig 15 shows the QA-SRL representation of a sentence in Fig 14.

Water from the ocean washes onto beaches in waves.	
Washes	
What washes into something ?	water
Where does something wash ?	onto beaches
Where does something wash from ?	the ocean
How does something wash into something ?	in waves

Figure 15: QA-SRL representation of a sentence.

To obtain a predicate logic representation of a sentence, each (question, answer) pair is wrapped inside an observedAT(V, Q, A, T) predicate, where V is the lemma of the verb for which the question is created, Q is the question with the verb replaced by the special symbol 'v', A stores the answer and T is the associated time point. For example, the sentence in Fig 15 would be translated as follows:

The second column of Fig 16 shows the predicate logic representation of all the sentences

```
observedAt("wash","what v into something ?","waves",1).
observedAt("wash","where does something v ?","onto beaches",1).
observedAt("wash","where does something v from ?","the ocean"),1.
observedAt("wash","how does something v into something ?","waves",1).
```

in paragraph of Fig 14. For each answer A of an observedAT(V, Q, A, T) predicate I also add a location(A, L) predicate to the paragraph representation if there exists a sub phrase L of A which represents a location. During the training phase, I use the available annotation to verify if a phrase L represents a location. For e.g., three predicates location("waves", "wave"), location("onto beaches", "beach") and location("the ocean", "ocean") will be added to the paragraph representation of Fig 14. We also use this data to fine tune a BERT Devlin et al. 2018 classifier to obtain a "is it a location ?" score for each sub-phrase of an answer in the test phase.

Each participant is given a symbolic name, p_i ($i = 1, 2, ...$). The predicate description(p_i, D) binds the symbolic name to the string description of the participant. For the running example of Fig 14, three description predicates, namely description(p_1, "waves"), description(p_2, "rocks") and description(p_3, "tiny parts of rocks") will be added to the paragraph representation.

The representation of the paragraph also contains refers(p_i, A, T) predicates if the answer A from the event at T contains a reference to the participant p_i. We use simple world overlap to generate the refers facts. The simple look up, may miss some of the reference. In the training data, we manually fix such error to learn better rules. No such manual annotation is used for the dev or the test set.

6.2.2 Events

To track the states of participants it is important to know whether the event(s) in a sentence creates, destroys or moves any participants. Accordingly, a high-level meaning

representation scheme is devised for the sentences, which contains the following five predicates:

1. create(P, T): the participant P is created during the event at time T.
2. destroy(P, T): the participant P is destroyed in the event at time T.
3. beforeLocation(P, L, T): the location of the participant P before the event at T is L.
4. afterLocation(P, L, T): the location of location of the participant P after the event at T is L.
5. terminate(P, T) : the location of the participant P is changed during the event T, but new location is not specified.

Fig 16 shows the high-level representation of the sentences from the paragraph in Fig 14 along with its QA-SRL based representation. For e.g., the first sentence is represented in terms of two facts: beforeLocation(p_1, "ocean", 1) and afterLocation(p_1, "beach", 1), which describes that p_1 ("waves") was at "ocean" at time point 1 and is at "beach" at time point 2. In the training phase, the proposed system learns rules that predicts the high-level representation of a sentence from its QA-SRL based representation which comprises of *observedAt, location, description* and *refers*. One sample rule might look like the following:

```
IF observedAt("wash","where does something v ?",A1,T),
   observedAt("wash","how does something v into
   something ?",A2), location(A1,L), refers(A2,P)
Then afterLocation(P,L,T)
```

6.3 Reasoning

The reasoning module uses the learned *event-centric* knowledge to first extract a high level representation of the events from the QA-SRL based paragraph representation. It then

Sentences	QA-SRL based representation	High level representation
Water from the ocean washes onto beaches in waves.	observedAt("wash","what v into something ?","waves",1). observedAt("wash","where does something v ?","onto beaches",1). observedAt("wash","where does something v from ?","the ocean"),1. observedAt("wash","how does something v into something ?","waves",1).	afterLocation(p1,``beach",1). beforeLocation(p1,``ocean",1)
The water and particles in it hit rocks and sand on the beach.	observedAt("hit","what v something ?","the water and particles in it",3). observedAt("hit","what does something v ?","rocks and sand",3). observedAt("hit","where does something v ?","on the beach",3).	beforeLocation(p2,``beach",3)
Tiny parts of the rocks come off the larger rocks.	observedAt("come","what v off something ?","tiny parts of the rocks",4). observedAt("come","what does something v off?","the larger rocks",4).	create(p3,4). afterLocation(p3,"beach",4).
The waves pick up sand and small rocks.	observedAt("pick","what v up something ?","the waves",5). observedAt("pick","what does something v up?","small rocks",5).	afterLocation(p3,"wave",5).
The waves go back out into the ocean.	observedAt("go","what v somewhere ?","the waves",6). observedAt("pick","where does something v ?","into the ocean",6).	afterLocation(p1,"wave",6). afterLocation(p3,"wave",6).

Figure 16: The QA-SRL based and High level representation of some of the sentences from Fig 14.

uses the high level representation to predict the state sequence. The prediction function, utilizes the notion of a *critical point*. For a participant P, a time point T is a *critical point* if any the following is true:

1. create(P, T − 1)
2. destroy(P, T − 1)
3. ∃L.afterLocation(P, L, T − 1)
4. ∃L.beforeLocation(P, L, T)
5. terminate(P, T − 1)

For the running example, according to the high level description in Fig 16, the critical points of p_1 are 1, 2 and 7. The reasoning module first computes the state of a participant in the *critical time points* using the definition of the five high level predicates. For e.g., it would infer that location of p_1 is "ocean" at time point 1, "beach" at time point 2 and "ocean" at time point 7 using the following information respectively beforeLocation(p_1, "ocean", 1), afterLocation(p_1, "beach", 1) and afterLocation(p_1, "ocean", 6). A participant P does not exist at T if it is destroyed during the event at T-1 i.e. destroy(P, T − 1) is true. The location of a participant is unknown at T if it is created during the event at T-1 but the event does

not provide any information about "afterLocation" and the event at the next critical point also does not provide any "beforeLocation". Similarly, the location of a participant can also be unknown at T if the previous location is terminated i.e. terminate(P, T − 1) is true and the event at T-1 does not provide any "locationAfter" information neither the next event provide any "beforeLocation".

The state at a non-critical time point is then computed with a set of inertia rules on a case-by-case basis. For any participant P, in a non-critical time point T only one of this must be true:

Case 1: There exists no critical point for P before or after T.
Case 2: There is no time point before T which is a critical point for P but there is one after it.
Case 3: There is no time point after T which is a critical point for P there is one before it.
Case 4: There is a critical point for P both before and after T.

In case 1, it is assumed that P does not exist at any time point T. In case 2, the location is L if the right critical point (the first critical point after T) T′ provides "beforeLocation" i.e. beforeLocation(P, L, T′) is true otherwise it is unknown. For e.g., the location of "rocks" (p_2) is "beach" at time time point 1, as the right critical point i.e. time point 3 provides beforeLocation(p_2, "beach″, 3). In case 3, the state at T is same as the state of the left critical point (the critical point that appears just before T). For e.g., the location of "wave" is "beach" at time point 5 as it was on "beach" at the left critical point which is 2. In case 4, the state at T is same as the state of the left critical point if the event at right critical point does not provide any "beforeLocation". Otherwise it might take any of the two values and there might be several possibilities.

The reasoning module also use two more defeasible rules to derive more afterLocation facts using the states of the other participants. One such rule, named "conversion" derives afterLocation(P, L, T − 1) if created(P, T − 1) is true and P′ participates in the event at T-1 and had location L at T-1 and the event at T-1 provides no information of "afterLocation" for P. In the running example, this rule triggers and output afterLocation(p_3, "beach″, 5) as tiny part of rock came off the larger rocks. Another rule, namely the container rule, infer afterLocation(P, L, T − 1), if the location of P at T-1 is P′ and afterLocation(P′, L, T − 1) is true. This rule infers afterLocation(p_3, "ocean″, 6) as the waves contain the tiny rocks while going back to the ocean.

6.4 Learning Commonsense Event-Centric Knowledge and Analyzing Learned Knowledge

Three separate learning tasks are created, one for each of the create, destroy and location change (terminate, beforeLocation,afterLocation) rules. The inductive logic programming (ILP) algorithm of I^2XHAIL Arindam Mitra and Baral 2018 is used for learning. The input to the learning algorithm, the working of the I^2XHAIL algorithm and a fine-grained analysis of the learned rules are presented in this section.

6.4.1 Learning Rules that Describe Creation

To learn the "create" rules with the Inductive Logic Programming algorithm of I^2XHAIL, each annotated paragraph from the training dataset is converted into an ILP sample S_i of the following form: $\langle O_i, E_i^+, E_i^- \rangle$, where O_i called an *observation* contains the predicate logic representation of the paragraph and the participants; E_i^+ is a set containing the true *create*

events and E_i^- is a set containing the false *create* events. For the running example, E_i^+ will contain only one fact create(p_3, 4) and E_i^- will contain all the grounding of *create* predicate that does not happen such as {create(p_3, 3), create(p_1, 1), create(p_2, 3)...}. The I^{2}XHAIL system takes a sequence of such examples $S_1, ..., S_n$, the name of the predicates that should be learned (in this case *create*), some background knowledge B (in this case empty) and outputs a set of rules H such that the following holds:

$$O_i \cup B \cup H \vdash E_i^+, \forall i = 1..n$$

$$O_i \cup B \cup H \nvdash E_i^-, \forall i = 1..n$$

Here, $\vdash$ represents logical entailment using stable model semantics Gelfond and Lifschitz 1988. In simple words, the above two equations describe that the output of the program P_i containing the rules and facts from H, B and O_i must contain all the facts in E_i^+ and must not contain any of the element from E_i^- . The I^{2}XHAIL algorithm finds a solution H incrementally, i.e., it first finds a solution for only S_1, then it expands the solution so it solves both S_1 and S_2 and the process continues until it finds a set of rules that solves all the n samples. It obtains the solution(s) of S_1 in three steps, called the abductive, deductive and the inductive step.

6.4.1.1 Abductive

In the abductive step, it finds out several minimal collection of the grounded *create* atoms [1] which if added to O_i and B, entails E_i^+ and does not entail any of E_i^-. These sets are called *abducibles*. One such abducible Δ for the running example is {create(p_3, 4)}.

[1] A predicate is grounded when the variable arguments are replaced with constants

6.4.1.2 Deductive

In the second stage, it considers the possible causes of each ground predicate in Δ and creates the most specific rule for each of them by adding the possible causes into the body of a rule. For the running example, there is only one predicate in Δ, namely create(p_3, 4) and the possible causes are all the facts about time point 4. Thus it will create the following rule:

```
IF observedAt("come","what v off something ?", "tiny parts of the rocks",4),
   observedAt("come","what does something v off?","the larger rocks",4),
   refers(p3,"tiny parts of the rocks",4), refers(p2,"the larger rocks",4)
  ,describe(P2,"rocks"), describe(p3,"tiny parts of the rocks")
THEN create(p3,4)
```

6.4.1.3 Inductive

In the third stage, it tries to generalize the rule as much as possible by removing elements from the "IF" condition or by replacing the constants (such as "tiny parts of the rocks", 4, "the larger rocks") by variables. The rule (H_1) that it learns from this example is the following:

```
IF observedAt("come","what v off something ?",X,T), refers(E,X,T)
THEN create(E,T)
```

Next it takes only S_2 and perform the abductive and the deductive steps to obtain another set of most specific rules. The new most specific rules along with the previous one from S_1 then provides a upper bound for the solution of S_1 and S_2. It then expands H_1 along the new upper bound until it finds a set of rules that solves both S_1 and S_2.

6.4.1.4 Analysis

Fig 17 shows some of the rules that the system learns from this task. There are two types of *create* rules. The first set of rules captures how some verbs such "provide", "form", "make" are used to describe a *create* event. The first three rules in Fig 17 show few such examples. The second type of rule is specific to the creation of a certain participant. The last three rules that respectively describe evaporation creates water vapour, melting of cans creates molten metal and pupa grows create adult butterfly are examples of this kind. If a new paragraph contains a participant such as "lava" whose creation involves a very specific event such as "magma going outside of volcano", our system would not be able to detect the creation. This shows what kind of additional knowledge might be given to the system to help to detect *create* events.

create(P,T) IF observedAt("provide","what does something v ?",X,T), refers(P,X,T).
create(P,T) IF observedAt("form","what is being v ?",X,T),refers(P,X,T).
create(P,T) IF observedAt("make","what does someone v ?",X,T),refers(P,X,T).
create(P,T) IF observedAt("evaporate","what v ?","water",T),description(P,"water vapor").
create(P,T) IF observedAt("melt","what v something ?","the cans",T),description(P,"molten metal").
create(P,T) IF observedAt("grow","what v ?","the pupa",T),description(P,"adult butterfly").

Figure 17: Examples of A is *true* IF B is *true* rules that our system learns to identify create events.

6.4.2 Learning Rules for Destroy

The task of learning *destroy* rules is similar to that of the *create* rules; O_i contains the representation of the paragraph and the participant, E_i^+ contains the true destroy events and the E_i^- contains the destroy events that should not be predicted. However instead of learning the rules which directly identify *destroy* events, I define the *destroy* event in terms of two predicates ("normallyDestroys" and "exception") and learn the definition of those two lower level predicates.

```
IF normallyDestroy(P,T),not exception(P,T)
THEN destroy(P,T)
```

The above rule describes, if the event at time point T normally destroys its participant, then P can be assumed to be destroyed at T unless P is a special case. For, e.g, water turning into water vapour destroys water, tadpole turning into adult frog ends its tadpole phase but water vapour turning into cloud does not destroy the water vapor. Thus in this case two rules will be learned. One that captures that the "turn" event normally destroys its agent and "water vapor" is an exception to this if it is transformed into "cloud".

6.4.2.1 Analysis

Fig 18 shows some of the rules that the system learns to identify *destroy* events. The rules for *destroy* can be divided into three categories. The first type of rule describes how some verbs such as "eat", "decompose" and "form" (e.g., Sulfur in the coal combines with oxygen to *form* sulfur oxides) are normally used to describe destruction. The first three rules in Fig 18 show examples of such rules. The second type of rules capture exceptions such

normallyDestroy(P,T) IF observedAt("eat","what does someone v ?",X,T), refers(P,X,T).
normallyDestroy(P,T) IF observedAt("decompose","what v ?",P,T).
normallyDestroy(P,T) IF observedAt("form","what v something ?",P,T).
exception(P,T) IF normallyDestroy(P,T), description(P,"caterpillar"), create(P1,T), description(P1,"cocoon").
exception(P,T) IF normallyDestroy(P,T), description(P,"caterpillar"), create(P1,T), description(P1,"saliva").
normallyDestroy(P,T) IF observedAt("hatch","what v ?","the cocoon",T),description(P,"pupa").
normallyDestroy(P,T) IF observedAt("flow","what v from something ?","magma",T), description(P,"magma"), observedAt("flow","how does something v from something ?","in the form of lava",T).

Figure 18: Examples of A is *true* IF B is *true* rules that our system learns to identify destroy events.

as caterpillar forming cocoon does not destroy the caterpillar (unlike the sulfur). Similarly, mixing food with saliva does not destroy the food unlike some chemical reactions. The 4*th* and 5*th* rules from Fig 18 are learned to deal with these two exceptions. The third type of rules are very specific to the destruction of a certain participant. The last two rules that describe 'hatching of cocoon marks the end of pupa" and "magma when flows outside as lava is no longer described as magma" are examples of this type of rules. Rules of first type are relatively small in number and can be learned well from a annotated dataset like ProPara. Several background knowledge such as examples of chemical reaction can help to understand the *exceptions* better.

6.4.3 Learning Rules for Location Changes

Unlike *destroy*, the *location change* events do not depend much on the participants in the domain of ProPara. For this task, O_i contains the predicate logic representation of paragraph and participants and the gold *create* and *destroy* events that happens during the process; the background knowledge B contains all the rules from section 3, and the E_i^+ contains the

state descriptions at each time point (Fig 14). From these it learns to identify locationAfter, terminate and locationBefore. Fig 19 shows few rules that describe location changes.

locationBefore(locationOf(P,L),T) **IF** lobservedAt("travel","what does something v from ?",L,T), eobservedAt("travel","what v ?",P,T).
locationBefore(locationOf(P,L),T) **IF** lobservedAt("fill","where is something being v from ?",L,T), eobservedAt("fill","what is something v with ?",P,T).
locationAfter(locationOf(P,L),T) **IF** eobservedAt("put","what is v ?",P,T), lobservedAt("put","what is something v in ?",L,T).
terminate(locationOf(P),T) **IF** eobservedAt("evaporate","what v from something ?",P,T).
terminate(locationOf(P),T) **IF** eobservedAt("launch","what is being v ?",P,T).
locationAfter(locationOf(P,L),T) **IF** eobservedAt("turn","what does something v into ?",P,T), value(L,"atmosphere").

Figure 19: Examples of rules that our system learns to identify move events. Here, *eobservedAt(V,Q,P,T)* stands for *observedAt(V,Q,A,T)* and *refers(P,A,T)*. Similarly *lobservedAt(V,Q,L,T)* stands for *observedAt(V,Q,A,T)* and *location(A,L)*.

6.4.3.1 Analysis

Fig 19 shows some of the rules that the system learns to identify *move* events. The rules for *move* can be divided into two categories. The first type describes how some verbs such as "travel", "fill", put", "evaporate" and "launch"are normally used to describe location changes. The first five rules in Fig 19 are examples of this type. Sometimes, the event and participant together can determine the location on the next time point, even if the sentence does not explicitly mention it. The second category refers to such rules. The last (over-generalized) rule in Fig 19 shows an example of such rule describing that water after turning into water vapour normally moves to atmosphere.

6.5 Related Works

The task of state tracking is a long studied problem in AI. However, there exist very few benchmarks to track the states in procedural text. The bAbI Weston et al. 2015 question answering dataset which also requires state tracking contains synthetically generated sentences with a very simplified grammar and few events. Thus the performance of the systems on the bAbI dataset does not carry forward to the ProPara dataset. For e.g., two state-of-the-art neural models on the bAbI task namely EntNet Henaff et al. 2016 and QRN Seo, Min, et al. 2016 gets only 39.40% and 41.10% F1 score on ProPara which is ~15% less than the performance of our method.

As of now, the following five systems have been proposed for ProPara.

6.5.1 ProComp

The ProComp Clark, Dalvi, and Tandon 2018 system uses formal reasoning to track the state changes. While doing so it uses a set of handcrafted rules describing the effects of all the verbs in VerbNet. The rules do not depend on the nature of the participant. By learning the rules, I observe how the very specific nature of the participants plays a crucial role in decision making which the ProComp system does not verbalize.

6.5.2 ProLocal

The ProLocal Dalvi et al. 2018 system uses a deep neural classifier to find out if the sentence (event) at time point T destroys, creates or moves a participant P. A set of rules (similar to our inertia rules) are then used to propagate these local predictions to find the

states at each time point. In our work, instead of a neural classifier I have used an Inductive Logic Programming algorithm. As a result, our system is explainable like ProComp. Also experiments show that our work performs better than the ProLocal system.

6.5.3 ProGlobal

The ProGlobal Dalvi et al. 2018 system aims to learn a state transition function, which takes as input the entire paragraph, a participiant, the sentence at the current time point and outputs the state as "-", "?" or a phrase from the paragraph describing the location in the next state.

6.5.4 ProStruct

The ProStruct Tandon et al. 2018 system improves upon the ProLocal and the ProGlobal system by injecting commonsense knowledge (e.g., do not move a participant which has been destroyed) as hard and weak constraints. It uses a similar neural network architecture to that of the ProGlobal and the ProLocal system to explore the several possible next states. It then uses commonsense constraints to filter out "bad" states.

6.5.5 KG-MRC

The KG-MRC Das et al. 2018 system represents the state as a bipartite graph where the nodes on one side corresponds to entities and nodes on the other side corresponds to location phrases. A neural network then takes the graph (state) at current time, the entire

	Precision	Recall	F1
ProLocal	77.4	22.9	35.3
QRN	55.5	31.3	40.0
EntNet	50.2	33.5	40.2
ProGlobal	46.7	52.4	49.4
ProStruct	74.2	42.1	53.7
KG-MRC	64.52	50.68	56.77
ProKR (ours)	76.00	45.10	56.60

Table 21: Results on the prediction task (test set).

paragraph, the sentence in the current time point and computes the graph for the next time point.

6.6 Results

Table 21 compares the results of our system. All the solvers are evaluated with the ProPara evaluator script on the following four metrics: (Q1) What are the inputs to the process? (Q2) What are the outputs of the process? (Q3) What conversions occur, when and where? (Q4) What movements occur, when and where?

Inputs to a process are defined as participants that existed before the process started, but not at the end. Outputs are participants that did not exist at the start, but did at the end. A conversion is when some participants are destroyed and others are created. Finally, a movement is an event where an entity changes location.

The evaluator script Tandon et al. 2018 computes a F1 score for each question by comparing the gold and predicted answers. For Q1 and Q2, this is straightforward as answers are atomic (i.e., individual names of entities). For Q3, each answer is a 4-tuple (convert-from, convert-to, location, sentence-id) and some answers may only be partially correct. To score partial correctness, the evaluator script pair gold and predicted answers by

requiring the sentence-id in each to be the same, and then score each pair by the Hamming distance of their tuples. For Q4, each answer is also a 4-tuple (entity, from-location, to-location, sentence-id), and the same procedure is applied. The four F1 scores are then macro-averaged. The total number of items to predict in the train/dev/test partitions is 7043/913/1095.

6.6.1 Error Analysis

6.6.1.1 Missing Verb

Sometimes the sentence in a paragraph does not explicitly mention the verb. For e.g., "*The air travels through your windpipe. Into your lungs.*" In this cases, the predicate representation fails to capture that air traveled to lungs, which results in error.

6.6.1.2 Symbolic Interpretation of Questions

In the current implementation questions are treated symbolically. However, "What forms?", "what is formed?", "what has been formed?" all represent the same thing and symbolic similarity does not capture that, which affects the generalizability.

6.6.1.3 Discourse

The current system does not learn to understand discourse relations such as coreference, which affects the performance. In some cases, coreference can be particularly challenging since two participants have the same description. Also sometimes, a participant in an event is

implicitly mentioned. Consider the following sentences: "*Trash is removed from everything else. Goes to a landfill.*". Here what goes to landfill is not directly mentioned which results in errors.

6.7 Conclusion

Reasoning about actions and events have been long studied in the KR community and have led to the development of a variety of tools. However, they do not work directly with natural language text, and require experts to manually write the knowledge. This makes it challenging to use them in reading comprehension. In this paper, I show that with the recent development in sentence parsing (QA-SRL) and Inductive Logic Programming from chapter 4, it is possible to learn interpretable models of actions from the training data which then can be used with KR formalisms to reason about procedural text. Moreover, experiments reveal that it is crucial to have some participant specific knowledge base to generalize better, which to the best of my knowledge has not been discussed before and shows the importance of an interpretable solution.

Chapter 7

TKR PARADIGM: DECLARATIVE QUESTION ANSWERING OVER KNOWLEDGE BASES CONTAINING NATURAL LANGUAGE TEXT WITH AN APPLICATION OF ANSWERING LIFE CYCLE QUESTIONS

In the previous chapters, we have seen how to develop interpretable solutions for the question-answering tasks for which the input can be translated to some predicate representation, i.e., a set of category 1 Non-Extractive Reading Comprehension tasks. However, currently, we do not have a parser, that works well for all sentences. In thus chapter, we thus try to avoid the need of a general purpose parser. Particularly, we want to develop solutions for Category 2 Non-Extractive Reading Comprehension tasks, where we have the missing "additional knowledge" that is required to correctly answer the questions however we do not have a parser that can translate the passage sentences well. To pursue this goal, I have developed a dataset which contains a particular genre of school level science questions (Clark et al. 2018), namely questions about life cycles (and more generally, sequences).

To get a better understanding of the "life cycle" questions and the "hard" ones among them consider the questions from Table 22. The text in Table 22, which describes the life cycle of a frog does not contain all the knowledge that is necessary to answer the questions. In fact, all the questions require some additional knowledge that is not given in the text. Question 1 requires knowing the definition of "middle" of a sequence. Question 2 requires the knowledge of "between". Question 3 on other hand requires the knowledge of "a good indicator". Note that for question 3, knowing whether an adult frog has lungs or if it is the adult stage where the frog loses its tail is not sufficient to decide if option (A) is the

indicator or option (B). In fact an adult frog satisfies both the conditions. An adult frog has lungs and the tail gets absorbed in the adult stage. It is the uniqueness property that decides that option (B) is an indicator for the adult stage. We believe to answer these questions the system requires access to this knowledge.

Life Cycle of a Frog

order: egg → tadpole → tadpole with legs → adult

egg - Tiny frog eggs are laid in masses in the water by a female frog. The eggs hatch into tadpoles.

tadpole - (also called the polliwog) This stage hatches from the egg. The tadpole spends its time swimming in the water, eating and growing. Tadpoles breathe using gills and have a tail.

tadpole with legs - In this stage the tadpole sprouts legs (and then arms), has a longer body, and has a more distinct head. It still breathes using gills and has a tail.

froglet - In this stage, the almost mature frog breathes with lungs and still has some of its tail.

adult - The adult frog breathes with lungs and has no tail (it has been absorbed by the body).

1. What is the middle stage in a frog's life? (A) tadpole with legs (B) froglet

2. What is a stage that comes between tadpole and adult in the life cycle of a frog? (A) egg (B) froglet

3. What best indicates that a frog has reached the adult stage? (A) When it has lungs (B) When its tail has been absorbed by the body

Table 22: A text for life cycle of a Frog with few questions.

Since this additional knowledge of "middle", "between", "indicator" (and some related ones which are shown later) is applicable to any sequence in general and is not specific to only life cycles, we aim to provide this knowledge to the question answering system and then plan to train it so that it can recognize the question types. The paradigm of declarative programming provides a natural solution for adding background knowledge. Also the

existing semantic parsers perform well on recognizing questions categories. However the existing declarative programming based question answering methods demand the premises (here the life cycle text) to be given in a logical form. For the domain of life cycle question answering this seems a very demanding and impractical requirement due to the wide variety of sentences that can be present in a life cycle text. Also a life cycle text in our dataset contains 25 lines on average which makes the translation more challenging.

The question that we then address is, "can the system utilize the additional knowledge (for e.g. the knowledge of an "indicator") without requiring the entire text to be given in a formal language?" I show that by using Answer Set Programming and some of its recent features (function symbols) to call external modules that are trained to do simple textual entailment, it is possible do declaratively reasoning over text. I have developed a system following this approach, which I will refer to as the TKR paradigm that answers questions from a given text by declaratively reasoning about background concepts such as "middle", "between", "indicator" over premises given in natural language text. To evaluate this method a new dataset has been created with the help of Amazon Mechanical Turk. The entire dataset contains 5811 questions that are created from 41 life cycle texts. A part of this dataset is used for testing. The developed system achieved up to 18% performance improvements when compared to standard baselines. The dataset and code is available from https://goo.gl/YmNQKp.

7.1 Background

7.1.1 Answer Set Programming

An Answer Set Program is a collection of rules of the form,

$$L_0 \text{ :- } L_1, ..., L_m, \textbf{not } L_{m+1}, ..., \textbf{not } L_n.$$

where each of the L_i's is a literal in the sense of classical logic. Intuitively, the above rule means that if $L_1, ..., L_m$ are true and if $L_{m+1}, ..., L_n$ can be safely assumed to be false then L_0 must be true Gelfond and Lifschitz 1988. The left-hand side of an ASP rule is called the *head* and the right-hand side is called the *body*. The symbol :- ("if") is dropped if the *body* is empty; such rules are called facts. Throughout this paper, predicates and constants in a rule start with a lower case letter, while variables start with a capital letter. The following ASP program represents question 3 from Table 22 with three facts and one rule.

Listing 7.1: a sample question representation

```
qIndicator(frog,adult).
option(a, has(lungs)).
option(b, hasNo(tail)).
ans(X):- option(X,V), indicator(O,S,V),
         qIndicator(O,S).
```

The first *fact* represents that question 3 is an 'indicator' type question and is looking for something which indicates that a *frog* is in the *adult* stage. The later two *facts* roughly describes the two answer choices, namely "(a) when it has lungs" and "(b) when its tail has been absorbed by the body". The last rule describes that for an indicator type question, the option number X is a correct answer if the answer choice V is an indicator for the organism O being in stage S i.e. if $indicator(O, S, V)$ is true.

Aggregates A rule in ASP can contain aggregate functions. An aggregate function takes as input a set. ASP has four built-in aggregates namely *#count, #max, #min, #sum* which respectively computes the number of elements in a set, the maximum, minimum or the sum

of numbers in the set. The follows rule defines the concept of an 'indicator' using the *#count* aggregate.

Listing 7.2: Defining Indicator of a stage

```
indicator(O,Stage,P) :-
  stageFact(O,Stage,P),
  #count {stageFact(O,S1,P)} = 1.
```

Here, $stageFact(O, Stage, P)$ captures the attributes P that are true when the organism O is in the stage S. The above rule then describes that P is an indicator for O being in stage S if P is true in S and it is only true in S i.e. the total number of stages $S1$ where $Prop$ is true is one.

String valued Terms The object constants in ASP can take *string* values (written inside quotes " "). This is useful while working with text. For example, the options in the question 3 can also be represented as follows:

```
option(a, "when it has lungs").
option(b, "when its tail has
          been absorbed by the body").
```

Function Symbols A *function symbol* allows calling an external function which is defined in a scripting language such as *lua* or *python* Calimeri et al. 2008. An occurrence of a function symbol making an external call is preceded by the '@' symbol. For e.g., $@stageFact(O, S, P)$ denotes a function symbol that calls to an external function named *stageFact* which takes three arguments as input. A function symbol can return any simple term such as *name*, *number* and *strings* as output.

7.1.2 QA using Declarative Programming

A question answering (QA) system that follows declarative programming approach primarily requires three components: a **semantic parser** $\mathcal{SP}$, a **knowledge base** $\mathcal{KB}$ and a **set of rules** (let's call it **theory**) $\mathcal{T}$.

- The goal of the **semantic parser** $\mathcal{SP}$ is to translate a given question into a logical form.
- The $\mathcal{KB}$ provides facts or "premises" with respect to which the question should be answered. For e.g. for the frog life cycle the $\mathcal{KB}$ might look like the following:

Listing 7.3: A sample KB for part of the Frog life cycle

```
stageFact(frog,tadpole,has(tail)).
stageFact(frog,froglet,has(lungs)).
stageFact(frog,froglet,has(tail)).
stageFact(frog,adult,has(lungs)).
stageFact(frog,adult,hasNo(tail)).
```

- The *theory* $\mathcal{T}$ contains inference enabling rules.

  To answer a question, the system first translates the question into a logical form and then combines that with the $\mathcal{KB}$ and the $\mathcal{T}$ to create a consolidated program. The output (models) of which provides the answer.

  For the running example of the 'indicator' question ($Q3$ from Table 22) if the theory $\mathcal{T}$ contains the rule in listing 7.2, some semantic parser provides the question representation in listing 7.1 and the $\mathcal{KB}$ contains the facts in listing 7.3, then the output will contain the deduced fact $ans(b)$ describing that option (B) is the correct answer. This is because, the rule in listing 7.2 will deduce from

the $\mathcal{KB}$ that $indicator(frog, adult, hasNo(tail))$ is true. The last rule in listing 7.1 will then conclude that $ans(b)$ is true. Since there is no reason to believe that $indicator(frog, adult, has(lungs))$ is true, $ans(a)$ will not be part of the output (model). The semantics of ASP is based on the stable model semantics Gelfond and Lifschitz 1988. For further details interested readers can refer to Gebser et al. 2012; Gelfond and Kahl 2014.

7.2 Proposed Approach

The issue in the running example is that it is difficult to get the facts in terms of $stageFact/3$ predicate and in the actual $\mathcal{KB}$ we do not have facts in this format. Rather we have the life cycle texts (Table 22) describing the facts. To deal with this we replace such predicates with two external function symbols, namely *generate* and *validate*.

Generate A *generate function* for a predicate takes the arguments of the predicate and returns a textual description of the predicate instance following some template. For example, a generate function for *stageFact* can take *(frog, adult, hasNo(tail))* as input and returns a string such as "an adult frog has no tail" or if it is for a predicate named *parent* it can take *(x, y)* and return "*x* is a parent of *y*".

Validate A *validate function* takes a string describing a proposition (e.g. an adult frog has no tail) and validates the truthfulness of the proposition against a $\mathcal{KB}$ containing text (e.g. Table 22). For now let us assume a *validate function*

Question Template	Example Question	Instantiated Template	#Qs
$qLookup(O)$	How do froglets breath?	$qLookup(\text{“}frog\text{”})$	2525
$qDifference(O, S1, S2)$	What is an adult newt able to do that a tadpole cannot?	$qDifference(\text{“}newt\text{”}, \text{“}tadpole\text{”}, \text{“}adult\text{”})$	167
$qIndicator(O, S)$	When do you consider a penguin to have reached the adult stage?	$qIndicator(\text{“}penguin\text{”}, \text{“}adult\text{”})$	125
$qNextStage(O, S)$	A salmon spends time as which of these after emerging from an egg?	$qNextStage(\text{“}salmon\text{”}, \text{“}egg\text{”})$	346
$qStageBefore(O, S)$	Newt has grown enough but it is not yet in the tadpole stage, where it might be?	$qStageBefore(\text{“}newt\text{”}, \text{“}tadpole\text{”})$	123
$qStageBetween(O, S1, S2)$	What is the stage that comes after egg and before eft in the newt life cycle?	$qStageBetween(\text{“}newt\text{”}, \text{“}egg\text{”}, \text{“}eft\text{”})$	123
$qStageAt(O, P)$	What stage a longleaf pine will be in when it is halfway through its life?	$qStageAt(\text{“}longleaf\ pine\text{”}, middle)$	520
$qCorrectlyOrdered(O)$	To grow into an adult, fleas go through several stages. Which of these is ordered correctly?	$qCorrectlyOrdered(\text{“}flea\text{”})$	43
$qCountStages(O)$	From start to finish, the growth process of a wolf consists of how many steps?	$qCountStages(\text{“}wolf''\text{”})$	113
$qIsAStageOf(O)$	The growth process of lizards includes which of these?	$qIsAStageOf(\text{“}lizard\text{”})$	1500
$qIsNotAStageOf(O)$	To grow into an adult, fleas go through 4 stages. Which of these is not one of them?	$qIsNotAStageOf(\text{“}flea\text{”})$	227

Table 23: Question templates and total number of questions for each question category.

returns 1 or 0 depending on whether the proposition is true or false according to the text in the $\mathcal{KB}$.

With this transformation the "indicator" rule from listing 7.2 will look as follows:

```
indicator(0,Stage,Prop) :-
  P = @g_StageFact(0,Stage,Prop),
  @v_StageFact(P) ==1,
  #count { S1: v_StageFact(P1)==1,
    P1 = @g_StageFact(0,S1,Prop)} == 1.
```

The above rule could be read as follows: $Prop$ denotes that O is in stage S if the natural language description of $StageFact(O, Stage, Prop)$ which is obtained by calling the $g\_StageFact$ function is true according to the $v\_StageFact$ function and also the number of stages $S1$ where the natural language description of $StageFact(O, S1, Prop)$ true is equal to 1.

The pair of **generate-validate** function symbols delegates the responsibility of verifying if a proposition is true or not to an external function and I believe that if the proposition is simple enough and close to the texts described in a KB, a simple **validate** function might be able to compute the truth value with good accuracy. However, one important issue with this rule is that it is not "safe". In simple words the above rule does not specify what values the variables $O, Stage, Prop, S1$ can take and as a result what to pass as arguments to the $g\_StageFact$ functions is undefined. To mitigate this issue one needs to add some domain predicates which describes the possible values of the unbounded variables. For our question answering task, I have used the predicates that represent the question as domain predicates. The resulting rule, then, looks as follows:

```
indicator(O,Stage,Prop) :-
  qIndicator(O,Stage),qOption(X,Prop),
  P = @g_StageFact(O,Stage,Prop),
  @v_StageFact(P) ==1,
  #count { S1: ( isAStageOf(S1,O)),
    v_StageFact(P1)==1,
    P1 = @g_StageFact(O,S1,Prop)} == 1.
```

The $isAStageOf(S1, O)$ describes the stages in the life cycle of the organism O and is extracted from the "order" field in life cycle texts ("Order" in Table 22).

7.2.1 On the choices of a Validate Function

The task of deciding if a proposition is true based on a given text is a much studied problem in the field of NLP and is known as *textual entailment*. There exist several textual entailment functions. All of which can be used as validate function. However, the textual entailment functions returns a real value between 0 to 1 denoting the probability that the proposition is true and thus one needs to decide a threshold value to obtain a boolean validate function. In the implementation of our system I have not used a boolean validate function but used the entailment score as it is. I describe how to use a fuzzy validate function in the next section after describing the life cycle dataset and the representation of the texts.

The life cycle question answering dataset contains a total of 41 texts and 5.8k questions. Each text contains a *sequence* which describes the order of stages and a *natural language description* of the life cycle as shown in Table 22. The life cycle texts are collected from the internet. The sequence of the stages are manually added by either looking at the associated image in the website that describes the order of the stages or from the headings of the text (Table 22).

Representing Life Cycle Texts

Each life cycle text is represented in terms of two predicates, namely, *stageAt(URL, O, P, S)* and *description(URL, O, T)*. The *stageAt* predicate describes that according to the source *URL* (from which the text is collected) the stage that comes at position P in the life cycle of O is S. The *description* stores the text that describes the life cycle. The following ASP program shows the representation of the text in Table 22. To save space the actual value of the *URL* is replaced by 'u'. The $\mathcal{KB}$ contains representations of 41 such texts.

```
stageAt(u,"frog",1,"egg").
stageAt(u,"frog",2,"tadpole").
stageAt(u,"frog",3,"tadpole with legs")
stageAt(u,"frog",4,"froglet").
stageAt(u,"frog",5,"adult").
description(u,"frog",
    "Egg: Tiny frog eggs are laid...").
```

7.3.1 Question Categories

The question that are created from these texts are divided into 11 categories. The first three types of questions namely *look up, difference, indicator* require reading the textual description of stages whereas the remaining six types of questions can be answered solely from the sequence of stages (egg → tadpole → tadpole with legs → adult).

Look Up Questions This category contains questions the answer to which can be directly looked up from the description of the stages and does not require any special thinking. The following list shows some questions in this category:

How do froglets breath? (A) using lungs (B) using gills

The tail of a frog disappears at what stage? (A) adult (B) froglet

Where do female frogs lay their eggs? (A) In water (B) On land

Difference Questions This category of questions compare two stages based on their physical attributes, abilities or need that is true in one stage but not in other. The following list shows examples:

What is an adult newt able to do that a tadpole cannot? (A) walk on land (B) swim in water

A tadpole just turned into an eft. What does it need now? (A) shade (B)water

A seedling develops what that a sprout does not have? (A) protective bark (B) root

Indicator Questions This category of questions mentions an organism, a stage, two answer choices and asks which one of those indicates that the organism is in the given stage. Question 3 in Table 22 provides an example of this.

Sequence Based Questions Questions from this category can be answered based on the sequence of stages that describes journey of an organism from beginning to the end (e.g. egg → tadpole → tadpole with legs → adult). Questions in this category are further divided into 8 classes which takes one of the following forms: (1) *Next Stage Questions*: given a stage and an organism, asks for the next stage. (2) *Before Stage Questions*: given a stage and an organism, asks for the stages that appear before. (3) *Between Stage Questions*: given two stages and an organism, asks for the stages that appear between those two. (4) *Stage At Questions*: given an organism and a position, asks for the stages that appear at that position. (5) *Count Questions*: given an organism asks how many stages are there in the life cycle. (6) *Correctly Ordered Questions*: given an organism asks the sequence that describes the correct order of the stages. (7) *Stage Of Questions*: given an organism asks for the stages that appear in its life cycle. (8) *Not a Stage of Questions*: given an organism asks for the stages that do not appear in its life cycle. Table 23 shows an example of each types of questions.

Question Representation

The representation of a question comprises of four ASP facts. Given an MCQ question of the form "⟨Q?⟩ (A) ⟨answer choice 1⟩ (B) ⟨answer choice 2⟩ ", the first three facts are computed trivially as follows:

```
question(``Q?'').
option(a,``answer choice 1'').
```

```
option(b,``answer choice 2'').
```

The fourth fact captures the type of the question (i.e. look up, difference etc.) and some associated attributes (i.e. organism, stages, position). For each one of the 11 types of questions in the dataset there is a fixed template which describes the associated attributes for each type of question. The fourth fact is an instantiation of that template which is computed by a semantic parser. Table 23 describes the questions templates and shows an example instantiation.

7.3.2 Theory

The *theory* contains a total of 36 ASP rules, 3 *generate* functions one for each of the *look up, difference* and *stage indicator* question type and a single *validate* function. The *validate* function, $@validate(Text, Hypothesis)$ takes as input a life cycle *text* and a *hypothesis* (string) and returns a score between 0 to 1. The score is computed using a textual entailment function as follows:

$score = max\{\ textual\_entailment(S, Hypothesis) :$

$S\ is\ a\ sentence\ in\ Text\}$

To find the answer, a confidence score $V \in [0, 1]$ is computed for each answer option X (denoted by $confidence(X, V)$). The rules in the *theory* computes these confidence scores. The correct answer is the option that gets maximum score. The following rule describes this:

```
ans(X):- option(X,V), confidence(X,V),
      V == #max {V1:confidence(X1,V1)}.
```

Here I describe only the non-trivial rules that call entailment functions through

function symbols.

Lookup Questions Given the representation of a lookup question such as: {*qLookup("frog"). question("How do froglets breathe?"). option(a,"using gills"). option(b,"using lungs").*}, the following rule computes the confidence score for each option.

```
confidence(X,V):-
  question(Q), qOption(X,C),
  H = @generate_lookup(Q,C),
  qLookup(Org), description(URL,Org,P),
  V = @validate(P,H),
```

While creating the confidence for option "a" this rule will call the *generate_lookup(Q,C)* function with Q = "How do froglets breathe?" and C = "using gills". The *generate_lookup* function then returns a hypothesis "froglets breathe using gills". The *validate* function then takes the description of the frog life cycle and the hypothesis and verifies if any of the sentence in the text supports the hypothesis: "froglets breathe using gills". The confidence score of option "a" is the score returned by the *validate* function. Similarly it will compute the confidence score for option "b".

The work of Khot, Sabharwal, and Clark 2018 presents a function that creates a hypothesis from a question and an answer choice which was used to solve MCQ questions. The *generate_lookup* function here reuses their implementation.

Difference Questions Given a difference question (e.g. "What is an adult newt able to do that a tadpole cannot?" and an answer choice (e.g. "walk on land") a *generate*

function returns two hypothesis H_1 and H_2. ("adult newt able to walk on land", "a tadpole cannot walk on land"). The fuzzy truth value for each each hypothesis is computed with the *validate* function. The product of which is assigned to be the confidence score of the answer choice. A rule is written in ASP to describe the same.

Indicator Questions When dealing with a fuzzy *validate* function the definition of an *indicator* is modified as follows: Let v be the score for an answer choice c that indicates that the organism O is in stage S. If O goes through n stages, S represents the j-th stage and p_i is the truth value that c is true in stage i, then $v = p_j * \prod_{k=1,k\neq j}^{n}(1-p_k)$. The following five ASP rules are written to describe the same.

```
stageIndicatorIndex(ID):-
  stageAt(URL, 0, ID, S),
  qStageIndicator(0,S).

trueForStage(Idx,X,V):-qIndicator(0,S),
  option(X,C),stageAt(URL,0, Idx, S1),
  H = @generate_indicator(S1,C)
  description(URL, 0, Text),
  V = @validate(Text, H).

result(1, X , @product("1.0",V,1,ID)):-
  trueForStage(0, 1,X,V),
  stageIndicatorIndex(SRC,ID).

result(0, N, X, @product(V1,V2,N,ID)):-
```

```
  result(O, N-1, X , V1),
  trueForStage(O, N,X,V2),
  stageIndicatorIndex(ID).

confidence(X,V):- res(N, X , V),
  N = #max {P:stageAt(URL,O, P, S )}.
```

The first rule finds out the index of the stage specified in the question. The second rule computes the truth value p_i (*trueForStage(Idx, X, V)*) for each stage index *Idx* and each option X. The last three rules compute the confidence score $v = p_j * \prod_{k=1,k\neq j}^{n}(1 - p_k)$ iteratively. Here *product(V1, V2, N, ID)* function returns either $V1 * V2$ or $V1 * (1 - V2)$ depending on whether N is equal to ID. The *generate_indicator* function follows a simple template. It takes as input a stage such as "froglet" and an answer choice, for e.g. "when it has lungs" and returns "In the ⟨froglet⟩ stage, ⟨it has lungs⟩".

7.4 Dataset Creation

I crowdsourced the dataset of 5811 multiple-choice life cycle questions with their logical forms with the help of Amazon Mechanical Turk. The workers did not create the logical forms. I collected them using reverse-engineering without exposing the workers to the underlying formalism.

To obtain the sequence based questions I followed the crowdsourcing technique in Wang, Berant, and Liang 2015. Using *stageAt* predicates in the $\mathcal{KB}$ and the rules in the *theory* I first computed a database of sequence based facts such as $nextStage(frog, egg, tadpole)$. I then used a simple grammar to create an MCQ question out of it, for e.g, "What stage comes after egg stage in frog's life? (A)

tadpole (B) adult". Finally I asked the workers to rephrase these questions as much as possible. Since the seed questions were generated using logical facts I could also compute the logical form and the correct answer beforehand.

To collect indicator type questions I gave the workers a life cycle text and described what is meant by an stage indicator question. Each worker were then asked to create two multiple choice stage indicator questions and write down the correct option and associated stage for each question. There were two workers working on each text. As a result I got $41 \times 2 \times 2 = 164$ questions. I manually removed the questions that did not meet the requirements and finally ended up with 125 questions. Using the stage name that was written down for each question I were able to compute the logical form $qStageIndicator(organism, stage)$. Similarly, a separate task was created to collect *stage difference* questions where the workers apart from the question and the answer choices also wrote down the two stages that are being compared. Using that I computed the logical form.

To obtain look up questions I gave the workers a life cycle text and asked them to create free form MCQ questions, which gave us 2710 questions. I then manually filtered the questions that should belong to the other 10 categories and ended up with 2525 look up questions. Since the question template of a look up question only contains the organism name I did not need any extra supervision to create the logical form.

7.5 Related Work

Many question answering systems Sharma et al. 2015; Arindam Mitra and Baral 2016a, 2015; Wang, Lee, and Kim 2017; Lierler, Inclezan, and Gelfond 2017; Clark, Dalvi,

and Tandon 2018; Moldovan et al. 2003 have been developed that use declarative programming paradigm. Among these the closest to our work are the works of Lierler, Inclezan, and Gelfond 2017; Arindam Mitra and Baral 2016a; Clark, Dalvi, and Tandon 2018 which try to answer a question with respect to a given text. But to do so they convert the associated text into some action language with existing natural language parsers Bos 2008; He et al. 2017; Flanigan et al. 2014. Having a formal representation of the text is helpful but the ability to provide special domain knowledge should not be impaired by the absence of a formal representation of the text. Our work can be considered as a step towards that direction.

Our work is also related to Eiter et al. 2006; Havur et al. 2014. Eiter et al. have used function symbols (referred to as *external atoms*) to interface ASP with an ontology language (e.g. OWL) that has different formats and semantics. In Havur et al. 2014 function symbols are used to delegate some low level feasibility checks (such as "is it possible to move left without colliding") in a robotics application.

The task of textual entailment Dagan, Glickman, and Magnini 2006 and semantic parsing Zelle and Mooney 1996 play a crucial role in our work. With access to new datasets both the task have received significant attention Bowman et al. 2015; Parikh et al. 2016; Chen et al. 2018; Wang, Berant, and Liang 2015; Krishnamurthy, Dasigi, and Gardner 2017.

Finally, recently there has been a surge of new question answering datasets. Depending on their restrictions on the possible answers they can be divided into three categories: (1) the answer is an exact substring of the text (2) the answer can take values from a fixed which is decided by the training dataset and (3) multiple choice questions. I have used the accuracy of existing science MCQ solvers Khot, Sabharwal, and Clark 2018 as baselines in our experiment.

Setup To evaluate our system I divide the 41 texts and the 5811 questions in two different ways:

Text Split : In this case, I follow the machine comprehension style question answering and divide the 41 life cycle texts into three sets. The training set then contains 29 texts and 4k associated questions, the dev set contains 4 texts and 487 questions and the test set contains 8 texts with 1368 questions. Given a text and a MCQ question the task is to find the correct answer choice.

Question Split : In this split I mimic the open book exam setting and divide the 5.8k questions randomly into train, dev and test set each containing 4011, 579 and 1221 questions respectively. Here the knowledge base contains all the texts. Given a MCQ question the task is to find out the correct answer choice with respect to the knowledge base.

Our System I experiment with four different textual entailment functions. One of those is a neural network based model Parikh et al. 2016. The remaining three are variations of n-grams and lexical similarity based model Jijkoun and De Rijke 2006. The first variation (NGram-LS-1) uses WordNet based lexical similarity. The second variation uses (NGram-LS-2) weighted words Jijkoun and De Rijke 2006 along with simple synonym based similarity. The third variation (NGram-LS-3) uses both word weights and WordNet based lexical similarity.

The semantic parser in Krishnamurthy, Dasigi, and Gardner 2017 is trained to obtain the question template instances (e.g. *qIndicator*("$frog$", "*adult*")). I observed that the semantic parser predicts the question types (e.g. *qIndicator*) with high accuracy

but often make errors in identifying the associated attributes (e.g. "adult"). For example it predicts that a given question is of *qStageAt* type with 100% accuracy but fails to identify the associated stage index attribute 38% times. Since the question templates in our dataset is quite simple and only contains one organism name, maximally two stage names or one stage index, I employ a simple search to extract the attributes. The resulting semantic parser then works as follows: it first obtains the question type from the trained parser of Krishnamurthy, Dasigi, and Gardner 2017. Then it calls a function with a list containing all the organism names and the question. The function then returns the specified organism based on the first organism name that appears in the question. Similarly it makes subsequent calls for extracting stage names and positions. From now on I refer to the semantic parser in Krishnamurthy, Dasigi, and Gardner 2017 as "KDG" and the customized version as "Customized-KDG".

Baselines I use the performance of the entailment functions as baseline scores. For each option a hypothesis is created by combining the question and the answer choice using the code from Khot, Sabharwal, and Clark 2018, which is then passed to an entailment function to compute the confidence score. A second set of baseline is computed using BiDaF Seo, Kembhavi, et al. 2016 which performed well across several machine comprehension tasks. Given a passage and a question, BiDaF returns a substring of the passage as an answer. I then use that substring to compute the confidence score for each option. Two versions of BiDaF is used: BiDaF-1 which is trained on Rajpurkar et al. 2016 and BiDaF-2 which is trained on both Rajpurkar et al. 2016; Clark et al. 2018. To make the comparison fair, I have added a sentence of the type "The *i*-th stage is S" for each $stageAt(O, I, S)$ fact in the $\mathcal{KB}$. Also during

the evaluation of "Question Split" only the necessary life cycle text is given as the passage.

System	Acc(%) Question Split	Acc(%) Text Split
Gold + Parikh et al. 2016	73.63	78.87
Gold + NGram-LS-1	78.95	**84.06**
Gold + NGram-LS-2	79.20	83.77
Gold + NGram-LS-3	**79.28**	83.77
KDG + Parikh et al. 2016	70.60	72.51
KDG + NGram-LS-1	73.87	**76.17**
KDG + NGram-LS-3	74.28	75.88
KDG + NGram-LS-3	**74.61**	76.02
Custom-KDG + Parikh et al. 2016	72.40	76.68
Custom-KDG + NGram-LS-1	77.07	**80.70**
Custom-KDG + NGram-LS-2	77.72	80.41
Custom-KDG + NGram-LS-3	**77.80**	80.48
Parikh et al. 2016	53.07	51.02
NGram-LS-1	61.29	61.25
NGram-LS-2	60.44	58.04
NGram-LS-3	**62.40**	**61.98**
BidaF-1	60.03	57.27
BidaF-2	58.44	60.20

Table 24: The first 12 rows show the performance of our method with different parsers and entailment functions. The last 6 rows show the performance of the baseline methods.

Results Table 24 presents the performance of all the systems on both splits. The first four rows show the accuracy of our system when gold representation of the question is used. This shows the best performance that the system can achieve with the entailment functions at hand; which is 79.28% with the NGram-LS-3 entailment function on the "Question Split" and 84.06% with the NGram-LS-2 entailment function on the "Text Split". The next four rows show the performance with the KDG parser. The errors

made by the parser result in an accuracy drop of ∼ 5% on “Question Split“ and a drop of ∼ 8% on “Text Split”. However, when the customized-KDG parser is used the accuracy on both the split increases. The best accuracy on “Text Spit“ is 77.8% which is within 1.5% of the achievable best with the entailments at hand. The accuracy drop on “Text split” also reduces from ∼ 8% to ∼ 3.3%. Among the baseline methods which are shown in the last 6 rows, the best score is achieved by the NGram-LS-3 entailment function which is 15.4% less than the best performance achieved by our system on “Question Split“ and 18.72% less on “Text Split”.

7.7 Conclusion

Developing methods that allow machines to reason with background knowledge with premises written in natural language enhances the applicability of logical reasoning methods and significantly reduces the effort required in building a knowledge based question answering system. In this chapter I describe one such method by using ASP with textual entailment functions. Experiments show the success of this method. However there is still scope for further improvements with the best accuracy being 80.7%. The life cycle dataset and the associated code is publicly available to track the progress towards this direction.

Chapter 8

DECLARATIVE QUESTION ANSWERING OVER KNOWLEDGE BASES CONTAINING NATURAL LANGUAGE TEXT: SOLVING QUALITATIVE WORD PROBLEMS

8.1 Introduction and Motivation

In this chapter, I describe how to apply the TKR paradigm for declarative question answering over Text for solving qualitative word problems. Qualitative relationships describe how increasing or decreasing one property (e.g. altitude) affects another (e.g. temperature). They are an important aspect of natural language question answering and are crucial for building chatbots or voice agents where one may enquire about qualitative relationships. In various natural language question answering domains, applications, and challenge corpora one often encounters textual content and questions about qualitative relationships. For example, a chatbot developer developing a chatbot for a company dealing with windows and curtains would need the chatbot to be able to answer questions such as: "*Will a larger window make the room warmer?*", and "*Will a white curtain in the window make the room cooler?*". Similarly, in the Aristo Clark 2015 corpus there are several items that involve qualitative relationships. An example from that corpus is as follows:

> *In a large forest with many animals, there are only a small number of bears. Which of these most likely limits the population of bears in the forest?*
> *(A) supply of food*
> *(B) type of tree*

(C) predation by carnivores
(D) amount of suitable shelter

Considering the importance of being able to answer questions about qualitative relationships in a question answering setting, recently the QUAREL corpus Tafjord et al. 2018 has been proposed. Table 25 shows some examples from the QUAREL corpus.

I: *A boomerang thrown into a windy sky heats up quite a bit, but one thrown into a calm sky stays about the same temperature. Which surface puts the least amount of friction on the boomerang? (A) windy sky (B) calm sky*
II: *Tank the kitten learned from trial and error that carpet is rougher then skin. When he scratches his claws over carpet it generates _____ then when he scratches his claws over skin (A) more heat (B) less heat*
III: *The propeller on Kate's boat moved slower in the ocean compared to the river. This means the propeller heated up less in the (A) ocean (B) river*
IV: *Juan is injured in a car accident, which necessitates a hospital stay where he is unable to maintain the strength in his arm. Juan notices that his throwing arm feels extremely frail compared to the level of strength it had when he was healthy. If Juan decides to throw a ball with his friend, when will his throw travel less distance? (A) When Juan's arm is healthy (B) When Juan's arm is weak after the hospital stay.*

Table 25: Example problems form the QUAREL corpus

My goal is to develop a method for answering questions about qualitative relationships, especially with respect to the QUAREL dataset. There are several challenges associated with question answering in this domain. First, it requires reasoning with external knowledge about qualitative relations. Although a small knowledge base related to QUAREL has been provided by the QUAREL authors, which I refer to as QRKB (Qualitative Relations Knowledge Base), incorporating that knowledge into the question answering process is a challenge. Second, as pointed out in Tafjord et al. 2018 direct IR based methods, and word association based methods do not do well in this domain. That is because neither of them properly capture reasoning with

external knowledge. A Knowledge Representation and Reasoning (KR&R) based approach, that can use reasoning modules from the qualitative reasoning literature Daniel G Bobrow 2012; Weld and De Kleer 2013 can be employed. For e.g., the problem I from table 25 can be translated to the following tuple: *(qrel(friction, higher, carpet),qrel(heat, higher, carpet),qrel(heat, lower, carpet))*[2]. The first component of the tuple *qrel(friction, higher, carpet)* denotes the given fact i.e. "friction is more on carpet". The second component denotes the claim corresponding to option A i.e. "more heat is generated on carpet" and the third component captures the claim corresponding to option B which is "less heat is generated on carpet". The reasoning module using the qualitative knowledge that more friction results in more heat can then decide that option A is true. However such approach requires accurate semantic parsing of the text and the question and that is a big challenge. Nevertheless, the authors of QUAREL provide annotations that can facilitate a limited semantic parsing and use that to develop a type constrained neural semantic parser (QUASP) which together with delexicalization results in their best performing system (QUASP+).

Our approach aims to address the drawbacks of using a traditional semantic parser for obtaining the logical representation. Existing semantic parsers are trained to translate the natural language sentences into an application specfic logical representation. Before training, the semantic parsers have some prior knowledge of the input (natural) language, which is normally captured by the word vectors, existing knowledge bases such as WordNet, ConceptNet or parse trees. The target language however is a complete unknown. The model must learn the meaning of the symbols in the target language (i.e. the association between the symbols in the target vocabulary to the

[2]This is for illustration purpose.This is not exactly same as the logical form that QUASP or QUASP+ translates to.

ones in input vocabulary) and how to combine these symbols given the input sentence solely from the annotated training data. These expectations naturally increase the demand for more annotated data and these models often suffer if some of the symbols from the output vocabulary do not appear in the training dataset but appear in test set. To address these challenges we apply the TKR framework from the previous chapter which promotes the following idea:

> If a reasoning algorithm requires facts to be given in a logical form and the application developer has natural language texts at hand, then instead of employing a semantic parser to convert the text to suitable logical facts, **generate** a natural language description of the logical fact and **validate** if the text entails the natural language description.

Thus instead of generating the logical form from the input problem as is done in Tafjord et al. 2018, I 'roughly iterate' over the space of possible logical forms, generate a natural language description for each logical form, validate (score) each of those natural language descriptions using multiple "textual entailment" calls and then finally use those scores to detect the correct answer choice. Since, the space of possible logical forms can be quite big, instead of performing a brute-force search I perform an efficient search, which we describe later in section 8.3.

Unlike in the previous chapter where we train to semantic parser to translate the question, here I use an NLI function to do both question understanding and passage understanding and which facilitates transfer like for. e.g. using Natural Language Inference dataset, or pre-trained models. This heavily boost the performance on QUAREL when instead of directly generating the logical form, semantic parsing is done through the generte-validate ideology of TKR. The developed system obtains an accuracy of 76.63% which is 7.93% better than QUASP+ model and 20.53% better than QUASP model.

8.2 Background

8.2.1 The QUAREL Dataset

The QUAREL dataset Tafjord et al. 2018 has 2771 annotated multiple choice story questions. Table 25 shows some sample questions from the QUAREL dataset. Each question in the QUAREL dataset has annotation in the form of logical forms and world literals which we show here for items I and II of Table 25:

Annotation for Problem I:

Logical Form

$qval(heat, high, world1), qval(heat, low, world2) \rightarrow$

$qrel(friction, lower, world1);$

$qrel(friction, lower, world2)$

Literals

$world1\_literal$:"windy sky"

$world2\_literal$: "calm sky"

Annotation for Problem II:

Logical Form

$qrel(smoothness, lower, world1) \rightarrow$

$qrel(heat, higher, world1); qrel(heat, lower, world1)$

Literals

$world1\_literal$: "carpet"

$world2\_literal$: "skin'

The two examples show two types of logical forms. Syntactically, the logical forms have two parts: the *setup part* that describes the set of explicitly given facts and the *answer choice part* that gives two claims, one for option A (here after claimA) and another for option B (here after claimB). The setup part and the answer choice part are separated by the '→' symbol whereas ';' separates the two claims inside the answer choice part.

Both the claims and the given facts are represented by the two predicates, *qrel* and *qval*. In the first example the *setup part* provides two facts: $qval(heat, high, world1), qval(heat, low, world2)$ which should be read as: *heat is high in world1* and *heat is low in world2*. The claimA is $qrel(friction, lower, world1)$ which should be read as *friction is lower in world1* ***compared to the other world*** whereas claimB is $qrel(friction, lower, world2)$ which represents *friction is lower in world2* ***compared to the other world***. Here, *world*1 and *world*2 are two special symbols which refer to "windy sky" and "calm sky" respectively. This information is given through the world literal annotation. Each logical form in QUAREL has at max two worlds however the meaning of the worlds i.e. *world*1_*literal* and *world*2_*literal* changes with each problem. Both the predicate *qrel* and *qval* has three arguments. The first one is a qualitative property, the second one is called *direction* which could be either low or high and the third one is the special variable *world* which also takes two values *world*1 or *world*2. In this work, we treat *qval* and *qrel* uniformly and same natural language description is generated for both of them as there only two worlds and thus the 'absolute' (*qval*) and the 'relative' (*qrel*) descriptions are equivalent.

The QRKB of QUAREL has the following 19 qualitative properties: friction, speed, distance, smoothness, heat, loudness, brightness, apparentSize, time, weight, strength, mass, flexibility, exerciseIntensity, acceleration, thickness, gravity, breakability, and

amountSweat. The QRKB has 25 qualitative relations about pairs of these properties. These relations use the predicates q+ and q-. Some example relations are: q-(friction, speed), and q+(friction, heat). Intuitively, q-(X,Y) means that the amount of X is inversely proportional to the amount of Y and q+(X,Y) means that the amount of X is proportional to the amount of Y. Every possible relation pairs are precomputed and stored in QRKB.

8.2.2 Textual Entailment and NLI

As briefly mentioned in Section 8.1 our approach uses Textual Entailment Dagan et al. 2013 and Natural Language Inference Bowman et al. 2015 models. Natural language inference (NLI) is the task of determining the truth value of a natural language text, called *hypothesis* given another piece of text called *premise*. The list of possible truth values include *entailment*, *contradiction* and *neutral*. *Entailment* means the hypothesis must be true if the premise is true. *Contradiction* indicates that the hypothesis can never be true if the premise is true. *Neutral* pertains to the scenario where the hypothesis can be both true and false as the premise does not provide enough information. Textual Entailment is a binary version of NLI task, where one has to decide if the truth value is *entailment* or not. Table 26 shows some examples. Recently, several large scale NLI dataset has been developed. One of which is SNLI Bowman et al. 2015 which we use in this work. Any NLI dataset can be converted to a textual entailment dataset by replacing the *contradiction* and *neutral* label with *not-entailment* label. Among the recent NLI models, the two most popular models are BERT Devlin et al. 2018 and ESIM Chen et al. 2016 which we use in our implementation.

premise: Tank the kitten learned from trial and error that carpet is rougher then skin. **hypothesis:** Carpet is less smooth. **label:** entailment.
premise: Tank the kitten learned from trial and error that carpet is rougher then skin. **hypothesis:** skin is less smooth. **label:** not-entailment.

Table 26: Example premise-hypothesis pairs with annotated labels.

8.3 Proposed approach

A qualitative problem P in QUAREL is a sequence of k sentences followed by two option choices. Let T denote the sequence of k sentences and A_1 and A_2 be the two answer choices. The last sentence in T is a question and is denoted by Q. For e.g., for the problem 1 in Table, T = *A boomerang thrown into a windy sky heats up quite a bit, but one thrown into a calm sky stays about the same temperature. Which surface puts the least amount of friction on the boomerang?*, A_1 = *windy sky*, A_2 = *calm sky* and Q = *Which surface puts the least amount of friction on the boomerang?* Given such a problem $P = (T, Q, A_1, A_2)$, the task is to decide if A_1 is a better answer choice or A_2. Our algorithm, namely generate validate qualitative problem solver (gvQPS), has three key steps, namely *generate, validate* and *inference*, which are discussed in this section.

8.3.1 Step 1: Generate

Given T, Q, A_1, and A_2 a set $H(T, Q, A_1, A_2)$ of 46×n hypothesis such as "windy sky has more friction" is created using templates such as "X has more friction". Our algorithm uses a total of 46 manually authored templates. Each template has only one

variable X which is substituted by the n noun phrases in the T, Q, A_1 and A_2 parts to create the set $H(T, Q, A_1, A_2)$.

Table 28 shows the templates. Each template pertains to a $qrel(P, D, X)$ predicate where P is a qualitative property from QUAREL, $D \in \{low, high\}$, X is a variable representing the textual description of the world. All the properties except *speed* and *distance* have two templates, one for $D = low$ and another for $D = high$. The two properties *speed* and *distance* however have more than two templates to capture different senses.

For the example 2 from Table 25, there are a total of 10 noun-phrases[3], namely "heat"■, "trial and error", "claws", "kitten", "carpet", "skin", "tank kitten", "error", "tank", "trial". Thus the set $H(T, Q, A_1, A_2)$ contains a total of 460 ($= 46 \times 10$) hypothesis. Among these the ones related to friction and high are as follows: *heat has more friction, trial and error has more friction, kitten has more friction, claws has more friction, carpet has more friction, skin has more friction, tank kitten has more friction, error has more friction, tank has more friction, trail has more friction*.

8.3.2 Step 2: Validate

Recall that the logical form has three parts: the given facts, the claimA and the claimB all of which are represented by the *qrel* or *qval* predicate. In step 1 the system has generated the set of natural language descriptions of all possible grounded *qval* predicates, some of which are the given facts, the claimA or claimB. The goal of step 2 is to precisely identify which statement from $H(T, Q, A_1, A_2)$ is claimA, which statement pertains to claimB and which statements represents the given facts. To do

[3]according to Spacy constituency parser

this, the system scores the statements in $H(T, Q, A_1, A_2)$ using two different Textual Entailment functions. Let $given_{score}(.)$, $claimA_{score}(.)$ and $claimB_{score}(.)$ respectively denote the score for a hypothesis to be a given fact, the claimA and the claimB. These scores are then computed as follows:

$given_{score}(H_i, T, Q, A_1, A_2) = f_{TE}^{given}(T, H_i)$

$$claimA_{score}(H_i, T, Q, A_1, A_2) = f_{TE}^{claim}(QA_1, H_i)$$

$$claimB_{score}(H_i, T, Q, A_1, A_2) = f_{TE}^{claim}(QA_2, H_i)$$

Here, QA_1 and QA_2 respectively denotes the concatenation of Q,"(option)", A_1 and Q,"(option)", A_2 and f_{TE}^{given} and f_{TE}^{claim} are the two different Textual Entailment functions. f_{TE}^{given} and f_{TE}^{claim} might have same architecture but they are trained on different datasets and take different inputs. For the example II from Table 25 which has a logical representation of $(smoothness, lower, world1) \rightarrow (heat, higher, world1); (heat, lower, world1)$, I expect the textual entailment functions to produce the following scores for the sample inputs of table 27.

8.3.3 Step 3: Answer Generation

In this step, the system computes the final answer by using the scores that are computed in step 2. Let $claimA^*$ and $claimB^*$ be the hypothesis in $H(T, Q, A_1, A_2)$ which has respectively the highest $claimA_{score}(.)$ and the highest $claimB_{score}(.)$ score. The answer is option A if $given_{score}(claimA^*)$ is more than $given_{score}(claimB^*)$, otherwise the answer is option B. Here, I assume that the $given_{score}$ will learn to capture the qualitative relationship. For e.g., if it assigns a high score to the hypothesis *skin has*

$given_{score}$(“*Carpet is less smooth.*”)	1
$given_{score}$(“*Skin is less smooth.*”)	0
$given_{score}$(“*Carpet is more smooth.*”)	0
$claimA_{score}$(“*Carpet is less smooth.*”)	0
$claimA_{score}$(“*more heat is generated on carpet*”)	1
$claimA_{score}$(“*less heat is generated on carpet*”)	0
$claimB_{score}$(“*more heat is generated on carpet*”)	0
$claimB_{score}$(“*less heat is generated on carpet*”)	1
$claimB_{score}$(“*less heat is generated on skin*”)	0

Table 27: Example of expected scores and sample inputs.The arguments T, Q, A_1 and A_2 take the following value: T = *Tank the kitten learned from trial and error that carpet is rougher then skin. When he scratches his claws over carpet it generates _____ then when he scratches his claws over skin*, Q = *When he scratches his claws over carpet it generates _____ then when he scratches his claws over skin*, A_1 = *more heat,* A_2 = *less heat*.

less friction, it will also assign high score to the hypothesis *less heat is generated on skin*.

8.4 Textual Entailment Dataset Generation

Our algorithm uses two textual entailment functions namely, f_{TE}^{given} and f_{TE}^{claim} both of which needs to be trained. In this section I describe the process that generates labeled premise-hypothesis pairs from the QUAREL annotations.

8.4.1 Dataset for f_{TE}^{claim}

Let $qrel(P^A, D^A, W^A)$ or $qval(P^A, D^A, W^A)$ be the claimA and $qrel(P^B, D^B, W^B)$ or $qval(P^B, D^B, W^B)$ be claimB as per the associated logical form. I use this information to create following annotated premise-hypothesis pairs (I use 1 to denote *entailment* and 0 to denote *not-entailment*):

(Property, Direction)	**Template(s)**
(Friction, high)	X has more friction
(Friction, low)	X has less friction
(Smoothness, high)	X is more smooth
(Smoothness, low)	X is less smooth
(Heat, high)	more heat is generated on X
(Heat, low)	small amount of heat is generated on X
(Loudness, high)	X sounds louder
(Loudness, low)	X sounds softer
(Brightness, high)	X shines more
(Brightness, low)	X looks dim
(apparentSize, high)	X appears big
(apparentSize, low)	X appears small
(Speed, high)	X is fast
	moves fast through X
(Speed, low)	X is slow
	moves slowly through X
(time, high)	X takes more time
(time, low)	X takes less time
(weight, high)	X has more weight
(weight, low)	X has less weight
(acceleration, high)	acceleration is more for X
(acceleration, low)	acceleration is less for X
(strength, high)	X has more strength
(strength, low)	X has little strength
(distance, high)	travelled more on X
	X is far
	X travelled more
	X threw the object far
(distance, low)	travelled less on X
	X is near
	X travelled less
	X could not throw the object far
(thickness, high)	X is thicker
(thickness, low)	X is thin
(mass, high)	X has more mass
(mass, low)	X has less mass
(gravity, high)	X has stronger gravity
(gravity, low)	X has weaker gravity
(flexibility, high)	X is more flexible
(flexibility, low)	X is less flexible
(breakability, high)	X is more likely to break
(breakability, low)	X is less likely to break
(amountSweat, high)	X is exercising more
(amountSweat, low)	X is almost idle
(exerciseIntensity, high)	X is sweating more
(exerciseIntensity, low)	X is sweating less

Table 28: Associated templates for each qualitative property.

1. premise = QA_1, hypothesis = generate(P^A, D^A, W^A) and label = 1
2. premise = QA_2, hypothesis = generate(P^B, D^B, W^B) and label = 1
3. premise = QA_1, hypothesis = generate($P^A, opposite(D^A), W^A$) and label = 0
4. premise = QA_2, hypothesis = generate($P^B, opposite(D^B), W^B$) and label = 0
5. If $W_A \neq W_B$, premise = QA_1, hypothesis = generate(P^A, D^A, W^B) and label = 0
6. If $W_A \neq W_B$, premise = QA_2, hypothesis = generate(P^B, D^B, W^A) and label = 0
7. premise = QA_1, hypothesis = generate(P, D, W^A) and label = 0 where $P \in QRKB$ and $P \notin \{P^A, P^B\}$, $D \in \{low, high\}$
8. premise = QA_2, hypothesis = generate(P, D, W^B) and label = 0 where $P \in QRKB$ and $P \notin \{P^A, P^B\}$, $D \in \{low, high\}$
9. premise = QA_1, hypothesis = generate(P^A, D^A, W) and label = 0 where $W \in bad$
10. premise = QA_2, hypothesis = generate(P^B, D^B, W) and label = 0 where $W \in bad$

Here, *generate*(.) denotes the string that is created for the given input of the type (*qualitative property, direction, world_literal*) using the templates in table 28; $opposite(D)$ returns the only member of the set $\{high, low\} \setminus D$ and *bad* is set of noun phrases from the problem P which does not have any word overlap with either world1_literal or world2_literal. For the problem II in table 25, world1_literal = "carpet" and world1_literal = "skin" and the noun phrases are = "heat"■, "trial and error", "claws", "kitten", "carpet", "skin", "tank kitten", "error", "tank", "trial". Thus the *bad* set contain the following elements: "heat"■, "trial and error", "claws", "kitten", "tank kitten", "error", "tank", "trial".

8.4.2 Dataset for f_{TE}^{given}

Similar to f_{TE}^{claim}, I create the following annotated premise-hypothesis pairs for each given fact (P^G, D^G, W^G):

1. premise = T, hypothesis = generate(P^G, D^G, W^G) and label = 1
2. premise = T, hypothesis = generate($P^G, opposite(D^G), W^G$) and label = 0
3. premise = T, hypothesis = generate($P^G, D^G, \{world1\_literal, world2\_literal\} \setminus W^G$) and label = 0
4. premise = T, hypothesis = generate(P^G, D^G, W) and label = 0, for all $W \in bad$
5. premise = T, hypothesis = generate(P, D, W) and label = 0, for all property P where none of q+(P, P^A), q-(P, P^A),q+(P, P^B), q-(P, P^B) is in QRKB, D is either *high* or *low*, $W \in \{world1\_literal, world2\_literal\}$.

However, unlike f_{TE}^{claim}, I also create the following annotated premise-hypothesis pairs for each given fact (P^G, D^G, W^G) using QRKB:

1. premise = T, hypothesis = generate(P, D^G, W^G) and label = 1, for all property P such that q+(P, P^G) *in* QRKB.
2. premise = T, hypothesis = generate($P, opposite(D^G), W^G$) and label = 1, for all property P such that q-(P, P^G) *in* QRKB.
3. premise = T, hypothesis = generate(P, D^G, W^G) and label = 0, for all property P such that q-(P, P^G) *in* QRKB.
4. premise = T, hypothesis = generate($P, opposite(D^G), W^G$) and label = 0, for all property P such that q+(P, P^G) *in* QRKB.

Let $Train_{Given}^{QUAREL}$, Dev_{Given}^{QUAREL} and $Test_{Given}^{QUAREL}$ respectively denote the dataset that are created for f_{TE}^{given} from train, dev and test split of the QUAREL dataset. Similarly, let

$Train^{QUAREL}_{Claim}$, Dev^{QUAREL}_{Claim} and $Test^{QUAREL}_{Claim}$ denote the dataset that are created for f^{claim}_{TE} from train, dev and test split of the QUAREL dataset. $Train^{QUAREL}_{Given}$, Dev^{QUAREL}_{Given} and $Test^{QUAREL}_{Given}$ respectively contains $3,58,647$, $50,874$ and $98,057$ premise-hypothesis pairs. On the other hand, $Train^{QUAREL}_{Claim}$, Dev^{QUAREL}_{Claim} and $Test^{QUAREL}_{Claim}$ respectively contains $3,06,545$, $43,914$ and $87,236$ premise-hypothesis pairs. Note that, to make the dataset balanced, the pairs with label 1 are oversampled. I also use the two-class version of the SNLI dataset to further increase the dataset size.

8.5 Related Work

Our work is related to both the works in semantic parsing Zelle and Mooney 1996; Kwiatkowski et al. 2011; Berant et al. 2013; Krishnamurthy, Dasigi, and Gardner 2017; Reddy, Lapata, and Steedman 2014 and question answering using semantic parsing Lev et al. 2004; Berant et al. 2014; Mitra et al. 2019a.

The problem of QUAREL is quite similar to the word math problems Hosseini et al. 2014; Kushman et al. 2014 in the sense that both are story problems and use semantic parsing to translate the input problem to a suitable representation.

Our work is also related to the work in Mitra et al. 2019a that uses generate-validate framework to answer questions w.r.t life cycle text. Mitra et al. 2019a uses generate-validate framework to verify "given facts". Particularly, it shows how rules can be used to infer new information over raw text without using a semantic parser to create a structured knowledge base. The work in Mitra et al. 2019a uses a semantic parser to translate the question into one of the predefined forms. In our work, however I use generate-validate for both question and "given fact" understanding.

The work of Tafjord et al. 2018 is most related to us. Tafjord et al. 2018 proposes

two models for QUAREL. One uses a state-of-the-art semantic parser Krishnamurthy, Dasigi, and Gardner 2017 to convert the input problem to the desired logical representation. They call this model QUASP, which obtains an accuracy of 56.1%. The other model, called QUASP+ uses a delexicalization step before giving the input to the semantic parser. The delexicalization step identifies the value(s) of world1_literal and word2_literal and then replaces all the occurrences of those strings in the text by the symbol "world1" and "world2". The modified input is then passed to the semantic parser. The delexicalization helps the semantic parser by giving explicit pointers to world1 and world2, which results in an accuracy of 68.7%. Our model does not use such preprocessing and still performs significantly better than QUASP+ model.

8.6 Experimental Evaluation

I use the notation f_D^M to denote that the textual entailment model in use is M which can be either ESIM or BERT and the model M is trained on the dataset D which can be any of following: $Train_{Given}^{QUAREL}$, $Train_{Given}^{QUAREL} \cup Train^{SNLI}$, $Train_{Claim}^{QUAREL}$, $Train_{Claim}^{QUAREL} \cup Train^{SNLI}$. Correspondingly there are a total of 4 possible values for f_{TE}^{given} namely $f^{ESIM}_{Train^{QUAREL}_{Fact}}$, $f^{BERT}_{Train^{QUAREL}_{Given}}$, $f^{ESIM}_{Train^{QUAREL}_{Given} \cup Train^{SNLI}}$ and $f^{BERT}_{Train^{QUAREL}_{Fact} \cup Train^{SNLI}}$. Similarly, there are a total of 4 possible values for f_{TE}^{claim} namely $f^{ESIM}_{Train^{QUAREL}_{claim}}$, $f^{BERT}_{Train^{QUAREL}_{claim}}$, $f^{ESIM}_{Train^{QUAREL}_{claim} \cup Train^{SNLI}}$ and $f^{BERT}_{Train^{QUAREL}_{Fact} \cup Train^{SNLI}}$. Table 29 shows the results of our algorithm for all these 4×4= 16 combinations.

- The best performance is achieved when, $f^{BERT}_{Train^{QUAREL}_{Fact} \cup Train^{SNLI}}$ is used as f_{TE}^{given} and $f^{ESIM}_{Train^{QUAREL}_{claim}}$ is used as f_{TE}^{claim}. I refer to this as gvQPS$^{\mathrm{B^+E}}$. The performance of this

f_{TE}^{given}	f_{TE}^{claim}	Dev(%)	Test(%)
$f_{G_1}^{ESIM}$	$f_{C_1}^{ESIM}$	67.27	71.2
$f_{G_1}^{ESIM}$	$f_{C_1}^{BERT}$	62.23	69.12
$f_{G_1}^{ESIM}$	$f_{C_2}^{ESIM}$	66.54	69.57
$f_{G_1}^{ESIM}$	$f_{C_2}^{BERT}$	59.71	67.39
$f_{G_1}^{BERT}$	$f_{C_1}^{ESIM}$	67.99	71.56
$f_{G_1}^{BERT}$	$f_{C_1}^{BERT}$	67.62	69.38
$f_{G_1}^{BERT}$	$f_{C_2}^{ESIM}$	62.95	69.2
$f_{G_1}^{BERT}$	$f_{C_2}^{BERT}$	68.35	67.93
$f_{G_2}^{ESIM}$	$f_{C_1}^{ESIM}$	68.34	67.21
$f_{G_2}^{ESIM}$	$f_{C_1}^{BERT}$	59.35	66.49
$f_{G_2}^{ESIM}$	$f_{C_2}^{ESIM}$	66.55	66.3
$f_{G_2}^{ESIM}$	$f_{C_2}^{BERT}$	58.63	64.3
$f_{G_2}^{BERT}$	$f_{C_1}^{ESIM}$	**73.38**	**76.63**
$f_{G_2}^{BERT}$	$f_{C_1}^{BERT}$	72.66	75.36
$f_{G_2}^{BERT}$	$f_{C_2}^{ESIM}$	70.50	73.55
$f_{G_2}^{BERT}$	$f_{C_2}^{BERT}$	73.02	70.29

Table 29: shows the accuracy on dev and test set of QUAREL for various choice of f_{TE}^{given} and f_{TE}^{claim}. Here, G_1, G_2, C_1 and C_2 respectively represents $Train_{Given}^{QUAREL}$, $Train_{Given}^{QUAREL} \cup Train^{SNLI}$, $Train_{Claim}^{QUAREL}$, $Train_{Claim}^{QUAREL} \cup Train^{SNLI}$.

combination is 5.07% more than the combination of $f_{Train_{Fact}^{ESIM}}^{ESIM}$ and $f_{Train_{claim}^{QUAREL}}^{ESIM}$ which shows the boost offered by BERT and SNLI.

- The accuracy normally drops when SNLI dataset is used in the training for the f_{TE}^{claim} function irrespective of the model on both dev and test set. I speculate that this happens because the premise in SNLI contain proper sentences whereas the premise in the $Train_{claim}^{QUAREL}$ are options appended to questions and thus have different distributions.
- ESIM models perform consistently better as $Train_{claim}^{QUAREL}$ than BERT models irrespective of the training dataset on both dev and test set.

Table 30 compares our best performing method with other approaches. As shown, in table 30 our model provides an improvement of 7.93% over the previous state-of-the-art QUASP+.

Model	Accuracy(%)
IR	48.6
PMI	50.5
QUASP	56.1
QUASP+	68.7
gvQPS$^{\mathbf{B^+E}}$	76.63

Table 30: Comparing our best performing model with existing solvers of QUAREL.

8.6.1 Error Analysis

Our best model, gvQPS$^{\mathrm{B^+E}}$ fails to solve 129 problems. The majority of the error occurs due to the error in $given_{score}(.)$. The following figure shows two examples of error with $claimA^*$ and $claimB^*$ and the scores of the relevant hypothesis by $given_{score}(.)$.

Error Example I:

Nell has very thick hair; Lynn's hair is much thinner. Whose hair is stronger? (A) Nell (B) Lynn

$claimA^*$: ($strength, high$, 'Nell')

$claimB^*$: ($strength, high$, 'Lynn's hair')

Sample $given_{score}(.)$ **scores**

lynn 's hair has more strength, 0.01

nell has more strength, 0.00003

Error Example II:

David noticed that it was harder to push his snow blower on snowy pavement than on dry pavement. This is because the dry pavement has (A) more friction or (B) less friction

$claimA^*$: ($friction, high$, 'dry pavement')

$claimB^*$: ($strength, low$, 'dry pavement')

Sample $given_{score}(.)$ **scores**

dry pavement has more friction, 0.9645242997992661

dry pavement has less friction, 0.000003

As seen in the above figure, for both the error examples, the $claimA^*$ and $claimB^*$ have been identified correctly, however the $given_{score}(.)$ predicts wrongly which results in an error.

8.7 Conclusion

In several situations one need to reason with background knowledge while answering questions. The TKR paradigm is one approach to deal with such scenarios. Apart from allowing declarative programming over text, it provides a sweet point for transfer learning where you can utilize existing datasets or models for Natural Language

Inference and allows on to work with limited numbers labelled data as is true for both the qualitative word problems and life cycle questions.

Chapter 9

NATURAL LANGUAGE INFERENCE FOR OPEN-BOOK QUESTION ANSWERING: EXPERIMENTS AND OBSERVATIONS

9.1 Introduction

Till now I have presented my solutions that uses knowledge representation and reasoning (KR) and learning for two types of reading comprehension problems. Even though the solutions that I have proposed have widens the applicability of KR frameworks to a larger extent the requirement of having a natural language parser or the set of additional knowledge as logic programs limits its applicability by a significant margin when it is compared to that of the deep neural networks. It is part of my future work, to bridge this gap. Towards that goal, I have selected 4 QA tasks which requires access to additional knowledge which is missing and where existing parser perform poorly and have started by analysing how DL solutions work for these datasets and what are their drawbacks in terms of using knowledge. Here, I present our analysis on one of the task, called open-book question answering.

In recent years, many NLQA datasets and challenges have been proposed, for example, SQuAD Rajpurkar et al. 2016, TriviaQA Joshi et al. 2017 and MultiRC Khashabi et al. 2018, and each of them have their own focus, sometimes by design and other times by virtue of their development methodology. Many of these datasets and challenges try to mimic human question answering settings. One such setting is open book question answering where humans are asked to answer questions in a setup where they can refer to books and other materials related to their questions. In such a setting,

the focus is not on memorization but, as mentioned in OpenBookQA2018, on "*deeper understanding of the materials and its application to new situations Jenkins 1995; Landsberger 1996.*" In OpenBookQA2018, they propose the OpenBookQA dataset mimicking this setting.

Question: *A tool used to identify the percent chance of a trait being passed down has how many squares ?* (A) Two squares (B) **Four squares** (C) Six squares (D) Eight squares
Extracted from OpenBook: a punnett square is used to identify the percent chance of a trait being passed down from a parent to its offspring.
Retrieved Missing Knowledge: Two squares is four. The Punnett square is made up of 4 squares and 2 of them are blue and 2 of them are brown, this means you have a 50% chance of having blue or brown eyes.

Table 31: An example of distracting retrieved knowledge

The OpenBookQA dataset has a collection of questions and four answer choices for each question. The dataset comes with 1326 facts representing an open book. It is expected that answering each question requires at least one of these facts. In addition it requires common knowledge. To obtain relevant common knowledge we use an IR system Clark et al. 2016 front end to a set of knowledge rich sentences. Compared to reading comprehension based QA (RCQA) setup where the answers to a question is usually found in the given small paragraph, in the OpenBookQA setup the open book part is much larger (than a small paragraph) and is not complete as additional common knowledge may be required. This leads to multiple challenges. First, finding

the relevant facts in an open book (which is much bigger than the small paragraphs in the RCQA setting) is a challenge. Then, finding the relevant common knowledge using the IR front end is an even bigger challenge, especially since standard IR approaches can be misled by distractions. For example, Table 31 shows a sample question from the OpenBookQA dataset. We can see the retrieved missing knowledge contains words which overlap with both answer options A and B. Introduction of such knowledge sentences increases confusion for the question answering model. Finally, reasoning involving both facts from open book, and common knowledge leads to multi-hop reasoning with respect to natural language text, which is also a challenge. We try to address the first two challenges in this chapter: (a) We improve on knowledge extraction from the OpenBook present in the dataset. We use semantic textual similarity models that are trained with different datasets for this task; (b) We propose natural language abduction to generate queries for retrieving missing knowledge; (c) We show how to use Information Gain based Re-ranking to reduce distractions and remove redundant information; (d) We provide an analysis of the dataset and the limitations of BERT Large model for such a question answering task.

The current best model on the leaderboard of OpenBookQA is the BERT Large model Devlin et al. 2018. It has an accuracy of 60.4% and does not use external knowledge. Our knowledge selection and retrieval techniques achieves an accuracy of 72%, with a margin of 11.6% on the current state of the art. We study how the accuracy of the BERT Large model varies with varying number of knowledge facts extracted from the OpenBook and through IR.

9.2 Related Work

In recent years, several datasets have been proposed for natural language question answering Rajpurkar et al. 2016; Joshi et al. 2017; Khashabi et al. 2018; Richardson, Burges, and Renshaw 2013; Lai et al. 2017; Reddy, Chen, and Manning 2018; Choi et al. 2018; Tafjord et al. 2018; Mitra et al. 2019b and many attempts have been made to solve these challenges Devlin et al. 2018; Vaswani et al. 2017; Seo, Kembhavi, et al. 2016.

Among these, the closest to our work is the work in Devlin et al. 2018 which perform QA using fine tuned language model and the works of Sun et al. 2018; Zhang et al. 2018 which performs QA using external knowledge.

Related to our work for extracting missing knowledge are the works of Ni et al. 2018; Musa et al. 2018; Khashabi et al. 2017 which respectively generate a query either by extracting key terms from a question and an answer option or by classifying key terms or by Seq2Seq models to generate key terms. In comparison, we generate queries using the question, an answer option and an extracted fact using natural language abduction.

The task of natural language abduction for natural language understanding has been studied for a long time Norvig 1983, 1987; Hobbs 2004; Hobbs et al. 1993; Wilensky 1983; Wilensky et al. 2000; Charniak and R. Goldman 1988; Charniak and R. P. Goldman 1989. However, such works transform the natural language text to a logical form and then use formal reasoning to perform the abduction. On the contrary, our system performs abduction over natural language text without translating the texts to a logical form.

9.3 Approach

Our approach involves six main modules: *Hypothesis Generation*, *OpenBook Knowledge Extraction*, *Abductive Information Retrieval*, *Information Gain based Re-ranking*, *Passage Selection* and *Question Answering*. A key aspect of our approach is to accurately hunt the needed knowledge facts from the OpenBook knowledge corpus and hunt missing common knowledge using IR. We explain our approach in the example given in Table 32.

Question: *A red-tailed hawk is searching for prey. It is most likely to swoop down on what?* (A) **a gecko**
Generated Hypothesis : H : A red-tailed hawk is searching for prey. It is most likely to swoop down on a gecko.
Retrieved Fact from OpenBook: F : hawks eat lizards
Abduced Query to find missing knowledge: K : gecko is lizard
Retrieved Missing Knowledge using IR: K : Every gecko is a lizard.

Table 32: Our approach with an example for the correct option

In *Hypothesis Generation*, our system generates a hypothesis $\mathbf{H_{ij}}$ for the *i*th question and *j*th answer option, where $j \in \{1, 2, 3, 4\}$. In *OpenBook Knowledge Extraction*, our system retrieves appropriate knowledge $\mathbf{F_{ij}}$ for a given hypothesis $\mathbf{H_{ij}}$ using semantic textual similarity, from the OpenBook knowledge corpus **F**. In *Abductive Information Retrieval*, our system abduces missing knowledge from $\mathbf{H_{ij}}$ and $\mathbf{F_{ij}}$. The system formulates queries to perform IR to retrieve missing knowledge $\mathbf{K_{ij}}$. With

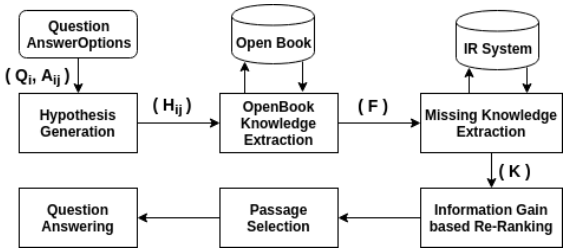

Figure 20: Our approach

the retrieved $\mathbf{K_{ij}}$, $\mathbf{F_{ij}}$, *Information Gain based Re-ranking* and *Passage Selection* our system creates a knowledge passage $\mathbf{P_{ij}}$. In *Question Answering*, our system uses $\mathbf{P_{ij}}$ to answer the questions using a BERT Large based MCQ model, similar to its use in solving SWAG Zellers et al. 2018.

9.3.1 Hypothesis Generation

Our system creates a hypothesis for each of the questions and candidate answer options as part of the data preparation phase as shown in the example in Table 32. The questions in the OpenBookQA dataset are either with *wh* word or are incomplete statements. To create hypothesis statements for questions with *wh* words, we use the rule-based model of demszky2018transforming. For the rest of the questions, we concatenate the questions with each of the answers to produce the four hypotheses. This has been done for all the training, test and validation sets.

9.3.2 OpenBook Knowledge Extraction

To retrieve a small set of relevant knowledge facts from the knowledge corpus **F**, a textual similarity model is trained in a supervised fashion on two different datasets

and the results are compared. We use the *large-cased* BERT Devlin et al. 2018 (BERT Large) as the textual similarity model.

9.3.2.1 BERT Model Trained on STS-B

We train it on the semantic textual similarity (STS-B) data from the GLUE dataset A. Wang et al. 2018. The trained model is then used to retrieve the top ten knowledge facts from corpus **F** based on the STS-B scores. The STS-B scores range from 0 to 5.0, with 0 being least similar.

9.3.2.2 BERT Model Trained on OpenBookQA

We generate the dataset using the gold OpenBookQA facts from **F** for the train and validation set provided. To prepare the train set, we first find the similarity of the OpenBook **F** facts with respect to each other using the BERT model trained on STS-B dataset. We assign a score 5.0 for the gold $\hat{\mathbf{F}}_\mathbf{i}$ fact for a hypothesis. We then sample different facts from the OpenBook and assign the STS-B similarity scores between the sampled fact and the gold fact $\hat{\mathbf{F}}_\mathbf{i}$ as the target score for that fact $\mathbf{F_{ij}}$ and $\mathbf{H_{ij}}$. For example:

Hypothesis : Frilled sharks and angler fish live far beneath the surface of the ocean, which is why they are known as Deep sea animals.

Gold Fact : deep sea animals live deep in the ocean : Score : 5.0

Sampled Facts :

coral lives in the ocean : Score : 3.4

a fish lives in water : Score : 2.8

We do this to ensure a balanced target score is present for each hypothesis and fact. We use this trained model to retrieve top ten relevant facts for each $\mathbf{H_{ij}}$ from the knowledge corpus **F**.

9.3.3 Natural Language Abduction and IR

To search for the missing knowledge, we need to know what we are missing. We use "*abduction*" to figure that out. Abduction is a long studied task in AI, where normally, both the observation (hypothesis) and the domain knowledge (known fact) is represented in a formal language from which a logical solver abduces possible explanations (missing knowledge). However, in our case, both the observation and the domain knowledge are given as natural language sentences from which we want to find out a possible missing knowledge, which we will then hunt using IR. For example, one of the hypothesis $\mathbf{H_{ij}}$ is "*A red-tailed hawk is searching for prey. It is most likely to swoop down on a gecko.*", and for which the known fact $\mathbf{F_{ij}}$ is "*hawks eats lizards*". From this we expect the output of the natural language abduction system to be $\mathbf{K_{ij}}$ or "*gecko is a lizard*". We will refer to this as "*natural language abduction*".

For natural language abduction, we propose three models, compare them against a baseline model and evaluate each on a downstream question answering task. All the models ignore stop words except the Seq2Seq model. We describe the three models and a baseline model in the subsequent subsections.

9.3.3.1 Word Symmetric Difference Model

We design a simple heuristic based model defined as below:

$$K_{ij} = (H_{ij} \cup F_{ij}) \setminus (H_{ij} \cap F_{ij}) \quad \forall j \in \{1, 2, 3, 4\}$$

where i is the ith question, j is the jth option, H_{ij}, F_{ij}, K_{ij} represents set of unique words of each instance of hypothesis, facts retrieved from knowledge corpus **F** and abduced missing knowledge of validation and test data respectively.

9.3.3.2 Supervised Bag of Words Model

In the Supervised Bag of Words model, we select words which satisfy the following condition:

$$P(w_n \in K_{ij}) > \theta$$

where $w_n \in \{H_{ij} \cup F_{ij}\}$. To elaborate, we learn the probability of a given word w_n from the set of words in $H_{ij} \cup F_{ij}$ belonging to the abduced missing knowledge K_{ij}. We select those words which are above the threshold θ.

To learn this probability, we create a training and validation dataset where the words similar (cosine similarity using spaCy) Honnibal and Montani 2017 to the words in the gold missing knowledge $\hat{K}_i$ (provided in the dataset) are labelled as positive class and all the other words not present in $\hat{K}_i$ but in $H_{ij} \cup F_{ij}$ are labelled as negative class.

Both classes are ensured to be balanced. Finally, we train a binary classifier using BERT Large with one additional feed forward network for classification. We define value for the threshold θ using the accuracy of the classifier on validation set. 0.4 was selected as the threshold.

9.3.3.3 Copynet Seq2Seq Model

In the final approach, we used the copynet sequence to sequence model Gu et al. 2016 to generate, instead of predict, the missing knowledge given, the hypothesis **H** and knowledge fact from the corpus **F**. The intuition behind using copynet model is to make use of the copy mechanism to generate essential yet *precise* (minimizing distractors) information which can help in answering the question. We generate the training and validation dataset using the gold $\hat{\mathbf{K}}_\mathbf{i}$ as the target sentence, but we replace out-of-vocabulary words from the target with words similar (cosine similarity using spaCy) Honnibal and Montani 2017 to the words present in $H_{ij} \cup F_{ij}$. Here, however, we did not remove the stopwords. We choose one, out of multiple generated knowledge based on our model which provided maximum *overlap_score*, given by

$$overlap\_score = \frac{\sum_i count((\hat{H}_i \cup F_i) \cap K_i)}{\sum_i count(\hat{K}_i)}$$

where i is the ith question, $\hat{H}_i$ being the set of unique words of correct hypothesis, F_i being the set of unique words from retrieved facts from knowledge corpus **F**, K_i being the set of unique words of predicted missing knowledge and $\hat{K}_i$ being the set of unique words of the gold missing knowledge .

9.3.3.4 Word Union Model

To see if abduction helps, we compare the above models with a Word Union Model. To extract the candidate words for missing knowledge, we used the set of unique words from both the hypothesis and OpenBook knowledge as candidate keywords. The model can be formally represented with the following:

$$K_{ij} = (H_{ij} \cup F_{ij}) \quad \forall j \in \{1, 2, 3, 4\}$$

9.3.4 Information Gain based Re-ranking

In our experiments we observe that, BERT QA model gives a higher score if similar sentences are repeated, leading to wrong classification. Thus, we introduce Information Gain based Re-ranking to remove redundant information.

We use the same BERT Knowledge Extraction model Trained on OpenBookQA data (section 9.3.2.2), which is used for extraction of knowledge facts from corpus **F** to do an initial ranking of the retrieved missing knowledge **K**. The scores of this knowledge extraction model is used as relevancy score, *rel*. To extract the top ten missing knowledge **K**, we define a redundancy score, red_{ij} , as the maximum cosine similarity, *sim*, between the previously selected missing knowledge, in the previous iterations till i, and the candidate missing knowledge K_j. If the last selected missing knowledge is K_i, then

$$red_{ij}(K_j) = max(red_{i-1,j}(K_j), sim(K_i, K_j))$$

$$rank\_score = (1 - red_{i,j}(K_j)) * rel(K_j)$$

For missing knowledge selection, we first take the missing knowledge with the highest *rel* score. From the subsequent iteration, we compute the redundancy score with the

last selected missing knowledge for each of the candidates and then rank them using the updated *rank_score*. We select the top ten missing knowledge for each $\mathbf{H_{ij}}$.

9.3.5 Question Answering

Once the OpenBook knowledge facts **F** and missing knowledge **K** have been extracted, we move onto the task of answering the questions.

9.3.5.1 Question-Answering Model

We use BERT Large model for the question answering task. For each question, we create a passage using the extracted facts and missing knowledge and fine-tune the BERT Large model for the QA task with one additional feed-forward layer for classification. The passages for the train dataset were prepared using the knowledge corpus facts, **F**. We create a passage using the top N facts, similar to the actual gold fact $\hat{\mathbf{F}}_\mathbf{i}$, for the train set. The similarities were scored using the STS-B trained model (section 9.3.2.1). The passages for the training dataset do not use the gold missing knowledge $\hat{\mathbf{K}}_\mathbf{i}$ provided in the dataset. For each of our experiments, we use the same trained model, with passages from different IR models.

The BERT Large model limits passage length to be lesser than equal to 512. This restricts the size of the passage. To be within the restrictions we create a passage for each of the answer options, and score for all answer options against each passage. We refer to this scoring as *sum score*, defined as follows:

For each answer options, A_j, we create a passage P_j and score against each of the answer options A_i. To compute the final score for the answer, we sum up each

F	Any Passage			Correct Passage			Accuracy(%)		
N	TF-IDF	Trained	STS-B	TF-IDF	Trained	STS-B	TF-IDF	Trained	STS-B
1	228	258	288	196	229	234	52.6	63.6	59.2
2	294	324	347	264	293	304	57.4	**66.2**	60.6
3	324	358	368	290	328	337	59.2	65.0	60.2
5	350	391	398	319	**370**	366	61.6	65.4	62.8
7	356	411	411	328	**390**	384	59.4	65.2	61.8
10	373	423	420	354	**405**	396	60.4	65.2	59.4

Table 33: Compares (a) The number of correct facts that appears across any four passages (b) The number of correct facts that appears in the passage of the correct hypothesis (c) The accuracy for TF-IDF, BERT model trained on STS-B dataset and BERT model trained on OpenBook dataset. N is the number of facts considered.

individual scores. If Q is the question, the score for the answer is defined as

$$Pr(Q, A_i) = \sum_{j=1}^{4} score(P_j, Q, A_i)$$

where *score* is the classification score given by the BERT Large model. The final answer is chosen based on,

$$A =_A Pr(Q, A_i)$$

9.3.5.2 Passage Selection and Weighted Scoring

In the first round, we score each of the answer options using a passage created from the selected knowledge facts from corpus **F**. For each question, we ignore the passages of the answer options which are in the bottom two. We refer to this as *Passage Selection*. In the second round, we score for only those passages which are selected after adding the missing knowledge **K**.

We assume that the correct answer has the highest score in each round. Therefore we multiply the scores obtained after both rounds. We refer to this as *Weighted Scoring*.

We define the combined passage selected scores and weighted scores as follows :

$$Pr(\mathbf{F}, Q, A_i) = \sum_{j=1}^{4} score(P_j, Q, A_i)$$

where P_j is the passage created from extracted OpenBook knowledge, **F**. The top two passages were selected based on the scores of $Pr(\mathbf{F}, Q, A_i)$.

$$Pr(\mathbf{F} \cup \mathbf{K}, Q, A_i) = \sum_{k=1}^{4} \delta * score(P_k, Q, A_i)$$

where $\delta = 1$ for the top two scores and $\delta = 0$ for the rest. P_k is the passage created using both the facts and missing knowledge. The final weighted score is :

$$wPr(Q, A_i) = Pr(\mathbf{F}, Q, A_i) * Pr(\mathbf{F} \cup \mathbf{K}, Q, A_i)$$

The answer is chosen based on the top weighted scores as below:

$$A =_A wPr(Q, A_i)$$

9.4 Experiments

9.4.1 Dataset and Experimental Setup

The dataset of OpenBookQA contains 4957 questions in the train set and 500 multiple choice questions in validation and test respectively. We train a BERT Large based QA model using the top ten knowledge facts from the corpus **F**, as a passage for both training and validation set. We select the model which gives the best score for the validation set. The same model is used to score the validation and test set with different passages derived from different methods of Abductive IR. The best Abductive IR model, the number of facts from **F** and **K** are selected from the best validation scores for the QA task.

9.4.2 OpenBook Knowledge Extraction

Question: .. *they decide the best way to save money is ?* (A) to quit eating lunch out (B) to make more phone calls (C) to buy less with monopoly money (D) to have lunch with friends

Knowledge extraction trained with STS-B:

using less resources usually causes money to be saved

a disperser disperses

each season occurs once per year

Knowledge extraction trained with OpenBookQA:

using less resources usually causes money to be saved

decreasing something negative has a positive impact on a thing

conserving resources has a positive impact on the environment

Table 33 shows a comparative study of our three approaches for OpenBook knowledge extraction. We show, the number of correct OpenBook knowledge extracted for all of the four answer options using the three approaches TF-IDF, BERT model trained on STS-B data and BERT model Trained on OpenBook data. Apart from that, we also show the count of the number of facts present precisely across the correct answer options. It can be seen that the Precision@N for the BERT model trained on OpenBook data is better than the other models as N increases.

The above example presents the facts retrieved from BERT model trained on OpenBook which are more relevant than the facts retrieved from BERT model trained on STS-B. Both the models were able to find the most relevant fact, but the other facts for STS-B model introduce more distractors and have lesser relevance. The impact of this is visible from the accuracy scores for the QA task in Table 33 . The best performance of the BERT QA model can be seen to be 66.2% using only OpenBook facts.

9.4.3 Abductive Information Retrieval

We evaluate the abductive IR techniques at different values for number of facts from **F** and number of missing knowledge **K** extracted using IR. Figure 21 shows the accuracy against different combinations of **F** and **K** , for all four techniques of IR prior to Information gain based Re-ranking. In general, we noticed that the trained models performed poorly compared to the baselines. The Word Symmetric Difference model performs better, indicating abductive IR helps. The poor performance of the trained models can be attributed to the challenge of learning abductive inference.
For the above example it can be seen, the pre-reranking facts are relevant to the question but contribute very less considering the knowledge facts retrieved from the corpus **F** and the correct answer. Figure 22 shows the impact of Information gain based Re-ranking. Removal of redundant data allows the scope of more relevant information being present in the Top N retrieved missing knowledge **K**.

Question: *A red-tailed hawk is searching for prey. It is most likely to swoop down on what?*

(A) an eagle (B) a cow (C) **a gecko** (D) a deer

Fact from F : hawks eats lizards

Pre-Reranking K :

red-tail hawk in their search for prey

Red-tailed hawks soar over the prairie and woodlands in search of prey.

Post-Reranking K:

Geckos - only vocal lizards.

Every gecko is a lizard.

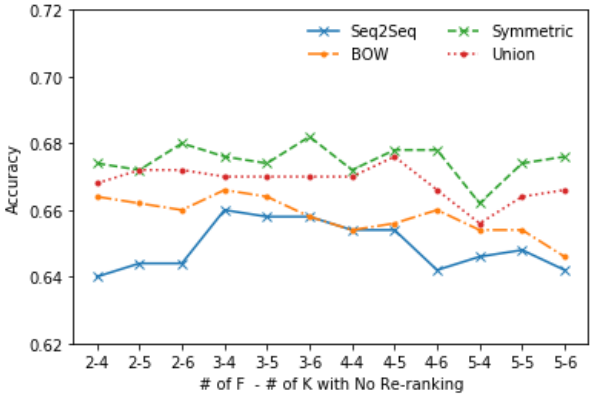

Figure 21: Accuracy v/s Number of facts from **F** - number of facts from **K**, without Information Gain based Re-ranking for 3 abductive IR models and Word Union model.

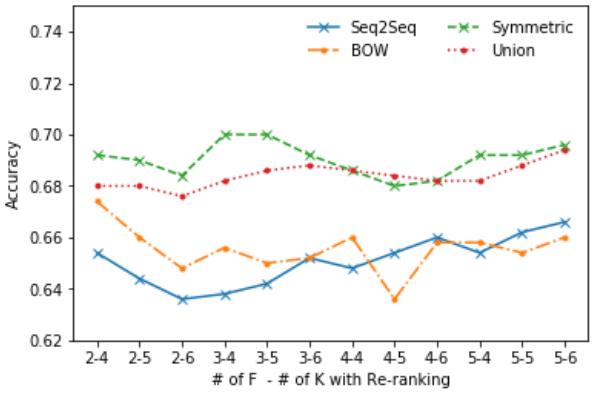

Figure 22: Accuracy v/s Number of facts from **F** - number of facts from **K**, with Information Gain based Re-ranking for 3 abductive IR models and Word Union model.

9.4.4 Question Answering

Table 34 shows the incremental improvement on the baselines after inclusion of carefully selected knowledge.

Passage Selection and Weighted Scoring are used to overcome the challenge of boosted prediction scores due to cascading effect of errors in each stage.

Solver	**Accuracy** (%)
Leaderboard	
Guess All ("random■)	25.0
Plausible Answer Detector	49.6
Odd-one-out Solver	50.2
Question Match	50.2
Reading Strategies	55.8
Model - BERT-Large (SOTA)	
Only Question (No KB)	60.4
Model - BERT-Large (Our)	
F - TF-IDF	61.6
F - Trained KE	66.2
F ∪ **K**	70.0
F ∪ **K** with Weighted Scoring	70.4
F ∪ **K** with Passage Selection	70.8
F ∪ **K** with Both	**72.0**
Oracle - BERT-Large	
F gold	74.4
F ∪ **K** gold	92.0

Table 34: Test Set Comparison of Different Components. Current state of the art (SOTA) is the Only Question model. K is retrieved from Symmetric Difference Model. KE refers to Knowledge Extraction.

Question: *What eat plants?* (A) leopards (B) eagles (C) owls (D) **robin**
Appropriate extracted Fact from F : some birds eat plants
Wrong Extracted Fact from F : a salamander eats insects
Wrong Retrieved Missing Knowledge: Leopard geckos eat mostly insects

For the example shown above, the wrong answer *leopards* had very low score with only the facts extracted from knowledge corpus **F**. But introduction of missing knowledge from the wrong fact from **F** boosts its scores, leading to wrong prediction. Passage selection helps in removal of such options and Weighted Scoring gives preference to

those answer options whose scores are relatively high before and after inclusion of missing knowledge.

9.5 Analysis & Discussion

9.5.1 Model Analysis

BERT Question Answering model: BERT performs well on this task, but is prone to distractions. Repetition of information leads to boosted prediction scores. BERT performs well for lookup based QA, as in RCQA tasks like SQuAD. But this poses a challenge for Open Domain QA, as the extracted knowledge enables lookup for all answer options, leading to an adversarial setting for lookup based QA. This model is able to find the correct answer, even under the adversarial setting, which is shown by the performance of the *sum score* to select the answer after passage selection.

Symmetric Difference Model This model improves on the baseline Word Union model by 1-2%. The improvement is dwarfed because of inappropriate domain knowledge from **F** being used for abduction. The intersection between the inappropriate domain knowledge and the answer hypothesis is ∅, which leads to queries which are exactly same as the Word Union model.

Supervised learned models The supervised learned models for abduction underperform. The Bag of Words and the Seq2Seq models fail to extract keywords for many **F – H** pairs, sometimes missing the keywords from the answers. The Seq2Seq model sometimes extracts the exact missing knowledge, for example it generates "*some birds is robin*" or "*lizard is gecko*". This shows there is promise in this approach

[3]No Passage Selection and Weighted Scoring.

and the poor performance can be attributed to insufficient train data size, which was 4957 only. A fact verification model might improve the accuracy of the supervised learned models. But, for many questions, it fails to extract proper keywords, copying just a part of the question or the knowledge fact.

9.5.2 Error Analysis

Other than errors due to distractions and failed IR, which were around 85% of the total errors, the errors seen are of four broad categories.

Temporal Reasoning: In the example [4] shown below, even though both the options can be considered as night, the fact that 2:00 AM is more suitable for the bats than 6:00 PM makes it difficult to reason. Such issues accounted for 5% of the errors.

Question: *Owls are likely to hunt at?*
(A) 3:00 PM (B) **2:00 AM** (C) 6:00 PM (D) *7:00 AM*

Negation: In the example shown below, a model is needed which handles negations specifically to reject incorrect options. Such issues accounted for 1% of the errors.

Question: *Which of the following is not an input in photosynthesis?* (A) *sunlight* (B) **oxygen** (C) water (D) carbon dioxide

Conjunctive Reasoning: In the example as shown below, each answer options are partially correct as the word " *bear*" is present. Thus a model has to learn whether all parts of the answer are true or not, i.e Conjunctive Reasoning. Logically, all answers are correct, as we can see an "or", but option (A) makes more sense. Such issues

[4] Predictions are in italics, Correct answers are in Bold.

accounted for 1% of the errors.

Question: *Some berries may be eaten by* (A) **a bear or person** (B) *a bear or shark* (C) a bear or lion (D) a bear or wolf

Qualitative Reasoning: In the example shown below, each answer options would stop a car but option (D) is more suitable since it will stop the car quicker. A deeper qualitative reasoning is needed to reject incorrect options. Such issues accounted for 8% of the errors.

Question: *Which of these would stop a car quicker?* (A) a wheel with wet brake pads (B) *a wheel without brake pads* (C) a wheel with worn brake pads (D) **a wheel with dry brake pads**

Chapter 10

EXPLORING WAYS TO INCORPORATE ADDITIONAL KNOWLEDGE TO IMPROVE NATURAL LANGUAGE COMMONSENSE QUESTION ANSWERING

10.1 Introduction

Continuing from previous chapter here I will describe my analysis and experiments with the three remaining category 3 tasks.

In recent months language models such as GPT Radford et al. 2018, BERT Devlin et al. 2019 and their variants (such as RoBERTa Liu et al. 2019) that have been pre-trained on Wikipedia articles and books are able to perform very well on many of the natural language question answering tasks. Most often they do better than models specifically designed for specific datasets and these days they form the defacto base line for most new datasets that are proposed. Some times, they even perform at superhuman level, on newly proposed natural language QA datasets Rajpurkar et al. 2016; Zellers et al. 2018. These models do well even on some of the question answering tasks where question answering seemingly requires knowledge beyond what is given in the QA items. Perhaps it is because some of the needed knowledge that may be present in textual form is "encapsulated" by the language model based systems as they are trained on huge text corpora. But one may wonder whether more can be done; i.e., can the performance be improved by further infusion of the needed knowledge (or a knowledge base containing the needed knowledge), and what are ways of doing such knowledge infusion. Few months back DARPA

and Allen AI upped the ante by developing several question answering challenges where commonsense knowledge and reasoning with them is expected to play an important rule. The expected additional challenge in these domains is that often commonsense knowledge is not readily available in textual form. To answer the above mentioned questions I consider three of those QA challenges: Abductive NLI, Physical Interaction QA and Social Interaction QA.

In this chapter, I explore ways to infuse knowledge into any language model to reason and solve multiple choice question answering task. Considering a baseline performance of BERT whole-word-masked model, I improve the performance on each of the datasets with three strategies. First, in *revision strategy*, I fine-tune the BERT model on a knowledge-base (KB) which has knowledge statements relevant to that of each of the datasets and then use the model to answer questions. In the second, *Open-Book Strategy*, I choose a certain number of knowledge statements from the KB that are textually similar to each of the samples of the datasets. Then I fine-tune the pre-trained BERT model for the question answering task to choose the answer. In the final strategy, I take the advantage of both the above mentioned strategies. I first fine-tune the pre-trained BERT model on the KB and then use additional knowledge extracted for each sample for the question-answering.

To use the extracted knowledge from the KB, I propose five models, *concat*, *max*, *simple sum*, *weighted sum*, *mac*. Each of the models use knowledge in a different way to choose the correct answer among the options.

Apart from these I have created a dataset, Parent and Family QA. The first dataset is intended to test BERT's memorizing ability for MCQ questions in a controlled environment, while the other is to test BERT's ability for answering MCQ questions with necessary information scattered over multiple knowledge sentences.

Abductive NLI	Physical IQA	Social IQA	Parent & Family QA
Obs1: Ron started his new job as a landscaper today. **Obs2**: Ron is immediately fired for insubordination. Hyp1: **Ron ignores his boss's orders and called him an idiot.** Hyp2: Ron's boss called him an idiot. **Knowledge :** Jimmy had one job to destroy the barracks. Everyone pleaded with him to do it. Alas Jimmy was hard-headed and would not listen. Instead of destroying the barracks, Jimmy went to the jungle. Jimmy was fired for insubordination.	**Goal**: When doing sit-ups **(A) place your tongue in the roof of your mouth it will stop you from straining your neck** (B) place your elbow in the roof of your mouth it will stop you from straining your neck. **Knowledge :** How to Do Superbrain Yoga.Place your tongue on the roof of your mouth	**Question**: Remy was an expert fisherman and was on the water with Kai. Remy baited Kai's hook. . What will Remy want to do next? **(A) cast the line** (B) put the boat in the water (C) invite Kai out on the boat **Knowledge :** Alex baits Pat's hook as a result others want to cast their line.	**Question**: Who is the grandparent of John ? **(A) John** (B) Johan (C) Johni (D) Joan **Knowledge :** The parent of John Radcliffe (died 1568) is Mary Arundell (courtier).The parent of John Radcliffe (died 1568) is Robert Radcliffe
Obs1: Sandy lived in New York. **Obs2**: Sandy was prepared Hyp1: **It stormed in New York** Hyp2: She partied all night. **Knowledge :** Sandy lived in New York. Sandy heard of a dangerous snow storm coming her way. Sandy decided to ensure that her home was prepared. During the snowstorm, the power went out. Sandy was prepared.	**Goal:** How to make the color orange with paint? **(A) mix together red and yellow paint** (B) mix together blue and yellow paint **Knowledge :** How to Make Paint Colors.Mix yellow and red to make orange	**Question:** Remy saved the town from destruction after the tornado had hit. . How would Others feel as a result? **(A) very grateful** (B) glad that they saved the town (C) very bitter **Knowledge :** Peyton saves the city from destruction . as a result others feel gratitude.	**Question:** Who is the grandparent of Igwe ? (A) Ilse **(B) Igwe** (C) Inwa (D) Ngwe **Knowledge :** The parent of Igwe Orizu I (Eze Ugbonyamba) is Igwe Iwuchukwu Ezeifekaibeya. The parent of Inwa Mibaya is Atula Thiri Maha Yaza Dewi.

Figure 23: Examples of Abductive NLI, Social IQA, Physical IQA and Parent & Family QA datasets with retrieved knowledge

10.2 MCQ Datasets

For the study of how to incorporate knowledge, we need datasets which are shown to need external knowledge for question-answering systems to be able to answer. We chose four datasets to evaluate our models, each with a different kind of common sense knowledge. Out of the four, three are made publicly available recently by Allen AI researchers and one is generated synthetically. To incorporate additional knowledge, we choose appropriate knowledge bases that are relevant to each of the datasets. The knowledge paragraphs are retrieved using Information Retrieval and Re-ranking methods.

10.2.1 Datasets

10.2.1.1 Abductive Natural Language Inference (aNLI)

This benchmark dataset Bhagavatula et al. 2019 is intended to judge potential of an AI system to do abductive reasoning and common sense in order to form possible explanations for a given set of observations. The dataset consists of a total of 169,654 training examples and 1532 validation examples. Given a pair of observations (O_1) and (O_2), the task is to find which of the hypothesis options (H_1) or (H_2) better explains the observations.

10.2.1.2 Physical Interaction QA

This commonsense QA benchmark is created to evaluate the physics reasoning capability of an AI system. The dataset requires reasoning about the use of physical objects and how we use them in our daily life. Given a goal (G) and a pair of choices (C_1) and (C_2), the task is to predict the choice which is most relevant to the goal (G). There are 16,113 training and 1,838 validation samples.

10.2.1.3 Social Interaction QA

The dataset is a collection of instances about reasoning on social interaction and the social implications of their statements. Given a context (C) of a social situation and a question (Q) about the situation, the task is to choose the most appropriate answer options (AO_i) out of three choices. There are several question types in this datasets,

which are derived from ATOMIC inference dimensions Sap, Rashkin, et al. 2019; Sap, Le Bras, et al. 2019. In total, there are 33,410 training and 1,954 validation samples.

10.2.1.4 Parent and Family QA

We synthetically create this dataset to test both, the memorizing capability of neural language models and the ability to combine knowledge spread over multiple sentences. The knowledge retrieved for the three datasets mentioned in the above subsections, may be error prone and in some cases, absent. This is due to the errors from the Information Retrieval step. We create this synthetic dataset to have a better control over the knowledge and ensure we do have the appropriate knowledge to answer the questions.

The source of this dataset is DBPedia Auer et al. 2007, from which we query for people and extract their parent information. Using this information, we generate 3 kinds of questions, which are, *Who is the parent of X?*, *Who is the grandparent of X?* and *Who is the sibling of X?*. The dataset has a question (Q) and 4 answer options (AO_i). The names of a parent and their family members have many things in common, which can be used to answer such a question. To make the task harder, we remove middle and last names from the answer options. To select wrong answer options, we select those names which are at an edit distance of one or two. This ensures, all the answer options are nearly same, and to actually answer the question, the system needs to have the appropriate knowledge. We also ensure all three kinds of questions for a particular person be present in that particular training or validation set. In total, there are 7,4035 training, 9,256 validation and 9,254 test questions.

10.2.2 Knowledge Sources

Reasoning with data from each of the above mentioned datasets, needs some commonsense knowledge. I choose four different knowledge bases for each of them.

For aNLI, we retrieve knowledge from the *Story Cloze Test* and *ROCStories Corpora* Mostafazadeh et al. 2016. Most of the examples in aNLI are based on everyday life stories which depict commonsense relations among daily life activities. Corpora consists of set of five sentence stories about daily life events. These are suitable for the situations present in the aNLI dataset. There are 101903 stories in the entire corpora consisting of ROCStories winter 2017 set, ROCStories spring 2016 set, Story Cloze Test Spring 2016 validation and test set.

Wikihow dataset Koupaee and Wang 2018 is an ideal commonsense knowledge-base for solving questions of PhysicalIQA dataset. This is a large collection of paragraphs of detailed steps or actions needed to complete a task. The answers of these *How* type questions mostly deals with interactions of humans with physical objects in our surroundings in everyday life. We selected only the titles and headlines from the answers of around 214,544 questions from the dataset and cleaned them to create paragraphs. We ignored the details of each points to reduce the volume of the knowledge.

For Social IQA, we synthetically generate a knowledge-base from the events and inference dimensions provided by the *ATOMIC dataset* Sap, Le Bras, et al. 2019. The ATOMIC dataset contains events and eight types of if-then inferences. The total number of events are 732,723. Some events are masked, which we fill by using a BERT Large model and the Masked Language Modelling task Devlin et al. 2019. We

extend the knowledge source, and replace *PersonX* and *PersonY*, as present in the original ATOMIC dataset, using gender neutral names.

For Parent and Family QA, we already possess the gold knowledge sentences. The knowledge for these questions are represented with a simple sentence, *The parent of X is Y.* We do not provide knowledge sentences for questions about *grandparents* and *siblings*. To answer such questions, the systems need to combine information spread over multiple sentences. Nearly all language models are trained over Wikipedia, so all language models would have seen this knowledge.

10.2.3 Relevant Knowledge Extraction

For knowledge retrieval, we use a similar approach as in Banerjee et al. 2019. We first use an information retrieval model and then re-rank using Information Gain based Re-ranking. The query is generated using a simple heuristic of unique non-stopwords present in the question, answer option and context if present. For each dataset, we select the top ten knowledge sentences.

Examples of each dataset and their retrieved knowledge from respective KBs are shown in Figure 23.

10.3 Standard BERT MCQ Model

After extracting relevant knowledge from the respective KBs, we move onto the task of Question Answering. In all our experiments we use BERT's uncased whole-word-masked model ($BERT_{UWWM}$) Devlin et al. 2019.

10.3.1 Question Answering Model

As a baseline model, we used pre-trained $BERT_{UWWM}$ for the question answering task with an extra feed-forward layer for classification as a fine-tuning step.

10.4 Modes of Knowledge Infusion

We experiment with five different models of using knowledge with the standard BERT architecture for the open-book strategy. Each of these modules take as input a problem instance which contains a question Q, n answer choices $a_1, ..., a_n$ and a list called *premises* of length n. Each element in *premises* contains m number of knowledge passages which might be useful while answering the question Q. Let k_{ij} denotes the j-th knowledge passage for the i-th answer option. Each model computes a score $score(i)$ for each of the n answer choices. The final answer is the answer choice that receives the maximum score. Here, we describe how the different models compute the scores differently.

10.4.1 Concat

In this model, all the m knowledge passages for the $i - th$ choice is joined together to make a single knowledge passage k_i. The sequence of tokens {[CLS] K_i [SEP] Qa_i [SEP]} is then passed to BERT to pool the [CLS] embedding from the last layer. This way we get n [CLS] embeddings for n answer choices, each of which is projected to a real number ($score(i)$) using a linear layer.

10.4.2 Parallel-Max

For each answer choice a_i, it uses each of the knowledge passage k_{ij} to create the sequence {[CLS] K_{ij} [SEP] Qa_i [SEP]} which is then passed to the BERT model to obtain the [CLS] embedding from the last layer which is then projected to a real number using a linear layer. $score(i)$ is then taken as the maximum of the m scores obtained using each of the m knowledge passage.

10.4.3 Simple Sum

Unlike the previous model, *simple sum* and the next two models assume that the information is scattered over multiple knowledge passages and try to aggregate those scattered information. To do this, the *simple sum* model, for each answer choice a_i and each of the knowledge passage k_{ij} creates the sequence {[CLS] K_{ij} [SEP] Qa_i [SEP]} which it then passes to the BERT model to obtain the [CLS] embedding from the last layer. All of these m vectors are then summed to find the summary vector, which then is projected to a scalar using a linear layer to obtain the $score(i)$.

10.4.4 Weighted Sum

The *weighted sum* model unlike the *simple sum* computes a weighted sum of the [CLS] embeddings as some of the knowledge passage might be more useful than others. It computes the [CLS] embeddings in a similar way to that of the *simple sum* model. It computes a scalar weight w_{ij} for each of the m [CLS] embedding using a linear projection layer which we will call as the *weight layer*. The weights are then

normalized through a softmax layer and used to compute the weighted sum of the [CLS] embeddings. It then uses (1) a new linear layer or (2) reuses the weight layer (*tied version*) to compute the final score $score(i)$ for the option a_i. We experiment with both of these options.

10.4.5 MAC

The Multi-Sentence Alignment Classification (MAC) model, similar to the *weighted sum* model, computes a weight-sum of the m [CLS] embeddings however with an additional weight-adjustment step. It first obtains a score w_{ij} for a knowledge passage k_{ij} following the *weighted sum* model and normalize them with a softmax. It then reduces the normalized scores further using the following formula:

$$w'_{ij} = w_{ij} - (1 - w_{ij}) * max_{j \neq l \wedge l \in \{1 \ldots m\}} \{link\_strength_{ijl}\} \tag{10.1}$$

Here, $link\_strength_{ijl} \in [0, 1]$ captures how well the two knowledge passage k_{ij} and k_{il} can be "joined" in the sense of joining rows of two tables. Intuitively we want a high *link strength* score between the two knowledge passages "Facebook was launched in Cambridge" and "Cambridge is in MA" but the score should be less for "Facebook was launched in Cambridge" and "Boston is in MA". If two knowledge passage has good *link strength* score then probably they can be joined to infer new information such as "Facebook was launched in MA". The intuition of the weight reduction in equation 10.1 is that if k_{il} is not strong enough to support the answer choice a_i and it cannot be "joined" with another knowledge passage then probably there is no need to consider it during the final prediction stage. See that if w_{ij} is too close to 1 i.e. if a k_{ij} is very informative, the penalty because of "joinable" or not is negligible. It only becomes prominent when w_{ij} neither too low or too high.

Dataset	Strategy	Concat	Max	Sim-Sum	Wtd-Sum	Mac
Abductive NLI	Only Openbook	73.89	73.69	73.50	73.26	73.69
	Only Revision	72.65	NA	NA	NA	NA
	Revision & Openbook	74.35	74.28	74.02	75.13	74.15
Physical IQA	Only Openbook	67.84	72.41	72.58	72.52	75.52
	Only Revision	74.53	NA	NA	NA	NA
	Revision & Openbook	67.74	73.83	76.76	76.82	75.46
Social IQA	Only Openbook	70.22	67.75	70.21	69.96	70.26
	Only Revision	69.45	NA	NA	NA	NA
	Revision & Openbook	68.80	66.56	68.86	69.29	70.01
Parent & Family QA	Only Openbook	91.21	89.8	93.16	91.96	91.15
	Only Revision	78.30	NA	NA	NA	NA
	Revision & Openbook	87.21	91.92	93.32	90.63	91.20

Table 35: Performance of each of the five models (Concat, Max, simple sum, Weighted sum, mac) across four datasets with external knowledge.

Dataset	Model	Dev	Test
Abductive NLI	Baseline	67.36	66.75
	Baseline (Ours)	70.36	NA
	Best Model	75.13	**74.96**
Physical IQA	Baseline	70.89	69.23
	Baseline (Ours)	71.44	NA
	Best Model	75.63	**72.28**
Social IQA	Baseline	66.00	64.50
	Baseline (Ours)	68.86	NA
	Best Model	70.36	**67.53**
Parent & Family QA	Baseline	NA	NA
	Baseline (Ours)	77.85	76.96
	Best Model	93.32	91.24

Table 36: Performance of the best knowledge infused model on the Test set. State-of-the-art models are in bold.

The *link strength* score $link\_strength_{ijl}$ can be computed in different ways. Here we show a memory-efficient way. Since, loading BERT itself takes lot of memory if we create sequences like {[CLS] K_{ij} [SEP] k_{il} [SEP]} to compute the $link\_strength_{ijl}$ score, it will add a lot of memory overhead and if m is big, it might throw memory exceptions. Here we show how we compute the link strength scores from the BERT outputs of the {[CLS] k_{ij} [SEP] Qa_i [SEP]} sequences without producing any additional {[CLS] K_{ij} [SEP] k_{il} [SEP]} sequences. We take the last layer output from the BERT model and use the segment id information (see that segment id for the tokens

starting from [CLS] to the first [SEP] token is 0 and is 1 for the remaining tokens) to extract only the token embeddings that belongs to the knowledge passage k_{ij}. Let $h_{ij}^1, ..., h_{ij}^p$ be those token embeddings. We compute a link vector $link_{ij}$ from these token embeddings for the the knowledge passage k_{ij}. The score $link\_strength_{ijl}$ is then computed as follows:

$$link\_strength_{ijl} = \frac{exp(link_{ij}^T \; link_{il})}{\sum_{x \neq j}^{x=1...m} exp(link_{ij}^T \; link_{ix})}$$

To compute the link vector $link_{ij}$ we first pass each token embedding h_{ij}^t through a linear layer which assigns a scalar score s_{ij}^t denoting whether h_{ij}^t should be part of link description $link_{ij}$ or not. The link vector is then calculated as follows:

$$link_{ij} = \sum_{t=1}^{p} s_{ij}^t * h_{ij}^t$$

10.5 Related Works

Datasets like SQuAD Rajpurkar et al. 2016, TriviaQA Joshi et al. 2017, WikiQA Yang, Yih, and Meek 2015, CoQA Reddy, Chen, and Manning 2019 have gained enormous attention over the past few years. Various models have been proposed to solve them. The questions from these datasets are easy to solve since the answers are present in either the passages, contexts or in the options itself.

A more challenging task is, when the multiple choice questions do not have sufficient knowledge to answer correctly given a passage, context or options like ARC Clark et al. 2018, RACE Lai et al. 2017, OpenBook QA Mihaylov et al. 2018b. But the language models trained on huge amount of data have been able to solve them quite comfortably.

Our focus in this paper is on datasets which not only requires external facts but also commonsense knowledge to predict the correct options like Abductive NLI Bhagavatula et al. 2019, Physical IQA AI 2018 and Social IQA Sap, Rashkin, et al. 2019.

10.6 Experiments

Let D be an MCQ dataset and T be a pre-trained language model, K_D be a knowledge base (a set of paragraphs or sentences) which is useful for D and let K be a general knowledge base where T was pre-trained and K might or might not contain K_D. We took three approaches to infuse knowledge.

10.6.1 Revision Strategy

In this strategy, T is fine-tuned on K_D with respect to Masked LM and next sentence prediction task and then fine-tuned on the dataset D with respect to the Question Answering task.

10.6.2 Open Book strategy

Here a subset of K_D is assigned to each of the training samples on the dataset D and the model T is fine-tuned on the modified dataset D.

10.6.3 Revision along with an Open Book Strategy

In this strategy, T is fine-tuned on K_D with respect to Masked LM and next sentence prediction task and also a subset of K_D is assigned to each of the training samples on D. The model is then fine-tuned with respect to the modified dataset as a Question Answering task.

10.6.4 Results

Table 35 and Table 36 show summary of our experiments on the four datasets. We can see knowledge helps in improving the performance of neural language models. Both the Open Book and the Revision strategy works, together the performance improves even further. We achieve state of the art performances on aNLI, Social IQA and Physical IQA datasets.

The performance of the Revision strategy is poor for the Social IQA dataset. The reason behind this drop in performance can be attributed to the synthetic nature of the sentences and the unavailability of next sentence prediction task data. This leads to a decrease in the performance of the language model. All the sentences in the KB for Social IQA are single sentence statements, and not paragraphs. The results for Physical IQA and Abductive NLI datasets are better due to the presence of natural and contiguous knowledge sentences.

10.7 Discussion and Error Analysis

To understand how knowledge is used and whether the knowledge is useful or not, we do the following analysis: For each of the datasets we have randomly selected 100 samples where our best performing model predicts correctly and 100 samples where it has failed. We identified the following broad categories of analysis.

For the correct predictions, we check, (1) Exact appropriate knowledge is present, (2) A related but relevant knowledge is present, (3) Knowledge is present only in the correct option, and (4) No knowledge is present. Figure 24 shows the counts for the above categories. All the cases do not occur in all the datasets.

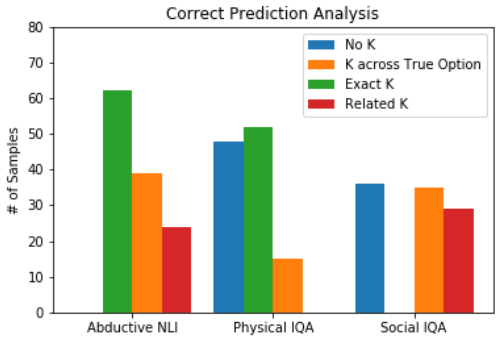

Figure 24: Measure of performance across different knowledge presence in correct predictions

For the errors (Figure 25), we analyze, (1) Is the knowledge insufficient, (2) Is the knowledge present in the wrong answer, (3) Knowledge is appropriate but model fails, and (4) Gold label is questionable.

We also analyze given appropriate knowledge, how the model performs. From Figure 24, it can be seen that BERT can answer quite a number of question without knowledge.

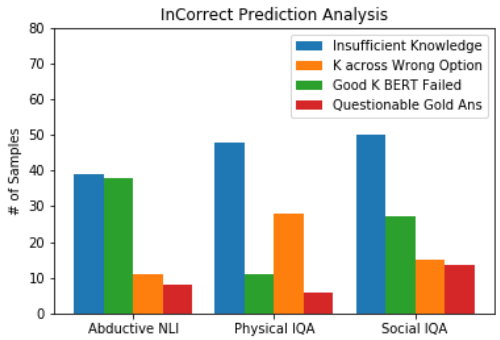

Figure 25: Measure of performance across different knowledge presence in incorrect predictions.

Also from Figure 25, it is clear that inspite of having good knowledge, BERT fails to answer correctly.

In the following subsections, we analyze the different dataset specific errors.

10.7.1 Social IQA

We measure the performance across the 8 different ATOMIC inference dimensions for the best knowledge infused model. In figure 26 we can see both with and without knowledge the model performs nearly equally across all dimensions. There is no considerable improvement across any particular dimension.

In some cases the model fails to predict the correct answer inspite of the appropriate knowledge being present.

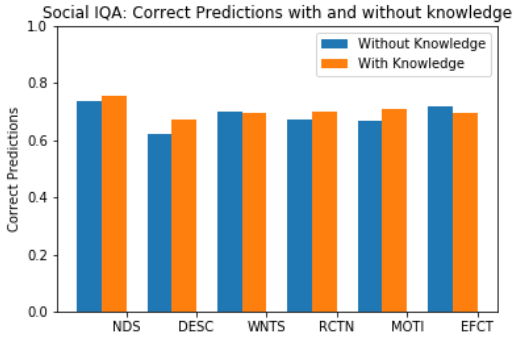

Figure 26: Performance of the model with (MAC model) and without knowledge (Baseline) across different types of ATOMIC inference dimensions.

Question: *Kendall took their dog to the new dog park in the neighborhood. . What will Kendall want to do next?*

(A) *walk the dog* (B) **meet other dog owners Knowledge:** Jody takes Jody's dog to the dog park, as a result Jody wants to socialize with other dog owners.

In the above example, the above knowledge was retrieved but still the model predicted the wrong option. 341 questions were predicted wrongly after addition of knowledge. We also identified out of the set of 100 analyzed correct predictions, 29% of the questions had partial information relevant to the question.

10.7.2 Parent and Family QA

In Figure 27, we see with addition of knowledge, there is a considerable improvement in performance. Other than questions asking about parents, which just need a look up to answer, the sibling and grandparent questions need models to combine information present across multiple sentences. We can see the model improves even in this

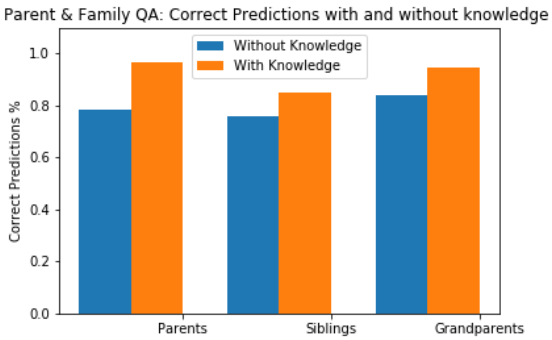

Figure 27: Performance of the model across the three different type of questions.

questions, showing knowledge infusion helps. Out of the three types of the questions, the performance is lowest on the sibling questions, indicating that it is harder for the models to perform this task. The model accuracy is reasonably good on this dataset, which shows BERT has a strong capability to memorize factual knowledge. Its performance improves with infusion of knowledge,

Here also, 1,790 questions which were previously predicted correctly, are predicted wrong with addition of knowledge.

10.7.3 Physical IQA

Out of the 100 failures that we have analysed, we found that for 8 samples the *goal* matches the knowledge statements but the answers present in the knowledge is different. As for example,

Goal: *How can I soothe my tongue if I burn it?*

(A) Put some salt on it. **(B) Put some sugar on it.**

Knowledge: How to Soothe a Burnt Tongue.Chew a menthol chewing gum.

Also, there are 33 samples in the whole train and dev dataset for which the words in one options are a subset of second option. In those cases, the knowledge retrieved is same for both the options and this confuses the BERT model.

Goal: *What can I drink wine out of if I don't have a wine glass?* (A) Just pour the wine into a regular mug or glass and drink. **(B) Just pour the wine into a regular mug or wine glass and drink.** **Knowledge:** How to Serve Foie Gras. Pour a glass of wine.

On addition of knowledge, 359 samples have become correctly predicted with our best model for Physical IQA dataset which were initially incorrect. But in the process, 166 samples which were correct in our baseline model have now been incorrectly predicted.

10.7.4 Abductive NLI

In this dataset, we also have some examples where negative knowledge is being fed to the model, and it still produces the correct output. There are 8 such examples among the 100 samples we analyzed. For example:

Obs1: *Pablo likes to eat worms.*

Obs2: *Pablo does not enjoy eating worms.*

(Hyp1) Pablo thought that worms were a delicious source of protein. **(Hyp2) Pablo then learned what worms really are.**

Knowledge: Pablo likes to eat worms. He read a book in school on how to do this. He fries them in olive oil. He likes to do this at least once a month. Pablo enjoys worms and views them as a delicacy.

Similarly, we have examples where knowledge favors incorrect hypothesis, however our system still produces correct output. We found 12 such examples among the 100 samples we analyzed. For example:

Obs1: *Dotty was being very grumpy.*

Obs2: *She felt much better afterwards.*

(Hyp1) Dotty ate something bad. **(Hyp2) Dotty call some close friends to chat.**

Knowledge: Allie felt not so good last night. She ate too much. So she had to sleep it off. Then she woke up. She felt so much better

We have 12 cases among 100 analyzed samples, where both hypothesis are very similar. So,our system is unable to produce correct output. For example:

Obs1: *Bob's parents grounded him.*

Obs2: *He came back home but his parents didn't even know he left.*

(Hyp1) Bob got caught sneaking out. **(Hyp2) Bob got away with sneaking out.**

We also have 34 examples where incorrect hypothesis has more word similarity with

the observation and knowledge, whereas correct hypothesis has been paraphrased or has less word similarity. The system predicts the wrong answer in such a situation. One such example is:

Obs1: *Mary's mom came home with more bananas than they could possibly eat.*
Obs2: *That was the best way ever to eat a banana!*
(Hyp1) Mary and her mom decided to make chocolate covered frozen bananas to avoid waste. (Hyp2) Mary made pineapple splits for everyone.
Knowledge: Mary s mom came home with more bananas than they could possibly eat. She wondered why she had bought them all. Then after dinner that night she got a surprise. Mom made banana splits for the whole family. That was the best way ever to eat a banana

Another area where the system fails, is where the problem seems to be open-ended, and many hypotheses can explain the pair of observations. It is tough to find exact knowledge in such a scenario. For example,

Obs1: *Lisa went for her routine bike ride.*
Obs2: *Some days turn out to be great adventures.*
(Hyp1) Lisa spotted a cat and followed it off trail (Hyp2) Lisa saw a lot of great food.
Knowledge: Lisa went for her routine bike ride.Only this time she noticed an abandoned house.She stopped to look in the house.It was full of amazing old antiques.Some days turn out to be great adventures.

10.8 Conclusion

In this work, we have evaluated different ways to incorporate knowledge into language models. We have pushed the current state of the art of the three commonsense knowledge tasks. We have provided five new models for multiple choice natural language QA using knowledge and analyzed their performance on these commonsense datasets. We also make a synthetic dataset available which measures the memorizing and reasoning ability of language models.

We observe that, existing knowledge bases even though do not contain all the knowledge that is needed to answer the questions, they do provide a significant amount of knowledge. BERT, even though utilizes some of the knowledge, there are areas where model can be further improved, particularly the ones where the knowledge is present but the model could not answer, and where it predicted wrong answers with irrelevant knowledge. Our future work is to analyze the source of this errors and try to explore possible solutions.

Chapter 11

FUTURE WORK & CONCLUSION

In this thesis, I have presented two new paradigms for knowledge based NLU systems and showed their efficacy with experiments on several datasets. Particularly, I have shown with the help of the scalable Inductive Logic Programming (ILP) algorithm, which I have developed as part of this research, it is possible to build knowledge based NLU systems using LKR framework which achieve state-of-the-art accuracy on various question answering datasets. However, the current ILP algorithm updates its hypothesis by looking at one example. This has drawbacks similar to that of a stochastic gradient descent when the batch size is 1. Thus, one of the future work is to parallaly look over multiple examples and make an update which works well for most of them.

The proposed framework of TKR that allows declarative programming over text requires the knowledge to be given as a logic program. However, sometimes such knowledge (rules) might be described in natural language sentences. One of my future work is to extend the TKR framework to such scenarios. The dataset of QUARTZ is a potential application requiring such functionality. Figure 28 shows an example problem from this dataset. The task is to retrieve a suitable knowledge such as "More pollutants mean poorer air quality." from a given knowledge base and then use it to answer the question.

One of my main goal is to extend my solutions to cover the category 3 questions. One possible approach to achieve it is to understand the latent structures that are present in

Q: If Mona lives in a city that begins producing a greater amount of pollutants, what happens to the air quality in that city? (A) increases (B) decreases **[correct]**

K: More pollutants mean poorer air quality.

Figure 28: A sample question from the QUARTZ dataset. The task is to retrieve a suitable knowledge such as *More pollutants mean poorer air quality* from a given knowledge base and then use it to answer the question.

pre-trained models. If such structures can be exploited to perform relational reasoning, it will not be much difficult to enable interpretable relational learning.

11.1 Conclusion

The initial popularity of Knowledge based Natural Language Understanding (NLU) systems, which was inspired by the original goals of AI, gradually decreased due to the knowledge bottleneck and the lack of good parsers. The reality that the knowledge based language understanding requires high quality machine-tractable knowledge which is expensive to build favoured supervised statistical machine learning. However, if the required knowledge can be learned at scale from the big noisy machine learning datasets the knowledge bottleneck can be successfully addressed. This proposal is a work towards this direction. A scalable learning algorithm has been developed that can learn Answer Set Programs from large noisy datasets. Also a technique to reason over text with rules in background has been developed. One further step to enable learning from text without parsing is needed to be done to make Knowledge based Natural Language Understanding (NLU) systems as applicable as the deep learning based systems.

REFERENCES

AI, Allen. 2018. "Physical IQA." *URL https://allenai.org/.*

Athakravi, Duangtida, Dalal Alrajeh, Krysia Broda, Alessandra Russo, and Ken Satoh. 2015. "Inductive learning using constraint-driven bias." In *Inductive Logic Programming,* 16–32. Springer, Cham.

Athakravi, Duangtida, Domenico Corapi, Krysia Broda, and Alessandra Russo. 2013. "Learning through hypothesis refinement using answer set programming." In *International Conference on Inductive Logic Programming,* 31–46. Springer.

Auer, Sören, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. "DBpedia: A Nucleus for a Web of Open Data." In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference,* 722–735. ISWC'07/ASWC'07. Busan, Korea: Springer-Verlag. http://dl.acm.org/citation.cfm?id=1785162.1785216.

Balduccini, Marcello, Chitta Baral, and Yuliya Lierler. 2008. "Knowledge representation and question answering." *Foundations of Artificial Intelligence* 3:779–819.

Banarescu, Laura, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. "Abstract Meaning Representation for Sembanking."

Banerjee, Pratyay, Kuntal Kumar Pal, Arindam Mitra, and Chitta Baral. 2019. "Careful Selection of Knowledge to Solve Open Book Question Answering." In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics,* 6120–6129. Florence, Italy: Association for Computational Linguistics, July. https://www.aclweb.org/anthology/P19-1615.

Baral, Chitta. 2003. *Knowledge representation, reasoning and declarative problem solving.* Cambridge university press.

Berant, Jonathan, Andrew Chou, Roy Frostig, and Percy Liang. 2013. "Semantic parsing on freebase from question-answer pairs." In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing,* 1533–1544.

Berant, Jonathan, Vivek Srikumar, Pei-Chun Chen, Abby Vander Linden, Brittany Harding, Brad Huang, Peter Clark, and Christopher D Manning. 2014. "Modeling biological processes for reading comprehension." In *Proceedings of the 2014*

Conference on Empirical Methods in Natural Language Processing (EMNLP), 1499–1510.

Bhagavatula, Chandra, Ronan Le Bras, Chaitanya Malaviya, Keisuke Sakaguchi, Ari Holtzman, Hannah Rashkin, Doug Downey, Scott Wen-tau Yih, and Yejin Choi. 2019. "Abductive Commonsense Reasoning." *arXiv preprint arXiv:1908.05739.*

Bobrow, Daniel G. 1964. "A Question-answering System for High School Algebra Word Problems." In *Proceedings of the October 27-29, 1964, Fall Joint Computer Conference, Part I,* 591–614. AFIPS '64 (Fall, part I). San Francisco, California: ACM. doi:10.1145/1464052.1464108.

Bobrow, Daniel G. 1964. "Natural language input for a computer problem solving system."

———. 2012. *Qualitative reasoning about physical systems.* Vol. 24. 1-3. Elsevier.

Bobrow, Daniel G, and Terry Winograd. 1977. "An overview of KRL, a knowledge representation language." *Cognitive science* 1 (1): 3–46.

Bos, Johan. 2008. "Wide-coverage semantic analysis with boxer." In *Proceedings of the 2008 Conference on Semantics in Text Processing.* Association for Computational Linguistics.

Bowman, Samuel R, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. "A large annotated corpus for learning natural language inference." *arXiv preprint arXiv:1508.05326.*

Brewka, Gerhard, Thomas Eiter, and Mirosław Truszczyński. 2011. "Answer set programming at a glance." *Communications of the ACM* 54 (12): 92–103.

Calimeri, Francesco, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. 2008. "Computable functions in ASP: Theory and implementation." In *International Conference on Logic Programming.* Springer.

Charniak, Eugene. 1972. "Toward a model of children's story comprehension." PhD diss., Massachusetts Institute of Technology.

Charniak, Eugene, and Robert Goldman. 1988. "A logic for semantic interpretation." In *Proceedings of the 26th annual meeting on Association for Computational Linguistics,* 87–94. Association for Computational Linguistics.

Charniak, Eugene, and Robert P Goldman. 1989. “A Semantics for Probabilistic Quantifier-Free First-Order Languages, with Particular Application to Story Understanding.” In *IJCAI,* 89:1074–1079. Citeseer.

Chen, Qian, Xiaodan Zhu, Zhen-Hua Ling, Diana Inkpen, and Si Wei. 2017. “Neural natural language inference models enhanced with external knowledge.” *arXiv preprint arXiv:1711.04289.*

———. 2018. “Neural natural language inference models enhanced with external knowledge.” In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* 1:2406–2417.

Chen, Qian, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2016. “Enhanced lstm for natural language inference.” *arXiv preprint arXiv:1609.06038.*

Choi, Eunsol, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. 2018. “QuAC: Question answering in context.” *arXiv preprint arXiv:1808.07036.*

Clark, Peter. 2015. “Elementary School Science and Math Tests as a Driver for AI: Take the Aristo Challenge!” In *AAAI,* 4019–4021.

Clark, Peter, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. “Think you have solved question answering? try arc, the ai2 reasoning challenge.” *arXiv preprint arXiv:1803.05457.*

Clark, Peter, Bhavana Dalvi, and Niket Tandon. 2018. “What Happened? Leveraging VerbNet to Predict the Effects of Actions in Procedural Text.” *arXiv preprint arXiv:1804.05435.*

Clark, Peter, and Oren Etzioni. 2016. “My Computer is an Honor Student but how Intelligent is it? Standardized Tests as a Measure of AI.” *AI Magazine.(To appear).*

Clark, Peter, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Turney, and Daniel Khashabi. 2016. “Combining retrieval, statistics, and inference to answer elementary science questions.” In *Thirtieth AAAI Conference on Artificial Intelligence.*

Dagan, Ido, Oren Glickman, and Bernardo Magnini. 2006. “The PASCAL recognising textual entailment challenge.” In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment,* 177–190. Springer.

Dagan, Ido, Dan Roth, Mark Sammons, and Fabio Massimo Zanzotto. 2013. "Recognizing textual entailment: Models and applications." *Synthesis Lectures on Human Language Technologies* 6 (4): 1–220.

Dai, Wang-Zhou, Stephen H Muggleton, and Zhi-Hua Zhou. 2015. "Logical Vision: Meta-Interpretive Learning for Simple Geometrical Concepts." In *ILP (Late Breaking Papers),* 1–16.

Dalvi, Bhavana, Lifu Huang, Niket Tandon, Wen-tau Yih, and Peter Clark. 2018. "Tracking State Changes in Procedural Text: a Challenge Dataset and Models for Process Paragraph Comprehension." In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers),* 1:1595–1604.

Das, Rajarshi, Tsendsuren Munkhdalai, Xingdi Yuan, Adam Trischler, and Andrew McCallum. 2018. "Building Dynamic Knowledge Graphs from Text using Machine Reading Comprehension." *arXiv preprint arXiv:1810.05682.*

De Marneffe, Marie-Catherine, and Christopher D Manning. 2008. *Stanford typed dependencies manual.* Technical report. Technical report, Stanford University.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805.*

———. 2019. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers),* 4171–4186. Minneapolis, Minnesota: Association for Computational Linguistics, June. doi:10.18653/v1/N19-1423.

Eiter, Thomas, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. 2006. "Effective integration of declarative rules with external evaluations for semantic-web reasoning." In *European Semantic Web Conference,* 273–287. Springer.

Feigenbaum, Edward A, and Julian Feldman. 1963. "Computers and Thought."

FitzGerald, Nicholas, Julian Michael, Luheng He, and Luke Zettlemoyer. 2018. "Large-scale qa-srl parsing." *arXiv preprint arXiv:1805.05377.*

Flanigan, Jeffrey, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A Smith. 2014. "A discriminative graph-based parser for the abstract meaning representation."

Fletcher, Charles R. 1985. "Understanding and solving arithmetic word problems: A computer simulation." *Behavior Research Methods, Instruments, & Computers* 17 (5): 565–571.

Gebser, Martin, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. 2012. "Answer set solving in practice." *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6 (3): 1–238.

Gelfond, Michael, and Yulia Kahl. 2014. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach.* Cambridge University Press.

Gelfond, Michael, and Vladimir Lifschitz. 1988. "The stable model semantics for logic programming." In *ICLP/SLP,* 88:1070–1080.

Green, Claude Cordell. 1969. *The application of theorem proving to question-answering systems.* Technical report. STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE.

Gu, Jiatao, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. "Incorporating Copying Mechanism in Sequence-to-Sequence Learning." In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* 1631–1640. Berlin, Germany: Association for Computational Linguistics. doi:10.18653/v1/P16-1154.

Gulwani, S., J. Hernandez-Orallo, E. Kitzelmann, S.H. Muggleton, U. Schmid, and B. Zorn. 2015. "Inductive programming meets the real world." *Communications of the ACM* 58 (11): 90–99.

Havur, Giray, Guchan Ozbilgin, Esra Erdem, and Volkan Patoglu. 2014. "Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach." In *Robotics and Automation (ICRA),* 445–452. IEEE.

He, Luheng, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. "Deep semantic role labeling: What works and what's next." In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).*

He, Luheng, Mike Lewis, and Luke Zettlemoyer. 2015. "Question-answer driven semantic role labeling: Using natural language to annotate natural language." In *Proceedings of the 2015 conference on empirical methods in natural language processing,* 643–653.

Henaff, Mikael, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. 2016. "Tracking the world state with recurrent entity networks." *arXiv preprint arXiv:1612.03969.*

Hinsley, Dan A, John R Hayes, and Herbert A Simon. 1977. "From words to equations: Meaning and representation in algebra word problems." *Cognitive processes in comprehension* 329.

Hobbs, Jerry R. 2004. "Abduction in natural language understanding." *Handbook of pragmatics:* 724–741.

Hobbs, Jerry R, Mark E Stickel, Douglas E Appelt, and Paul Martin. 1993. "Interpretation as abduction." *Artificial intelligence* 63 (1-2): 69–142.

Honnibal, Matthew, and Ines Montani. 2017. "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing." *To appear.*

Hosseini, Mohammad Javad, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. "Learning to solve arithmetic word problems with verb categorization." In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP),* 523–533.

Huang, Danqing, Shuming Shi, Chin-Yew Lin, and Jian Yin. 2017. "Learning fine-grained expressions to solve math word problems." In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing,* 805–814.

Jenkins, Tony. 1995. *Open Book Assessment in Computing Degree Programmes.* Citeseer.

Jijkoun, Valentin, and Maarten De Rijke. 2006. "Recognizing textual entailment: Is word similarity enough?" In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment.*

Joshi, Mandar, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. "TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Compre-

hension." In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* 1:1601–1611.

Katzouris, Nikos, Alexander Artikis, and Georgios Paliouras. 2015. "Incremental learning of event definitions with Inductive Logic Programming" [in English]. *Machine Learning* 100 (2-3): 555–585. doi:10.1007/s10994-015-5512-1.

———. 2017. "Distributed Online Learning of Event Definitions." *CoRR* abs/1705.02175. arXiv: 1705.02175. http://arxiv.org/abs/1705.02175.

Kazmi, Mishal, Peter Schüller, and Yücel Saygın. 2017. "Improving Scalability of Inductive Logic Programming via Pruning and Best-Effort Optimisation." *Expert Systems with Applications.*

Khashabi, Daniel, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. "Looking beyond the surface: A challenge set for reading comprehension over multiple sentences." In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers),* 1:252–262.

Khashabi, Daniel, Tushar Khot, Ashish Sabharwal, and Dan Roth. 2017. "Learning what is essential in questions." In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017),* 80–89.

Khot, Tushar, Ashish Sabharwal, and Peter Clark. 2018. "SciTail: A textual entailment dataset from science question answering." In *Proceedings of AAAI.*

Kintsch, Walter, and James G Greeno. 1985. "Understanding and solving word arithmetic problems." *Psychological review* 92 (1): 109.

Koncel-Kedziorski, Rik, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. "Parsing Algebraic Word Problems into Equations." *Transactions of the Association for Computational Linguistics* 3:585–597.

Koupaee, Mahnaz, and William Yang Wang. 2018. "WikiHow: A Large Scale Text Summarization Dataset." *arXiv preprint arXiv:1810.09305.*

Krishnamurthy, Jayant, Pradeep Dasigi, and Matt Gardner. 2017. "Neural semantic parsing with type constraints for semi-structured tables." In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing,* 1516–1526.

Kumar, Ankit, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. 2015. "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing." *arXiv preprint arXiv:1506.07285.*

Kushman, Nate, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. "Learning to automatically solve algebra word problems." Association for Computational Linguistics.

Kwiatkowski, Tom, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. "Lexical generalization in CCG grammar induction for semantic parsing." In *Proceedings of the conference on empirical methods in natural language processing,* 1512–1523. Association for Computational Linguistics.

Lai, Guokun, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. "RACE: Large-scale ReAding Comprehension Dataset From Examinations." In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing,* 785–794. Copenhagen, Denmark: Association for Computational Linguistics. doi:10.18653/v1/D17-1082.

Landsberger, J. 1996. "Study guides and strategies." Http://www.studygs.net/tsttak7.htm..

Law, Mark, Alessandra Russo, and Krysia Broda. 2014. "Inductive learning of answer set programs." In *European Workshop on Logics in Artificial Intelligence,* 311–325. Springer, Cham.

———. 2015. "Learning weak constraints in answer set programming." *Theory and Practice of Logic Programming* 15 (4-5): 511–525.

———. 2016. "Iterative learning of answer set programs from context dependent examples." *Theory and Practice of Logic Programming* 16 (5-6): 834–848.

LeCun, Yann. 1998. "The MNIST database of handwritten digits." *http://yann. lecun. com/exdb/mnist/.*

Lev, Iddo, Bill MacCartney, Christopher Manning, and Roger Levy. 2004. "Solving logic puzzles: From robust processing to precise semantics." In *Proceedings of the 2nd Workshop on Text Meaning and Interpretation.*

Levesque, Hector J, Ernest Davis, and Leora Morgenstern. 2012. "The Winograd schema challenge." In *KR.*

Lierler, Yuliya, Daniela Inclezan, and Michael Gelfond. 2017. "Action Languages and Question Answering." In *IWCS 2017—12th International Conference on Computational Semantics.*

Liu, Hugo, and Push Singh. 2004. "ConceptNet—a practical commonsense reasoning tool-kit." *BT technology journal* 22 (4): 211–226.

Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. "RoBERTa: A Robustly Optimized BERT Pretraining Approach." *CoRR* abs/1907.11692. arXiv: 1907.11692. http://arxiv.org/abs/1907.11692.

Manning, Christopher D, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. "The Stanford CoreNLP Natural Language Processing Toolkit." In *ACL (System Demonstrations),* 55–60.

Matsuzaki, Takuya, Takumi Ito, Hidenao Iwane, Hirokazu Anai, and Noriko H Arai. 2017. "Semantic parsing of pre-university math problems." In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* 1:2131–2141.

McCarthy, John. 1960. *Programs with common sense.* RLE / MIT computation center.

Michael, Julian, Gabriel Stanovsky, Luheng He, Ido Dagan, and Luke Zettlemoyer. 2017. "Crowdsourcing question-answer meaning representations." *arXiv preprint arXiv:1711.05885.*

Mihaylov, Todor, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018a. "Can a suit of armor conduct electricity? a new dataset for open book question answering." *arXiv preprint arXiv:1809.02789.*

———. 2018b. "Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering." In *EMNLP.*

Mihaylov, Todor, and Anette Frank. 2018. "Knowledgeable reader: Enhancing cloze-style reading comprehension with external commonsense knowledge." *arXiv preprint arXiv:1805.07858.*

Miller, George A. 1995. "WordNet: a lexical database for English." *Communications of the ACM* 38 (11): 39–41.

Mitra, Arinam, and Chitta Baral. 2018. "Incremental and Iterative Learning of Answer Set Programs from Mutually Distinct Examples." *arXiv preprint arXiv:1802.07966.*

Mitra, Arindam, and Chitta Baral. 2015. "Learning to automatically solve logic grid puzzles." In *EMNLP,* 1023–1033.

———. 2016a. "Addressing a Question Answering Challenge by Combining Statistical Methods with Inductive Rule Learning and Reasoning." In *AAAI,* 2779–2785.

———. 2016b. "Learning to use formulas to solve simple arithmetic problems." ACL.

———. 2018. "Incremental and Iterative Learning of Answer Set Programs from Mutually Distinct Examples." *Theory and Practice of Logic Programming* 18 (3-4): 623–637.

Mitra, Arindam, Peter Clark, Oyvind Tafjord, and Chitta Baral. 2019a. "Declarative Question Answering over Knowledge Bases containing Natural Language Text with Answer Set Programming." In *AAAI 2019.*

———. 2019b. "Declarative Question Answering over Knowledge Bases containing Natural Language Text with Answer Set Programming."

Mitra, Arindam, Ishan Shrivastava, and Chitta Baral. 2019. *Understanding Roles and Entities: Datasets and Models for Natural Language Inference.* arXiv: 1904.09720 [cs.CL].

Moldovan, Dan, Christine Clark, Sanda Harabagiu, and Steve Maiorano. 2003. "Cogex: A logic prover for question answering." In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1.*

Mostafazadeh, Nasrin, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. "A corpus and evaluation framework for deeper understanding of commonsense stories." *arXiv preprint arXiv:1604.01696.*

Muggleton, Stephen. 1991. "Inductive logic programming." *New generation computing* 8 (4): 295–318.

———. 1995. "Inverse entailment and Progol." *New generation computing* 13 (3-4): 245–286.

Mukherjee, Anirban, and Utpal Garain. 2008. "A review of methods for automatic understanding of natural language mathematical problems." *Artificial Intelligence Review* 29 (2): 93–122.

Musa, Ryan, Xiaoyan Wang, Achille Fokoue, Nicholas Mattei, Maria Chang, Pavan Kapanipathi, Bassem Makni, Kartik Talamadupula, and Michael Witbrock. 2018. "Answering Science Exam Questions Using Query Rewriting with Background Knowledge." *arXiv preprint arXiv:1809.05726.*

Ni, Jianmo, Chenguang Zhu, Weizhu Chen, and Julian McAuley. 2018. "Learning to attend on essential terms: An enhanced retriever-reader model for scientific question answering." *arXiv preprint arXiv:1808.09492.*

Norvig, Peter. 1983. "Frame Activated Inferences in a Story Understanding Program." In *IJCAI,* 624–626.

———. 1987. "Inference in text understanding." In *AAAI,* 561–565.

Otero, Ramon. 2001. "Induction of stable models." *Inductive Logic Programming:* 193–205.

Palmer, Martha, Daniel Gildea, and Nianwen Xue. 2010. "Semantic role labeling." *Synthesis Lectures on Human Language Technologies* 3 (1): 1–103.

Parikh, Ankur P, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. "A decomposable attention model for natural language inference." *arXiv preprint arXiv:1606.01933.*

Perrault, C Raymond, and James F Allen. 1980. "A plan-based analysis of indirect speech acts." *Computational Linguistics* 6 (3-4): 167–182.

Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. "Improving language understanding by generative pre-training." *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf.*

Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. "Squad: 100,000+ questions for machine comprehension of text." *arXiv preprint arXiv:1606.05250.*

Ray, Oliver. 2009. "Nonmonotonic abductive inductive learning." *Journal of Applied Logic* 7 (3): 329–340.

Reddy, Siva, Danqi Chen, and Christopher D Manning. 2018. "Coqa: A conversational question answering challenge." *arXiv preprint arXiv:1808.07042.*

———. 2019. "Coqa: A conversational question answering challenge." *Transactions of the Association for Computational Linguistics* 7:249–266.

Reddy, Siva, Mirella Lapata, and Mark Steedman. 2014. "Large-scale semantic parsing without question-answer pairs." *Transactions of the Association for Computational Linguistics* 2:377–392.

Richardson, Matthew, Christopher JC Burges, and Erin Renshaw. 2013. "MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text." In *EMNLP,* 1:2.

Roy, Subhro, and Dan Roth. 2015. "Solving general arithmetic word problems." EMNLP.

———. 2016. "Unit dependency graph and its application to arithmetic word problem solving." *arXiv preprint arXiv:1612.00969.*

———. 2017. "Mapping to Declarative Knowledge for Word Problem Solving." *arXiv preprint arXiv:1712.09391.*

Roy, Subhro, Tim Vieira, and Dan Roth. 2015. "Reasoning about quantities in natural language." *Transactions of the Association for Computational Linguistics* 3:1–13.

Sakaguchi, Keisuke, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. "WINOGRANDE: An Adversarial Winograd Schema Challenge at Scale." *arXiv preprint arXiv:1907.10641.*

Sakama, Chiaki. 2005. "Induction from Answer Sets in Nonmonotonic Logic Programs." *ACM Trans. Comput. Logic* (New York, NY, USA) 6, no. 2 (April): 203–231. doi:10.1145/1055686.1055687.

Sakama, Chiaki, and Katsumi Inoue. 2009. "Brave induction: a logical framework for learning from incomplete information." *Machine Learning* 76, no. 1 (July): 3–35. doi:10.1007/s10994-009-5113-y.

Sap, Maarten, Ronan Le Bras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A Smith, and Yejin Choi. 2019. "ATOMIC: an atlas of machine commonsense for if-then reasoning." In *Proceedings of the AAAI Conference on Artificial Intelligence,* 33:3027–3035.

Sap, Maarten, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. "SocialIQA: Commonsense Reasoning about Social Interactions." *arXiv preprint arXiv:1904.09728.*

Schüller, Peter, and Mishal Kazmi. 2017. "Best-Effort Inductive Logic Programming via Fine-grained Cost-based Hypothesis Generation." *arXiv preprint arXiv:1707.02729.*

Seo, Minjoon, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. "Bidirectional attention flow for machine comprehension." *arXiv preprint arXiv:1611.01603.*

Seo, Minjoon, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2016. "Query-reduction networks for question answering." *arXiv preprint arXiv:1606.04582.*

Sharma, Arpit, Nguyen Ha Vo, Somak Aditya, and Chitta Baral. 2015. "Towards Addressing the Winograd Schema Challenge-Building and Using a Semantic Parser and a Knowledge Hunting Module." In *IJCAI.*

Shi, Shuming, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. "Automatically solving number word problems by semantic parsing and reasoning." In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Lisbon, Portugal.*

Simmons, Robert F. 1970. "Natural language question-answering systems: 1969." *Communications of the ACM* 13 (1): 15–30.

Stewart, Russell, and Stefano Ermon. 2017. "Label-free supervision of neural networks with physics and domain knowledge." In *AAAI,* 1:1–7. 1.

Sun, Kai, Dian Yu, Dong Yu, and Claire Cardie. 2018. "Improving Machine Reading Comprehension with General Reading Strategies." *CoRR* abs/1810.13441.

Tafjord, Oyvind, Peter Clark, Matt Gardner, Wen-tau Yih, and Ashish Sabharwal. 2018. "QuaRel: A Dataset and Models for Answering Questions about Qualitative Relationships." *arXiv preprint arXiv:1811.08048.*

———. 2019. "Quarel: A dataset and models for answering questions about qualitative relationships." In *Proceedings of the AAAI Conference on Artificial Intelligence,* 33:7063–7071.

Tandon, Niket, Bhavana Dalvi Mishra, Joel Grus, Wen-tau Yih, Antoine Bosselut, and Peter Clark. 2018. "Reasoning about actions and state changes by injecting commonsense knowledge." *arXiv preprint arXiv:1808.10012*.

Upadhyay, Shyam, Ming-Wei Chang, Kai-Wei Chang, and Wen-tau Yih. 2016. "Learning from explicit and implicit supervision jointly for algebra word problems." In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing,* 297–306.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. "Attention is all you need." In *Advances in Neural Information Processing Systems,* 5998–6008.

Wan, Li, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. 2013. "Regularization of neural networks using dropconnect." In *International Conference on Machine Learning,* 1058–1066.

Wang, Alex, Amapreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. "Glue: A multi-task benchmark and analysis platform for natural language understanding." *arXiv preprint arXiv:1804.07461*.

Wang, Lei, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018. "MathDQN: Solving Arithmetic Word Problems via Deep Reinforcement Learning."

Wang, Xiaoyan, Pavan Kapanipathi, Ryan Musa, Mo Yu, Kartik Talamadupula, Ibrahim Abdelaziz, Maria Chang, Achille Fokoue, Bassem Makni, Nicholas Mattei, et al. 2019. "Improving Natural Language Inference Using External Knowledge in the Science Questions Domain." In *Proceedings of the AAAI Conference on Artificial Intelligence,* 33:7208–7215.

Wang, Yan, Xiaojiang Liu, and Shuming Shi. 2017. "Deep Neural Solver for Math Word Problems." In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing,* 845–854.

Wang, Yi, Joohyung Lee, and Doo Soon Kim. 2017. "A Logic Based Approach to Answering Questions about Alternatives in DIY Domains." In *AAAI,* 4753–4759.

Wang, Yushi, Jonathan Berant, and Percy Liang. 2015. "Building a semantic parser overnight." In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers),* vol. 1.

Weld, Daniel S, and Johan De Kleer. 2013. *Readings in qualitative reasoning about physical systems.* Morgan Kaufmann.

Weston, Jason, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. 2015. "Towards AI-complete question answering: a set of prerequisite toy tasks." *arXiv preprint arXiv:1502.05698.*

Weston, Jason, Sumit Chopra, and Antoine Bordes. 2014. "Memory networks." *arXiv preprint arXiv:1410.3916.*

Wilensky, Robert. 1983. "Planning and understanding: A computational approach to human reasoning."

Wilensky, Robert, David N Chin, Marc Luria, James Martin, James Mayfield, and Dekai Wu. 2000. "The Berkeley UNIX consultant project." In *Intelligent Help Systems for UNIX,* 49–94. Springer.

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

Yang, An, Quan Wang, Jing Liu, Kai Liu, Yajuan Lyu, Hua Wu, Qiaoqiao She, and Sujian Li. 2019. "Enhancing Pre-Trained Language Representations with Rich Knowledge for Machine Reading Comprehension." In *Proceedings of the 57th Conference of the Association for Computational Linguistics,* 2346–2357.

Yang, Yi, Wen-tau Yih, and Christopher Meek. 2015. "Wikiqa: A challenge dataset for open-domain question answering." In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing,* 2013–2018.

Zelle, John M, and Raymond J Mooney. 1996. "Learning to parse database queries using inductive logic programming." In *Proceedings of the national conference on artificial intelligence,* 1050–1055.

Zellers, Rowan, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. "Swag: A large-scale adversarial dataset for grounded commonsense inference." *arXiv preprint arXiv:1808.05326.*

Zettlemoyer, Luke S, and Michael Collins. 2012. "Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars." *arXiv preprint arXiv:1207.1420.*

Zhang, Yuyu, Hanjun Dai, Kamil Toraman, and Le Song. 2018. "KGˆ 2: Learning to Reason Science Exam Questions with Contextual Knowledge Graph Embeddings." *arXiv preprint arXiv:1805.12393.*

Zhou, Lipu, Shuaixiang Dai, and Liwei Chen. 2015. "Learn to solve algebra word problems using quadratic programming." In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing,* 817–822.

APPENDIX A

PROOF OF THEOREM 1

Theorem 1
For any solution $\langle H_I, H_G, \triangle\rangle$ of $ILP^{DE}(B, M, \langle E_1, ..., E_n\rangle)$ there exists a solution $\langle H'_I, H'_G, \triangle'\rangle$ of $ILP^{DE}(B, M, \langle E_1, ..., E_{n-1}\rangle)$ and a generalization H''_G in $ILP^{DE}(B, M, E_n)$ such that, $H'_I \leq H_I \leq H'_G \cup H''_G$, when $H \cup B \cup O_i$ is *stratified* for any choice of $i \in \{1, ..., n\}$ and $H \in \{H_G, H'_G, H''_G\}$. Here, O_i is the *observation* from E_i.

A.0.0.0.1 Proof

Recall that $\Delta(B, M, E) = \{\triangle | \langle H_I, H_G, \triangle\rangle \in XHAIL(B, M, E)$ for some $H_I, H_G\}$. We further define,

$$\triangle(B, M, \langle E_1, ..., E_n\rangle) = \{(\triangle_1, \triangle_2, ..., \triangle_n) | \triangle_i \in \triangle(B, M, E_i), \forall i = 1..n\}$$
$$H_G(\triangle = (\triangle_1, \triangle_2, ..., \triangle_n)) = \cup_{i=1}^{n} H_G(\triangle_i)$$

Since H_I is a solution to $ILP^{DE}(B, M, \langle E_1, E_2, ..., E_{n-1}\rangle)$ and $H_I \cup B \cup O_i$ is assumed to be a stratified program, there is a unique set containing only ground instances of *modeh* literals (abducible predicates), $\triangle^* = (\triangle^*_1, \triangle^*_2, ..., \triangle^*_{n-1})$ in $\triangle(B, M, \langle E_1, ..., E_n\rangle)$ such that $\forall i \in 1, ..., n-1$,

i $B \cup O_i \cup H_I \vdash \triangle^*_i$,
ii $\nexists \triangle'_i . (\triangle'_i \in \triangle(B, M, E_i)) \wedge (B \cup O_i \cup H_I \vdash \triangle'_i) \wedge (\triangle^*_i \subset \triangle'_i)$.

Similarly, since H_I is a solution to $ILP^{DE}(B, M, E_n)$ there is a unique $\bar{\triangle}$ such that,

i $B \cup O_n \cup H_I \vdash \bar{\triangle}$,
ii $\nexists \triangle'_n . (\triangle'_n \in \triangle(B, M, E_n)) \wedge (B \cup O_n \cup H_I \vdash \bar{\triangle}) \wedge (\bar{\triangle} \subset \triangle'_n)$.

H_I is then bounded by $H_G(\triangle^*) \cup H_G(\bar{\triangle})$. If this is not the case then H_I has at least one rule whose body is not satisfied in any of the context provided by $B \cup O_i$, for all $i = 1, ..., n$. And hence H_I cannot be minimal. Now consider the set S containing all the minimal solution $\langle H'_I, H'_G, \triangle^*\rangle$ of $ILP^{DE}(B, M, \langle E_1, E_2, ..., E_{n-1}\rangle)$ that can be obtained from $\triangle^*$. Let H^*_I denote the set of all rules from H_I that are satisfied in at least one of the context $B \cup O_i \cup H_I$, for $i = 1...n-1$. Then, there must exist at least one $H'_I \in S$ such that $H'_I \leq H^*_I \leq H_I$. Otherwise, H^*_I is a minimal solution of $ILP^{DE}(B, M, \langle E_1, E_2, ..., E_{n-1}\rangle)$ that can be obtained from $\triangle^*$ but not in S. A contradiction. ■