

© 2017 by Subhro Roy. All rights reserved.

REASONING ABOUT QUANTITIES IN NATURAL LANGUAGE

BY

SUBHRO ROY

DISSERTATION

Submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy in Computer Science  
in the Graduate College of the  
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Doctoral Committee:

Professor Dan Roth, Chair

Professor Gerald DeJong

Associate Professor Julia Hockenmaier

Associate Professor Luke Zettlemoyer, University of Washington

# Abstract

Quantitative reasoning involves understanding the use of quantities and numeric relations in text, and reasoning with respect to them. It forms an essential part of everyday interaction. However, little work from the Natural Language Processing community has focused on quantitative reasoning. In this thesis, we investigate the challenges in performing automated quantitative reasoning over natural language text. We formulate several tasks to tackle some of the fundamental problems of quantitative reasoning, and address the problem of developing robust statistical methods for these tasks.

We show that standard NLP tools are not sufficient to obtain the abstraction needed for quantitative reasoning; the standard NLP pipeline needs to be extended in various ways. We propose several technical ideas for these extensions. We first look at the problem of detecting and normalizing quantities expressed in free form text, and show that correct detection and normalization can support several simple quantitative inferences. We then focus on numeric relation extraction from sentences, and show that several natural properties of language can be leveraged to effectively extract numeric relations from a sentence.

We finally investigate the problem of quantitative reasoning over multiple quantities mentioned across several sentences. We develop a decomposition strategy which allows reasoning over pairs of numbers to be combined effectively to perform global reasoning. We also look at the problem of effectively using math domain knowledge in quantitative reasoning. On this front, we first propose graph representations called “unit dependency graphs”, and show that these graph representations can be used to effectively incorporate dimensional analysis knowledge in quantitative reasoning. Next, we develop a general framework to incorporate any declarative knowledge into quantitative reasoning. This framework is used to incorporate several mathematical concepts into textual quantitative reasoning, leading to robust reasoning systems.

*To my mom.*

# Acknowledgments

I consider myself fortunate to have Dan as my research advisor. He took me on as a PhD student when I was a confused theory deserter with no clue about NLP. He had the most encouraging words about my progress, even when I felt frustrated with the lack of results. He always encouraged me to strive for better results, and understand the bigger picture of the research. He has taught me to not only be a better researcher, but also to be a kind and compassionate human being.

I would like to thank my committee members: Prof. Gerald DeJong, Prof. Julia Hockenmaier and Prof. Luke Zettlemoyer, from whom I got valuable feedback on my research. I would also like to thank all my internship mentors and collaborators, namely Ming-Wei Chang, Scott Yih, Hannaneh Hajishirzi, Rik Koncel-Kedziorski, Mark Hopkins, JD Chen; I have learnt a lot working with them.

Special thanks to my friends Shyam and John; I will miss our long afternoon walks discussing research among other things. I would like to thank Snigdha; her help was instrumental in finishing this thesis on time. I would also like to thank the past and present members of Cogcomp, namely Daniel, Stephen, Haoruo, Vinod, Rajhans, Kai-Wei, Gourab, Nitish, Colin, Shashank (Srivastava and Gupta), Michael, Christos, Parisa, Chen-Tse, Pavan, Yangqiu and Mark. They had gifted me an ideal stimulating environment for research. Also, special thanks to my friends Arka, Arun, Anish, Sangeetha and Mainak, who provided me a lot of support in troubled times.

Finally, I would like to thank my family – my mom, dad and brother. My parents left no stone unturned to provide me the best quality education, in spite of facing lots of hardships themselves to make it possible. This thesis could not have been completed without my family’s encouragement, support and love. I wish my mom was here today to see me graduate. Ma, words cannot describe how much I miss you.

# Table of Contents

<b>List of Tables</b> . . . . .	<b>viii</b>
<b>List of Figures</b> . . . . .	<b>ix</b>
<b>Chapter 1 Introduction</b> . . . . .	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Math Word Problems: An Abstraction for Quantitative Reasoning . . . . .	2
1.3 Challenges in Automated Quantitative Reasoning . . . . .	3
1.3.1 Variability of Natural Language Text . . . . .	3
1.3.2 Quantity Argument Detection . . . . .	4
1.3.3 Reasoning about Multiple Quantities . . . . .	5
1.3.4 Incorporating Domain Knowledge . . . . .	5
1.4 Contributions of the Thesis . . . . .	6
1.5 Overview of the Thesis . . . . .	7
<b>Chapter 2 Background</b> . . . . .	<b>9</b>
2.1 What is a Structure? . . . . .	9
2.2 Structured Prediction . . . . .	9
2.3 Inference in Structured Models . . . . .	10
2.4 Learning Structured Prediction Models . . . . .	11
2.4.1 Structural Support Vector Machines . . . . .	12
2.5 Discussion and Special Case . . . . .	12
2.6 Structured Prediction with Latent Variables . . . . .	13
<b>Chapter 3 Early Work in Quantitative Reasoning</b> . . . . .	<b>14</b>
3.1 Early Algebra and Arithmetic Solvers . . . . .	14
3.1.1 STUDENT Program . . . . .	14
3.1.2 Other Arithmetic Word Problem Solvers . . . . .	15
3.2 Solvers for other Domains . . . . .	17
3.3 Recent Advances in Statistical Solvers . . . . .	17
<b>Chapter 4 Quantity Entailment</b> . . . . .	<b>19</b>
4.1 Introduction . . . . .	19
4.2 Related Work . . . . .	20
4.3 Representing Quantities . . . . .	21
4.4 Extraction of Quantities . . . . .	21
4.4.1 Features . . . . .	23
4.4.2 Mapping Text Segments into QVR . . . . .	23
4.4.3 Extraction of Units . . . . .	24
4.5 Quantity Entailment . . . . .	25
4.5.1 Reasoning Framework . . . . .	26

4.5.2	Scope of QE Inference . . . . .	29
4.6	Experimental Study . . . . .	30
4.6.1	Datasets . . . . .	30
4.6.2	Quantity Segmentation . . . . .	31
4.6.3	Quantity Entailment . . . . .	32
4.6.4	Currency Range Search . . . . .	33
4.6.5	Qualitative Analysis . . . . .	34
4.7	Conclusion . . . . .	34
<b>Chapter 5</b>	<b>Equation Parsing . . . . .</b>	<b>35</b>
5.1	Introduction . . . . .	35
5.2	Related Work . . . . .	37
5.3	The Equation Parsing Task . . . . .	37
5.3.1	Equation Parse Representation . . . . .	38
5.4	Projectivity . . . . .	40
5.5	Predicting Equation Parse . . . . .	41
5.5.1	Predicting Quantity Trigger List . . . . .	41
5.5.2	Predicting Variable Trigger List . . . . .	42
5.5.3	Predicting Equation Tree . . . . .	44
5.5.4	Alternatives . . . . .	45
5.6	Experimental Results . . . . .	45
5.6.1	Dataset . . . . .	46
5.6.2	Equation Parsing Modules . . . . .	47
5.6.3	Equation Parsing Results . . . . .	48
5.6.4	Error Analysis . . . . .	49
5.7	Conclusion . . . . .	49
<b>Chapter 6</b>	<b>Arithmetic Word Problems . . . . .</b>	<b>50</b>
6.1	Introduction . . . . .	50
6.2	Related Work . . . . .	52
6.3	Expression Tree and Problem Decomposition . . . . .	53
6.4	Mapping Problems to Expression Trees . . . . .	58
6.4.1	Global Inference for Expression Trees . . . . .	58
6.4.2	Quantity Schema . . . . .	60
6.4.3	Relevance Classifier . . . . .	61
6.4.4	LCA Operation Classifier . . . . .	62
6.5	Experimental Results . . . . .	63
6.5.1	Datasets . . . . .	63
6.5.2	Relevance Classifier . . . . .	65
6.5.3	LCA Operation Classifier . . . . .	65
6.5.4	Global Inference Module . . . . .	66
6.5.5	Discussion . . . . .	67
6.6	Conclusion . . . . .	67
<b>Chapter 7</b>	<b>Unit Dependency Graphs . . . . .</b>	<b>69</b>
7.1	Introduction . . . . .	69
7.2	Unit Dependency Graph . . . . .	70
7.3	Learning to Predict UDGs . . . . .	72
7.3.1	Vertex Classifier . . . . .	72
7.3.2	Edge Classifier . . . . .	73
7.3.3	Constrained Inference . . . . .	73
7.4	Joint Inference With An Arithmetic Solver . . . . .	74

7.4.1	Monotonic Expression Tree . . . . .	74
7.4.2	Arithmetic Word Problem Solver . . . . .	74
7.4.3	Joint Inference . . . . .	75
7.4.4	Consistent Rate Unit Graphs . . . . .	76
7.5	Experiments . . . . .	78
7.5.1	Dataset . . . . .	78
7.5.2	Data Acquisition . . . . .	79
7.5.3	UDG Prediction . . . . .	80
7.5.4	Solving Arithmetic Word Problems . . . . .	81
7.5.5	Discussion . . . . .	82
7.6	Conclusion . . . . .	83
<b>Chapter 8 Mapping to Declarative Knowledge . . . . .</b>		<b>84</b>
8.1	Introduction . . . . .	84
8.2	Knowledge Representation . . . . .	85
8.2.1	Transfer . . . . .	86
8.2.2	Dimensional Analysis . . . . .	87
8.2.3	Explicit Math . . . . .	88
8.2.4	Part-Whole Relation . . . . .	88
8.3	Mapping of Word Problems to Declarative Knowledge . . . . .	89
8.3.1	Scoring Answer Derivations . . . . .	90
8.3.2	Learning . . . . .	91
8.3.3	Inference . . . . .	92
8.4	Model and Implementation Details . . . . .	92
8.4.1	Supervision . . . . .	92
8.4.2	Features . . . . .	93
8.5	Experiments . . . . .	94
8.5.1	Results on Existing Dataset . . . . .	94
8.5.2	New Dataset Creation . . . . .	97
8.5.3	Generalization from Biased Dataset . . . . .	97
8.5.4	Results on the New Dataset . . . . .	98
8.5.5	Analysis . . . . .	98
8.6	Related Work . . . . .	99
8.7	Conclusion . . . . .	100
<b>Chapter 9 Conclusion and Future Work . . . . .</b>		<b>102</b>
9.1	Summary . . . . .	102
9.2	Limitations . . . . .	103
9.3	Future Directions . . . . .	103
9.3.1	Commonsense Quantitative Reasoning . . . . .	104
9.3.2	Unified Framework for Solvers . . . . .	104
9.3.3	Automated Math Tutoring System . . . . .	104
<b>References . . . . .</b>		<b>105</b>
<b>Appendix A</b>		
<b>Lexicon for Equation Parsing . . . . .</b>		<b>110</b>
<b>Appendix B</b>		
<b>Declarative Rules for Arithmetic Word Problems . . . . .</b>		<b>112</b>



# List of Tables

3.1	Schemas used in WORDPRO . . . . .	16
4.1	10-fold cross-validation results of segmentation accuracy and time required for segmentation, the columns for runtime have been normalized and expressed as ratios . . . . .	32
4.2	Results of QE; Adding Semantics(+SEM) consistently improves performance; Only 43.3% of entailing quantities can be recovered by simple string matching . . . . .	33
4.3	Micro-averaged accuracy in detecting monetary mentions . . . . .	33
5.1	Input and output for Equation Parsing . . . . .	38
5.2	Summary of notations used in this chapter . . . . .	39
5.3	Statistics of dataset . . . . .	46
5.4	Performance of Quantity Trigger List Prediction . . . . .	47
5.5	Performance of Variable Trigger List Prediction . . . . .	47
5.6	Performance of Equation Tree Prediction . . . . .	48
5.7	Performance on equation parsing . . . . .	48
6.1	Performance of LCA Operation classifier on the datasets AI2, IL and CC. . . . .	63
6.2	Performance of Relevance classifier on the datasets AI2 and IL. . . . .	65
6.3	Accuracy in correctly solving arithmetic problems. First four rows represent various configurations of our system. We achieve state of the art results in both AI2 and IL datasets. . . . .	66
7.1	Units of rate quantities . . . . .	70
7.2	Performance of system components for predicting vertex and edge labels for unit dependency graphs . . . . .	80
7.3	Performance in predicting UDGs . . . . .	81
7.4	Performance in solving arithmetic word problems . . . . .	81
7.5	Examples of problems which UNITDEP gets correct, but LCA++ does not. . . . .	83
8.1	Two examples of arithmetic word problems, and derivation of the answer. For each combination, first a knowledge type is chosen, and then a declarative rule from that knowledge type is chosen to infer the operation. . . . .	89
8.2	Accuracy in solving arithmetic word problems. All columns except the last report 5-fold cross validation results. * indicates statistically significant improvement ( $p = 0.05$ ) over second highest score in the column. . . . .	96
8.3	Pairs of pertubed problems, along with the systems which get them correct . . . . .	96
8.4	Examples which KNOWLEDGE gets correct, but UNITDEP does not. . . . .	99
8.5	Examples of errors made by KNOWLEDGE . . . . .	99

# List of Figures

5.1	A sentence with its trigger list and equation tree. $-_r$ indicates subtraction with order $rl$ . . .	38
6.1	An arithmetic word problem, solution expression and the corresponding expression tree . . .	54
6.2	Two different expression trees for the numeric expression $(3 \times 5) + 7 - 8 - 9$ . The right one is monotonic, whereas the left one is not. . . . .	55
7.1	An arithmetic word problem, its UDG, and a tree representation of the solution $(66 - 10)/8$ . Several dependencies exist between the UDG and the final solution of a problem. Here, “66” and “10” are connected via SAME UNIT edge, hence they can be added or subtracted, “8” is connected by DEN UNIT to the question, indicating that some expression will be divided by “8” to get the answer’s unit. . . . .	71
8.1	An example arithmetic word problem and its solution, along with the type of domain knowledge required to generate each operation of the solution . . . . .	85
8.2	Annotations in our dataset. Number List refers to the numbers detected in the problem. The subscripts in the solution indicate the position of the numbers in the number list. . . . .	93

# Chapter 1

## Introduction

### 1.1 Motivation

Numbers are an integral part of natural language. We use numbers extensively to communicate how hot the weather is, how well our favorite team played in the last game, etc. Newspaper articles regularly report statistics to present an objective assessment of a situation, ranging from the number of people killed in a bomb blast, to the amount of money embezzled by a politician. To understand such articles, one often needs to figure out several properties of the numbers used in them, and how these numbers interact with the rest of the story. Consider the following sentence taken from a news article in the Chicago Tribune [Chi, 2017]:

*Emanuel spent \$13.6 million from July until the Feb. 24 election and spent an additional \$6.3 million in the following five weeks.*

In order to understand the sentence, one has to understand the following for each numeric mention:

1. **Unit:** Indicates which object is being referred to. One has to understand which numbers indicate monetary amounts, and which ones indicate date and time, etc.
2. **Associated Verb:** Indicates the effect of the number on the world, whether it is indicating the state of an economy, or amount of money in a transaction, etc. In this case, both “13.6 million” and “6.3 million” have the verb “spend” associated with it, indicating both are expenditures made by the campaign.
3. **Arguments:** Indicates the entities interacting with the number. For example, for a monetary transaction, you want to know the recipient and the donor. In the example above, knowing that both “13.6 million” and “6.3 million” are expenditures made by Emanuel, enables the reader to understand that in total, Emanuel spent 19.9 million dollars.

There is also a need to understand interactions of several numbers mentioned across several sentence in text. Consider the following excerpt from a recent news article [Syr, 2017].

*Local sources told The New Arab that a car bomb had targeted a Free Syria Army checkpoint at the refugee camp on the Syrian border with Jordan, killing two FSA fighters and injuring several others . . . Jordanian border police responded by opening fire on the attacking vehicle killing a child, according to the sources. A car bomb in January killed at least seven people at the same camp. Around 85,000 people live in the Rukban camp . . . An earlier car bomb attack near the camp, in the hinterland border area on June 21 last year that resulted in the deaths of six Jordanian military personnel.*

In order to report the total number of deaths reported in the article, one has to analyze how the different numbers in the story interact. One has to understand that the number of people living in the camp (“85,000”) is irrelevant for calculating the number of casualties. All numbers stating the number of victims need to be identified from all sentences in the article, like “two FSA fighters”, “a child”, “at least seven people”, etc. Finally all such numbers need to be summed up.

All these challenges come under the umbrella of quantitative reasoning. In spite of the difficulties involved, human beings are adept at performing quantitative reasoning to understand natural language text. However, this is not true for automated language understanding systems. Relatively little work in Natural Language Processing has analyzed the use of quantities in text. Even in areas where we have relatively mature solutions, like information retrieval, we fail to deal with quantities; for example, one cannot search the financial media for “transactions in the 1-2 million pounds range.”

In this dissertation, we investigate the challenges in performing automated quantitative reasoning. We formulate several tasks that we believe to be fundamental to addressing these challenges, and address the problem of developing robust natural language understanding methods for these tasks, using statistical machine learning.

## 1.2 Math Word Problems: An Abstraction for Quantitative Reasoning

In this thesis we suggest that the vast majority of quantitative reasoning problems that occur frequently in text can be viewed, as math word problems that are solved by elementary and middle schools kids. Lets look at a simple math word problem which students solve in elementary school.

*Ryan has 72 marbles and 17 blocks. If he shares the marbles among 9 friends, how many marbles does each friend get?*

Even in such a simple quantitative reasoning problem, there are several challenges involving multiple interactions among the numbers in the text. One needs to understand that the question asks for the “marbles” each “friend” gets, so the number of “blocks” should have no effect on the answer. One needs to understand the impact of the associated verb “shares”, and that division is needed to calculate the correct answer. Note that this bears clear resemblance with the challenges of quantitative reasoning discussed earlier.

There are also several advantages in focusing on math word problems. The language in these problems is relatively simple, and requires understanding of a few related concepts. This helps us to focus on quantitative reasoning, without addressing overly complicated NLP issues. These problems are readily available from textbooks and tutoring websites. As a result, developing benchmark datasets is relatively easy.

Several pieces of work described in this thesis, particularly in the later chapters (Chapter 6, 7 and 8), have been evaluated on the ability to automatically solve math word problems. However, the solutions are applicable for general quantitative reasoning problems.

## 1.3 Challenges in Automated Quantitative Reasoning

Building automated systems for quantitative reasoning involves developing an information extraction system which detects key properties of a quantity, as well as developing a reasoning system which captures the interaction of several numbers in text. Developing such a system is challenging because of several reasons which we explain and illustrate in the following subsections.

### 1.3.1 Variability of Natural Language Text

An initial step in performing quantitative reasoning is to identify quantities mentioned in natural language text, to understand what they mean and what they refer to. We address this by identifying the quantity and its unit, and representing them in a canonical form. This step is challenging because the same quantity can be expressed in several forms, a problem we refer to as *variability*. For example, the quantity “\$13.6 million” can be written as “USD 13.6 million”, “13600000 dollars”, “13600000\$”. In addition, the same amount of money can be expressed in some other currency unit (like euros). This will require some unit conversion to understand that it refers to the same amount of money. Another challenge is the effect of several modifiers

on quantities. Consider the difference between “more than 5 hours”, “5 hours”, and “around 5 hours”. Although all three phrases are similar in terms of lexical overlap, they represent different quantity values.

In order to capture the difficulties in detecting and normalizing numbers, and their impact on quantitative reasoning, we formulate a new task called *Quantity Entailment (QE)*. This task draws its motivation from the general Recognizing Textual Entailment (RTE) problem [Dagan et al., 2013]. Given two sentences  $T$  and  $H$ , the task of QE is to determine whether a given quantity in  $H$  can be inferred from a given text snippet  $T$ . An example is shown below.

*T: A bomb in a Hebrew University cafeteria killed five Americans and four Israelis.*

*H: A bombing at Hebrew University in Jerusalem killed nine people, including five Americans.*

For this example, the output of QE is that the phrase “nine people” can be implied by the phrase “fives Americans and four Israelis”. We investigate this task in Chapter 4, and discuss effective solutions to quantity detection and normalization, and ultimately show its effectiveness in determining quantity entailment.

### 1.3.2 Quantity Argument Detection

To reason with respect to numbers, it is important to extract all information related to it from the sentence. For example, if a number is indicating a transaction, one needs to know the donor and the recipient; if a number represents a relation between two entities, one needs to know what those entities are. Consider the following example:

*Emanuel’s campaign contributions total thrice those of his opponents put together.*

To understand the sentence above, its important to know that “thrice” denotes a relation between “Emanuel’s campaign contributions” and “the contributions of his opponents”. Often, not all the entities associated with a relation are explicitly mentioned in the sentence; they have to be inferred from the context. Consider the following sentence [Gun, 2015].

*On an average day this year, 36 Americans were killed by guns.*

Here, one needs to use the understanding of “average”, and infer that the relation that is described involves the number of gun violence deaths each day of the year.

In order to capture these challenges, we formulate the task of *equation parsing*. The task takes as input a sentence representing a mathematical relation among at most two entities. The output of equation parsing is an equation which represents the relation expressed in the sentence. In addition to the equation, the output also provides a grounding for the variables in the equation, that is, it provides a mapping of the variables in the equation to text snippets indicating what the variables stand for. We discuss this problem in Chapter 5.

### 1.3.3 Reasoning about Multiple Quantities

Reasoning over multiple quantities often require correct hierarchical composition of numbers with mathematical operations, as in the case of the following math word problem:

*Isabel picked 66 flowers for her friends wedding. She was making bouquets with 8 flowers in each one. If 10 of the flowers wilted before the wedding, how many bouquets could she still make?*

In order to answer this question, 66 needs to be subtracted from 10, and only then the result needs to be divided by 8. First, we have to decide the order in which the numbers need to be composed, and second, we have to decide the math operation for each combination. This kind of hierarchical composition poses a challenge for quantitative reasoning.

To address this, we introduce novel decomposition strategies in Chapters 6 and 8. Our decomposition strategies allow reducing a math word problem with any number of quantities, into simple quantitative reasoning problems over at most two quantities each. We utilize this decomposition to develop a system for automatically solving math word problems like the example above, and achieve state of the art results.

### 1.3.4 Incorporating Domain Knowledge

Domain knowledge often significantly influences the outcome of quantitative reasoning. A key reason humans do very well in quantitative reasoning is that they can easily use their understanding of the world and background knowledge to make inferences. For example, in the math word problem in subsection 1.3.3, a child can easily infer that “wilt” should result in subtraction, since no one uses wilted flowers in bouquets. Also, since the unit of “8” is “flowers per bouquet”, it should be used in a division operation. Indeed, understanding of verb interaction and rate relationships seem essential to solving the problem. All these indicate that there is a need to effectively integrate domain knowledge into any quantitative reasoning system, which will be the focus of the last part of the thesis.

## 1.4 Contributions of the Thesis

In this thesis, we investigate each of the challenges mentioned above, for automated quantitative reasoning.

In particular, we make the following claim:

*Current NLP pipeline is not sufficient to extract an abstraction of text required to perform quantitative reasoning. However quantitative reasoning over text can be facilitated by identifying quantities, units, and integrating domain knowledge about mathematical relations into machine learning methods. To perform quantitative reasoning over multiple quantities, we can decompose the problem into simpler components, where each component is a simpler quantitative reasoning problem over at most two quantities.*

The primary contributions of the thesis are summarized below:

1. We propose a statistical approach to quantity detection and normalization based on numbers, units and modifiers. We show that this is an effective approach to support simple quantitative reasoning.
2. We exemplify the drawbacks of current syntactic and semantic parsers to extract the abstraction need to support quantitative reasoning.
3. We exploit natural properties of equations to reduce search space, and propose a pipeline architecture to efficiently and effectively predict mathematical relations expressed in a single sentence.
4. We propose a novel decomposition procedure to support quantitative reasoning over multiple mentions of numbers. We show that this decomposition allows simple reasoning over pairs of quantities to be composed effectively, to perform quantitative reasoning over multiple numbers across several sentences.
5. We introduce a structure called “unit dependency graphs” to represent the relationships between the units of various numbers in text, and the question you need to answer using that text. We show that unit dependency graphs capture the domain knowledge about unit compatibility and dimensional analysis, and can be effectively used to incorporate such domain knowledge in quantitative reasoning systems.
6. We propose a framework to integrate any form of declarative knowledge into word problem solving. We show that this declarative knowledge based method learns better models from limited data, as well as, obtains the right abstraction even in the presence of biased data.



## 1.5 Overview of the Thesis

This section provides a guide for the rest of the thesis. The thesis can be broadly categorized into four logical segments.

### Part I: Background

The first part of this thesis surveys background work in the areas of quantitative reasoning, structured learning and inference. These chapters do not provide exhaustive surveys of these research directions and we will only go over relevant material here and include additional pointers as necessary. A reader familiar with machine learning and structured learning paradigms, can only skim these parts, and focus only on the background work in quantitative reasoning. Chapter 2 is devoted to this part.

### Part II: Quantity Extraction, Normalization and Equation Parsing

This part of the thesis discusses work related to quantities at a mention level or sentence level. We discuss effective approaches to extract and normalize quantities into a standard form. We also show how we can leverage certain natural properties of a sentence to extract mathematical relations between entities in the sentence. The part constitutes two chapters.

- Chapter 4 discusses our approach to quantity extraction and normalization, and its application to textual entailment problems.
- Chapter 5 describes methods to analyze mathematical relations described within a single sentence, as well as, extract the relevant entities for the relation.

### Part III: Solving Arithmetic Word Problems

This part focuses on quantitative reasoning problems which span across multiple sentences. We found that math word problems form a natural abstraction to the challenges we usually encounter in across sentence quantitative reasoning. As a result, this part will focus on building robust solvers for math word problems. We however do not look at the entire range of word problems, which might require a wide range of background knowledge. We restrict our focus to arithmetic word problems; these are story problems which students solve in elementary school. The answers to these problems can be usually found by combining the numbers in the problem text by simple basic operations. This part constitutes three chapters:

- Chapter 6 describes a decomposition strategy which is key in applying quantitative reasoning over multiple numbers mentioned across several sentences. We show how this decomposition can be used to develop an end to end arithmetic word problem solver.
- Chapter 7 introduces a structure called “unit dependency graphs” to capture the relations among the units of different numbers mentioned in text. We show how these graphs can be used to incorporate domain knowledge about rate relationships, unit compatibility and dimensional analysis to word problem solving and to general quantitative reasoning.
- Chapter 8 describes a framework to integrate any form of declarative knowledge into a word problem solver. We use the framework to integrate four common forms of domain knowledge for arithmetic word problems. We show that the declarative knowledge helps our system learn more effectively in the presence of limited data.

## **Part IV: Conclusion**

The final part of the thesis offers concluding remarks. It summarizes the contributions of this dissertation and identifies directions for future research that can be built on top of this work. This can be found in Chapter 9.

# Chapter 2

## Background

Statistical machine learning approaches are widely used in many areas of natural language of processing, in particular, in predicting semantic annotations and linguistic structures. In this chapter, we cover a few basic machine learning concepts which will be used in the rest of the thesis. We mainly focus on structured prediction, since this is the paradigm most widely used in this thesis. We introduce the problem of structured prediction via the example of part of speech tagging, and outline common approaches to solve this problem.

### 2.1 What is a Structure?

The concept of a structure is an important one in computational linguistics and machine learning. Burton-Roberts (1997) informally defines a structure as a complex object that is divisible into component parts, each of which can belong to different categories, have specific functions in the complete object and are arranged in a specifiable way. We will use the task of part-of-speech tagging as a running example in this section. We will use the following sentence as an illustrative example:

*I have twenty dollars in my pocket .*

The goal of part of speech tagging is to label each word in the sentence by one of 45 part-of-speech tags. Any valid output for the sentence above is a set of 8 part-of-speech tags, one for each word in the sentence. This set of 8 part-of-speech tags can be referred as a **structure**.

### 2.2 Structured Prediction

Let us now formulate the part of speech tagging problem as a structured prediction problem. We will denote the input sentence to the problem as  $\mathbf{x}$ , and the output structure is denoted as  $yvec$ . In this case, the output structure  $\mathbf{y}$  can be decomposed into a set of  $m$  variables  $y_1, y_2, \dots, y_k$ , where  $m$  is the number of words in the

input sentence  $\mathbf{x}$ . Each of the  $y_i$  variables denotes the part of speech tag for the  $i$ -th word of input sentence  $\mathbf{x}$ . Therefore, each  $y_i$  can be assigned one of the 45 part-of-speech tags. Finally we define  $\mathcal{X}$  as the space of all possible inputs to a problem. Here, it will denote the space of all sentences. We also define  $\mathcal{Y}$  as the space of all possible outputs. In our example, this will denote the set of all sequences of part of speech tags.

The structured prediction problem is to learn a mapping function  $f(\mathbf{x}; \mathbf{w})$  from an input space  $\mathcal{X}$  to an output space  $\mathcal{Y}$ . This mapping function is parameterized with weights  $\mathbf{w}$  of task relevant features  $\phi(\mathbf{x}, \mathbf{y})$ . Note that the feature function is defined over both the input  $\mathbf{x}$  and the entire output structure  $\mathbf{y}$ . This definition of a feature function allows us to design features to express the dependency between the local output variables for instances. In general, designing feature templates requires domain knowledge. For example, in our running example, it is common to use words in  $\mathbf{x}$  conjuncted with the corresponding part of speech tag in  $\mathbf{y}$  as features. We use the weight vector  $\mathbf{w}$  and the feature function  $\phi(\mathbf{x}, \mathbf{y})$ , to define a scoring function for output structures. Often it takes the following linear form:

$$S(\mathbf{y}; \mathbf{w}, \mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}, \mathbf{y})$$

We want this scoring function to assign a higher score to the correct output structure, and assign a lower score to all other structures.

## 2.3 Inference in Structured Models

Suppose we have a weight vector  $\mathbf{w}$ , such that scoring function  $S(\mathbf{y}; \mathbf{w}, \mathbf{x})$  scores the correct output structure higher than all other structures. How will we use such a scoring function to predict an output structure for an input  $\mathbf{x}$ ? This involves searching for the best structure according to the scoring function; this is referred to as the inference problem in structured prediction. Mathematically, this seen as computing the following.

$$\mathbf{y}^* = \arg \max_{\mathbf{y} \in \mathcal{Y}} \mathbf{w}^T \phi(\mathbf{x}, \mathbf{y})$$

where  $\mathcal{Y}$  is the space of feasible output structures.

Note that the part of  $\mathcal{Y}$  you need to search over, is usually very large. Consider the case of our part of speech tagging example. For an input sentence of  $m$  words, the number of possible tag sequence is  $(45)^m$ , since each word can have one of the 45 tags. As a result, naively enumerating all output sequences and checking their score, is not a feasible solution.

There are three options to solve the inference problem in a reasonably efficient way. First, you can transform the problem to an Integer Linear Program (ILP), and then use an off the shelf ILP solver to solve the problem. This can still become intractable if the ILP contains a lot of variables. Another option is to make independence assumptions between different parts of the output structure. An example of this in our tagging task can be to assume that the part of speech of the  $i$  th word is dependent only on the  $i$  th word and the part of speech tag of the  $(i - 1)$  th word. These assumptions are often valid for the problem, and lead to efficient inference algorithms. However, for the problems we will address in this thesis, independence assumptions usually do not hold. Hence, we opt for the third option, which is to perform approximate inference using **beam search**.

The main idea behind beam search is to enumerate all possible assignments to the first output variable, and score them. Keep only the top  $k$  assignments and discard the rest. Next for each assignment of the first variable, enumerate all possible assignments for the second output variable, and again, keep the  $k$  highest scoring partial assignments, and discard the rest, and so on. Here  $k$  is often referred to as the beam size.

Lets see how we can perform beam search for the part of speech tagging example. We will assign  $y_1$  (the output variable for the first word) all the 45 tags, and score each assignment. Scores of these partial assignments can be computed by using only the features that are dependent on  $y_1$ . We keep the top  $k$ , and move on to  $y_2$ , and so on.

Although beam search is an approximate procedure, in practice, it is very effective. Also, the beam size is a parameter, which can be chosen to be a small value if the output space for the problem is very large.

## 2.4 Learning Structured Prediction Models

Finally, we discuss how to obtain a good scoring function for output structures. For this problem, assume we have access to a set of  $n$  labeled training examples  $\mathcal{D} = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^n$ . The learning problem is to find a weight vector  $\mathbf{w}$ , such that it scores the target output structure higher than other structure. This means that you want the value of  $\mathbf{w}^T \phi(\mathbf{x}^i, \mathbf{y}^i)$  to be higher than  $\mathbf{w}^T \phi(\mathbf{x}^i, \mathbf{y})$ , where  $\mathbf{y}$  is any other feasible output structure. There are several learning algorithms which can be used to obtain such a weight vector. In this thesis, we mostly use structural Support Vector Machines; we introduce them in the following subsection.

### 2.4.1 Structural Support Vector Machines

Structured SVMs are a generalization of the binary SVM algorithm [Cortes and Vapnik, 1995] to structured outputs [Tsochantaridis et al., 2005]. The goal of the learning is to solve the following optimization problem:

$$\min_{\mathbf{w}} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_i L(\mathbf{x}^i, \mathbf{y}^i, \mathbf{w})$$

where  $L(\mathbf{x}^i, \mathbf{y}^i, \mathbf{w})$  is the loss function for the structured prediction. The loss function is written as

$$L(\mathbf{x}^i, \mathbf{y}^i, \mathbf{w}) = l(\max_{\mathbf{y} \in \mathcal{Y}} \Delta(\mathbf{y}^i, \mathbf{y}) - w^T \phi(\mathbf{x}^i, \mathbf{y}^i) + w^T \phi(\mathbf{x}^i, \mathbf{y}))$$

Here  $\Delta$  signifies the quality of the structure  $\mathbf{y}$  with respect to the annotated structure  $\mathbf{y}^i$ . The function  $l(\cdot)$  can be instantiated by loss functions like the hinge loss, with  $l(x) = \max(0, x)$ .

Different algorithms have been proposed in the literature to solve the optimization problem [Joachims et al., 2009, Shalev-Shwartz et al., 2007, Chang et al., 2010]. In this thesis, we use the Illinois-SL package [Chang et al., 2015] with the dual-coordinate descent based algorithms.

## 2.5 Discussion and Special Case

For learning and inference with structured models, we need to specify the following:

1. The feature representation for a structure  $\mathbf{y}$  for an input  $\mathbf{x}$ , denoted as  $\phi(\mathbf{x}, \mathbf{y})$ .
2. An inference algorithm that finds the best structure for an input  $\mathbf{x}$  using a weight vector  $\mathbf{w}$ .
3. A learning algorithm.

Note that both binary and multiclass classification problems are special cases of structured prediction. As a result, a structured predictor can be used for binary and multiclass classification. Here we describe how can we convert a multiclass classification problem to a structured problem. Lets assume a multiclass classification problem of  $K$  classes. Usually feature functions for multiclass problems are defined only on the input  $\mathbf{x}$ ; lets denote it as  $\phi(\mathbf{x})$ . To convert this to a structured problem, we consider the output space  $\mathcal{Y}$  to be the set of class labels. We also define the feature function  $\phi(\mathbf{x}, \mathbf{y})$  to be the set of features  $\phi(\mathbf{x})$  conjoined with the label  $\mathbf{y}$ . With these definitions, we can now use the standard structured learning algorithm to do multiclass classification. In this thesis, we follow this procedure to perform multiclass classification with Illinois-SL.

## 2.6 Structured Prediction with Latent Variables

Until now, our structured prediction framework has only two types of variables – input variables  $\mathbf{x}$  and output variables  $\mathbf{y}$ . Both these variables are observed, that is, they are available as part of our training data. However, often times, important modeling information is not provided as part of the training data. For example, for the task of machine translation from English to French, training data comprises English sentences paired with corresponding translated French sentences. There are several intermediate variables which allow the translation of an English sentence to French, like the ones indicating word level or phrase level alignments. However these alignment variables are not provided as part of the training data. Even though these variables are not provided, it is important to model them, in order to capture the process of translation. In such cases, they are modeled as latent variables  $\mathbf{h} \in \mathcal{H}$ , where  $\mathcal{H}$  is the set of feasible assignments to the hidden variables.

In order to extend the structured prediction formulation to the case of hidden variables, we redefine the feature function to also use the hidden variables (written as  $\phi(\mathbf{x}, \mathbf{h}, \mathbf{y})$ ). The inference problem is now to compute

$$(\mathbf{h}^*, \mathbf{y}^*) = \arg \max_{(\mathbf{h}, \mathbf{y}) \in \mathcal{H} \times \mathcal{Y}} \mathbf{w}^T \phi(\mathbf{x}, \mathbf{h}, \mathbf{y})$$

This part remains similar to the earlier case; we simply have to think of the output structure as a tuple comprising both the output and hidden variables.

For learning, we again assume access to  $n$  labeled examples  $\mathcal{D} = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^n$ . The learning problem is to find a weight vector  $\mathbf{w}$ , such that it scores the target output structure higher than other structures. The score of a structure  $\mathbf{y}$  is now given as follows:

$$\max_{\mathbf{h} \in \mathcal{H}} \mathbf{w}^T \phi(\mathbf{x}, \mathbf{h}, \mathbf{y})$$

In this thesis, we mostly use structural SVM with latent variables [Yu and Joachims, 2009]. This involves solving the following optimization problem:

$$\min_{\mathbf{w}} \left[ \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_i \max_{(\hat{\mathbf{h}}, \hat{\mathbf{y}}) \in \mathcal{H} \times \mathcal{Y}} [\mathbf{w}^T \phi(\mathbf{x}^i, \hat{\mathbf{h}}, \hat{\mathbf{y}}) + \Delta(\mathbf{y}^i, \hat{\mathbf{h}}, \hat{\mathbf{y}})] \right] - \left[ C \sum_i \max_{\mathbf{h} \in \mathcal{H}} \mathbf{w}^T \phi(\mathbf{x}^i, \mathbf{h}, \mathbf{y}^i) \right]$$

where  $\Delta(\cdot)$  is again a loss function indicating the dissimilarity between  $\hat{\mathbf{y}}$  and  $\mathbf{y}^i$ . See [Yu and Joachims, 2009] for details of the algorithm to solve the above optimization.

## Chapter 3

# Early Work in Quantitative Reasoning

Work on quantitative reasoning problems in NLP dates back to the 1960's. Over the years, several efforts have been made to automatically solve different types of quantitative problems described in natural language text. Most of these systems were essentially rule based, and imposed strong restrictions on the input text. Only recently, statistical methods have been used in this context, and the focus has been on achieving robustness in handling a wide range of natural language text. In this chapter, we give an overview of these early systems, starting from the 1960's to the present day.

### 3.1 Early Algebra and Arithmetic Solvers

Several efforts were made to automatically solve arithmetic and algebra word problems. We describe a few of them in the following subsections.

#### 3.1.1 STUDENT Program

The STUDENT program written by [Bobrow, 1964] is the first system that tried to solve algebraic problems stated in natural language. It accepts inputs in a restricted set of English language and tries to solve the algebraic problem expressed in it. An example input to the STUDENT system is

*If the number of customer Tom gets is twice the square of 20% of the number of advertisement he runs, and the number of advertisements he runs is 45, what is the number of customer Tom gets ?*

The system parses individual sentences to extract mathematical relations among variables. They identify variables by several keywords, and consider the words around these keywords as identifiers for the variables. For example, “the number of customer Tom gets” is a variable where the key word is “number”. They cannot handle coreference, hence multiple usage of the same variable will require the exact same phrase to



be mentioned. In the above example, both the variable phrases “the number of customer Tom gets” and “the number of advertisements he runs” are repeated verbatim when the same variable is used again.

Applying a set of rules recursively, complex sentence constructs are simplified into simpler forms free of conjunctions. Certain substitutions (e.g. “2 times” for “twice”), and truncations (e.g. “square of” to “square”) etc. are done until only the phrases representing variables remain, and are related with basic operators (like plus, times, percent etc.) to form a set of equations representing the problem. After such transformation the above problem becomes:

(EQUAL X00001 (NUMBER OF CUSTOMER TOM GETS))
(EQUAL (NUMBER OF ADVERTISEMENTS HE RUNS) 45)
(EQUAL (NUMBER OF CUSTOMERS TOM GETS) (TIMES 2 (EXPT (TIMES 0.2 (NUMBER OF ADVERTISEMENTS HE RUNS)) 2)))

Finally standard techniques for solving system of equations are used to get the answer.

### 3.1.2 Other Arithmetic Word Problem Solvers

Another system called **WORDPRO** was developed by [Fletcher, 1985] for solving arithmetic word problems. WORDPRO was the first to introduce the concept of set “schema” [Marshall, 1995] as the basis of construction of problem model. Schemas can be thought as categories for the arithmetic word problem; each schema also has several attributes associated with it. For example, a “change” schema represents relation amongst a start-set (“Joe had 3 marbles”), transfer-set (“Then Tom gave him 5 marbles”) and result-set (“How many marbles does Joe have now?”). Table 3.1 lists the instantiations of three “schemas” used in WORDPRO. They also used a categorization of these schemas based on which component of the schema is the question asking about. The program solves a problem guided by list of rules which are applied sequentially. A drawback of WORDPRO was that it did not accept natural language text as input, it assumes that the input problem has already been mapped to a set of propositions.

Two other programs, namely CHIPS [Diane J. Briars, 1984] and ARITHPRO [Dellarosa, 1986] could also solve one-step arithmetic word problems with only one possible operation – addition/subtraction. Both CHIPS and ARITHPRO categorize simple word problems into the same three categories: compare, combine and change. But for both these models rigid limitations were imposed on the change verb (only “to give”) and on the order of the problem sentences (the first sentence of the problem must describe the number of objects the owner had to begin with, whereas the second sentence must contain the verb “gave”).

Schema	Instantiations
Change	John has 3 apples. Then he gave 2 apples to Susan. How many apples does Fred have now ?
Combine	Adam has 3 apples. Sarah has 6 apples. How many apples do they have altogether ?
Compare	Dennis has 6 apples. Fred has 5 apples. How many apples does Fred have less than Dennis ?

Table 3.1: Schemas used in WORDPRO

[Bakman, 2007] developed a more advanced arithmetic problem solver called ROBUST, that could understand multi-step arithmetic word problems and that too with extraneous information. An example for a problem handled by ROBUST is as follows:

*Ruth had 5 nuts more than Dan had. Ruth gave Dan 3 nuts. Dan gave 2 nuts to David. Now Dan has 4 nuts and David has 6 nuts. How many nuts does Ruth have now ?*

The concept of “schema” as used in the earlier works is adopted with further expansion of “change” schemas into six distinct categories as against only two used earlier. This fine grained categorization for change helped them to detect irrelevant numbers in the problem. The ROBUST simulation works by first parsing the problem text to split all sentences into propositions or simple sentences like “Ruth had 5 nuts more than Dan had”, “Ruth has ? nuts”, etc. for the above problem. The propositions having complex change verbs like “give” are further split into elementary ones (like “Ruth gave Dan 3 nuts” is split into “Ruth forfeited 3 nuts” and “Dan got 3 nuts”). Then the change schema relations are applied to the elementary propositions by substituting the constant values for the corresponding variables. Although it handles irrelevant numbers, ROBUST still restricts the operations to be addition and subtraction.

A detailed discussion of these systems can be found in [Mukherjee and Garain, 2008], and indeed a lot of the examples were taken from that survey. All the systems described above have different restrictions in the types of problems they handle. Also, they do not perform evaluation on any benchmark datasets; some of the systems are also not publicly available. As a result, it is impossible to compare these systems.

## 3.2 Solvers for other Domains

There were several systems developed also for related fields like Geometry, Physics, Chemistry and Calculus problems. Since they are less relevant for this dissertation, we only give a brief overview of these systems.

- The program, CARPS i.e. Calculus Rate Problem Solver [Charniak, 1968] written by Eugene Charniak reads, solves calculus rate problems stated in English.
- The system of HAPPINESS was developed by [Gelb, 1971] to solve simple probability questions.
- The NEWTON program written by [de Kleer, 1977] is an expert problem solver in the domain of mechanics, specifically, relating to kinematics of objects moving on surface.
- The program MECHO developed in Prolog by [Bundy et al., 1979] solves mechanics problems stated in English in the areas of pulley problems, statics problems, motion on smooth complex paths and motion under constant acceleration.
- ISAAC program written by [Novak, 1976] can read, understand, solve and draw pictures of physics problems stated in English.
- ALBERT developed by [Oberem, 1987] is an intelligent tutoring (CAI) system that not only understands and solves physics (more specifically kinematics) problems but can also teach a student how to solve them.

## 3.3 Recent Advances in Statistical Solvers

All the methods discussed earlier in this chapter use some form of rule based systems to arrive at the output equation or answer. However, hand-engineered rules have an obvious drawback – they do not provide robustness to the variability of input text. Indeed, most of the early systems constrained their input to be of a few specific forms, which allowed them to develop transformation rules for problem solving.

Recently, there has been a renewed interest in solving quantitative reasoning problems. It started in 2014 with the template based system of Nate Kushman [Kushman et al., 2014], which focused on solving algebra word problems. This was followed by the ARIES system of Allen Institute for Artificial Intelligence (AI2), which focused on solving addition subtraction problems. All recent methods (including the work done in this thesis) use some form of machine learning to achieve robustness, something which earlier systems lacked. However, it should be noted that some of the challenges that were identified and ideas introduced by the

early works are still relevant for building robust statistical methods. Many of these ideas are used alongside statistical methods in this thesis.

# Chapter 4

## Quantity Entailment

### 4.1 Introduction

In this chapter, we investigate the problem of reasoning with respect to quantities using only the local context. Consider the textual inference in Example 4.1, which we present as a Textual Entailment query [Dagan et al., 2013].

**Example 4.1**

T: *A bomb in a Hebrew University cafeteria killed five Americans and four Israelis.*

H: *A bombing at Hebrew University in Jerusalem killed nine people, including five Americans.*

To determine whether H is implied or entailed by T, we need to identify the quantities and the units from “*five Americans*” and “*four Israelis*”, as well as use the fact that “*Americans*” and “*Israelis*” are “*people*”. All these inferences can be made by processing local context of the quantities.

In this chapter, we describe some key steps necessary to facilitate reasoning about quantities with local context. We first describe a system developed to recognize quantities in free form text, infer units associated with them and convert them to a standardized form. For example, in

**Example 4.2**

*About six and a half hours later , Mr. Armstrong opened the landing craft’s hatch.*

we would like to extract the number 6.5, the corresponding unit, “*hour*”, and also determine that the quantity describes an *approximate* figure, not an exact one. One of the difficulties is that any noun or noun phrase can be a unit, and inferring them requires analyzing contextual cues and local sentence structure. As we show, in some cases deeper NLP techniques are required to support that.

We then develop a reasoning framework for quantities that we believe can play an important role in

general purpose textual inference. As an evaluation metric for local context based reasoning, we isolate the quantity reasoning component of the RTE task, and formulate Quantity Entailment (QE) – the task of determining whether a given quantity can be inferred from a given text snippet. We then describe our approach towards solving it.

As an additional evaluation, we also show the effectiveness of our system on an application of QE, a search for ranges of currency values. Given a query range, say from 1 million USD to 3 million USD, we want to find all mentions of money with values in this range. Using standard search engine technology to query all values in the range, in the various forms they could be expressed, is not feasible. Instead, we use our proposed approach to extract monetary mentions from text and normalize them, and then we use QE to verify them against the query.

We develop and annotate datasets<sup>1</sup> for evaluation, and show that our approach can handle the aforementioned reasoning tasks quite well.

The next section presents some related work on quantities and reasoning. We then formally define a *quantity* and describe our knowledge representation. The following sections describe quantities extraction and standardization. We next present the formulation of Quantity Entailment, and describe our reasoning framework for it.

## 4.2 Related Work

**Quantities in Textual Entailment** Quantities have been recognized as an important part of a textual entailment system [de Marneffe et al., 2008, Maccartney and Manning, 2008, Sammons et al., 2010], and [de Marneffe et al., 2008] claims that discrepancies in numbers are a common source of contradictions in natural language text. The authors describe a corpus of real-life contradictory pairs from multiple sources such as Wikipedia and Google News in which they found that 29% of the contradictions were due to numeric discrepancies. In addition, they analyzed several Textual Entailment datasets [Dagan et al., 2006] and found that numeric contradictions constitute 8.8% of contradictory entailment pairs.

**Monotonicity in Reasoning** The reasoning framework described in this chapter borrows ideas of monotonicity. depends quantities often depends on reasoning about monotonicity. The role of monotonicity in NL reasoning has been described in [Barwise and Cooper, 1981]. The authors categorize noun phrases as upward

---

<sup>1</sup>The datasets are available for download at [http://cogcomp.cs.illinois.edu/page/resource\\_view/95](http://cogcomp.cs.illinois.edu/page/resource_view/95). The related software is available at [http://cogcomp.cs.illinois.edu/page/software\\_view/Quantifier](http://cogcomp.cs.illinois.edu/page/software_view/Quantifier).

or downward monotonic, and also detect constructs where monotonicity depends on context. The large role of monotonicity in reasoning motivated attempts to reason directly at the surface level [Purdy, 1991], rather than converting first to logical forms. Our approach advocates this direction too.

**Representation of Quantities** [Kuehne, 2004a] investigates the various cases in which physical quantities are represented in descriptions of physical processes. Later, in [Kuehne, 2004b], a system to extract Qualitative Process Theory [Forbus, 1984] representations is implemented for a controlled subset of the English language. While these approaches do not scale to unrestricted English, they have influenced the quantity representation that we use.

### 4.3 Representing Quantities

In general, quantity refers to anything which is measurable. Our quantities representation is influenced by the one proposed in [Forbus, 1984] but we propose a simpler version of their Qualitative Process theory:

**Definition (Quantity-Value Representation)** In Quantity-Value Representation (QVR), a quantity is represented as a triple  $(v, u, c)$ , where constituents in the triple correspond, respectively, to:

1. Value: a numeric value, range, or set of values which measure the aspect, e.g. more than 500, one or two, thousands, March 18, 1986. The value can also be described via symbolic value (e.g., “below the freezing point”). We do not store surface forms explicitly, but convert them to a set or range. For example, “more than 500” is stored as the range  $(500, +\infty)$ . Details of these conversions are given in Section 4.4.2.
2. Units: a noun phrase that describes what the value is associated with. e.g., inches, minutes, bananas. The phrase “US soldiers” in the phrase “Five US soldiers” is a unit.
3. Change: specifies how the parameter is changing, e.g., increasing. This constituent often serves as an indication of whether or not the value is relative to another. For example, “She will receive an [additional 50 cents per hour]”, “The stock [increased 10 percent]”, “Jim has [5 balls more] than Tim”.

### 4.4 Extraction of Quantities

In this section we describe the first component of our approach, that of identifying quantities and units in text and standardizing their representation. We use a a two step approach to extract quantities from free

form text.

1. **Segmentation** This step takes raw text and finds segments of contiguous text which describe quantities.
2. **Standardization** Using the phrases extracted in the previous step, we derive the QVR.

An overview of our method is given in Algorithm 1.

---

**Algorithm 1** QuantityExtraction(  $T$  )

---

**Input:** Text  $T$

**Output:** Set of Quantity-value triples extracted from  $T$

```
1:  $Q \leftarrow \emptyset$ 
2:  $S \leftarrow \text{Segmentation}( T )$ 
3: for all segment  $s \in S$  do
4:    $q \leftarrow \text{Standardization}( s )$ 
5:   if unit of  $q$  not inferred then
6:      $q \leftarrow \text{InferUnitFromSemantics}( q, s, T )$ 
7:   end if
8:    $Q \leftarrow Q \cup \{q\}$ 
9: end for
10: return  $Q$ 
```

---

We model the **segmentation** step as a sequence segmentation task because quantities often appear as segments of contiguous text. We adapt and compare two approaches that were found successful in previous sequential segmentation work in NLP:

1. A Semi-CRF model [Sarawagi and Cohen, 2004], trained using a structured Perceptron algorithm [Collins, 2002], with Parameter Averaging [Freund and Schapire, 1998].
2. A bank of classifiers approach [Punyakanok and Roth, 2001] that we retrain with a new set of features.

The same feature set was used for both approaches. Despite the additional expressive power of CRFs, we found that the bank of classifiers (which is followed by a simple and tractable inference step) performs better for our task, and also requires significantly less computation time.



### 4.4.1 Features

For each token  $x_i$  in the input sequence we extract the following features:

1. **Word class features:**  $x_i$  appears in a list of known scientific units (e.g., meters, Fahrenheit), written numbers (e.g., two, fifteen), names of a months, day of the week, miscellaneous temporal words (e.g. today, tomorrow), currency units, etc.
2. **Character-based:**  $x_i$  contains a digit, is all digits, has a suffix (st,nd,rd,th).
3. **Part of speech tags:** we use the Illinois POS Tagger [Roth and Zelenko, 1998].
4. Most of the features were generated from a window of  $[-3, 3]$  around the current word. Additional features were generated from these by conjoining them with offset values from the current word.

### 4.4.2 Mapping Text Segments into QVR

We develop a rule-based **standardization** step, that is informed, as needed, by deeper NL processing, including semantic role labeling (SRL, [Palmer et al., 2010]) and Co-reference resolution. Some key steps of this procedure are as follows:

1. Convert written numbers to floating point: e.g., three thousand five hundred twenty  $\rightarrow$  3520.0
2. Convert dates to an internal date type: e.g., March 18th  $\rightarrow$  Date(03/18/XXXX)
3. Replace known names for ranges: e.g., teenage  $\rightarrow$  [13, 19] years-old.
4. Convert all scientific units to a standard base unit: e.g., 1 mile  $\rightarrow$  1609.344 meters.
5. Replace non-scientific units with WordNet synsets
6. Rewrite known units to a standard unit: e.g., USD, US\$, dollars  $\rightarrow$  US\$.
7. Standardize changing quantity: e.g., “additional 10 books”  $\rightarrow$  +10 [book].
8. Extract bounds: we use a list of phrases, such as “more than”, “less than”, “roughly”, “nearly”. By default, if a bound keyword is not present we assume the bound is “=”.
9. Modify value using bounds : We convert values which have a bound to a range of values.

Scalar implicature is taken into consideration here. Consider the sentence “John bought 10 books.”, although it can be interpreted that buying 5 books is a corollary of buying 10, in this case, we make the assumption that 5 books were not purchased. See section 4.5.2 for a discussion on the subject.

We use the following rules, where  $v$  is the value extracted before using bound information.

- $\leq v \rightarrow (-\infty, v]$ , similarly for  $\geq, <, >$ .
- $= v \rightarrow \{v\}$
- $\approx v \rightarrow [v - c.v, v + c.v]$ , we use  $c = 0.2$ .

### 4.4.3 Extraction of Units

In most cases, the units related to the numeric values appear adjacent to them. For example, in the sentence “*There are two books on the table*”, the unit “*book*” follows “*two*”. The sequence segmentation groups these words together, from which it is easy to extract the unit. However, in some cases, a better understanding of the text is needed to infer the units. Consider the following example:

**Example 4.3**

*A report from UNAIDS, the Joint United Nations Program on HIV/AIDS, released on Tuesday, shows the number of adults and children with HIV/AIDS reached 39.4 million in 2004.*

Here, we need to know that “*39.4 million*” refers to “*the number of adults and children with HIV/AIDS*”.

Also, in:

**Example 4.4**

*The number of member nations was 80 in 2000, and then it increased to 95.*

we need to know that the pronoun “*it*” refers to “*the number of member nations*”.

We employ a sequential process in our standardization. In case the first step described above fails to extract units, we make use of deeper processing of the sentence to accomplish that (see an evaluation of the contribution of this in the experimental section). These steps are denoted by the function **InferUnitFromSemantics()** in Algorithm 1. We apply coreference resolution to identify pronoun referents and then apply a Semantic Role Labeler, to recognize which terms are associated with the quantity, and can be potential units. In the case of example 4.3, the SRL tells us that for the verb “*reached*”, the associated subject is “*the*

*number of adults and children with HIV/AIDS*” and the object is the mention “39.4 million”. Hence, we conclude that the subject can be a candidate for the unit of “39.4 million”. For the purpose of entailment, we keep the entire set of possible word chunks, which are linked by the SRL to our quantity mention, as candidate units.

Since most units are found in positions adjacent to the numeric mention, we optimize on runtime by applying the SRL and coreference resolver only when the segmented chunk does not have adequate information to infer the unit. We use the Illinois Coreference Resolver ([Bengtson and Roth, 2008], [Kai-Wei Chang and Roth, 2013]) and the Illinois SRL [Punyakanok et al., 2008], for coreference and semantic role labelling, respectively.

## 4.5 Quantity Entailment

In this section we describe our approach to quantitative reasoning using local context of quantities. We first formulate the task of Quantity Entailment, and then describe our reasoning framework.

**Definition (Quantity Entailment)** Given a text passage  $T$  and a Quantity-Value triple  $h(c_h, v_h, u_h)$ , Quantity Entailment is a 3-way decision problem:

1. **entails**: there exists a quantity in  $T$  which entails  $h$ .
2. **contradicts**: no quantity in  $T$  entails  $h$ , but there is a quantity in  $T$  which contradicts  $h$ .
3. **no relation**: there exists no quantity in  $T$ , which is comparable with  $h$ .

Since the decision is about a single quantity-value triple, we believe this will test the capability to capture local contextual cues of the quantity. This task can also be seen as a building block for a general textual entailment system. The need to identify sub-problems of textual inference, in the context of the RTE task, has been motivated by [Sammons et al., 2010]. Quantity Entailment can be considered as one such step. Since we envision that our QE module will be one module in an RTE system, we expect that the RTE system will provide it with some control information. For example, it is often important to know whether the quantity is mentioned in an upward or downward monotonic context. Since we are evaluating our QE approach in isolation, we will always assume upward monotonicity, which is a lot more common. Monotonicity has been modeled with some success in entailment systems [Maccartney and Manning, 2008], thus providing a clear and intuitive framework for incorporating an inference resource like the Quantity Entailment module into a full textual entailment system.

### 4.5.1 Reasoning Framework

Our Quantity Entailment process has two phases: Extraction and Reasoning. In the Extraction Phase, we take a text passage  $T$  and extract Quantity-Value triples (value, units, change) from it. In the Reasoning phase, we apply a lightweight logical inference procedure to the triples extracted from  $T$  to check if  $h$  can be derived.

There are two types of rules applied in the Reasoning phase: Implicit Quantity Productions and Quantity Comparisons. The combination of these rules provides good coverage for the QE task.

#### Quantity Comparison

Quantity Comparison compares a quantity  $t : (v_t, u_t, c_t)$  extracted from  $T$  and the quantity  $h : (v_h, u_h, c_h)$  and decides whether  $h$  can be derived via some truth preserving transformation of  $t$ . There are three possibilities: ( $t$  entails  $h$ ), ( $t$  contradicts  $h$ ), or ( $t$  has no relation with  $h$ ). The overview is given in Alg. 2, which is designed under our assumption that entailing quantities should respect upward monotonicity. This requires monotonicity verification of both units and values.

In order for a quantity to contradict or entail another, their units must be comparable. Determining the comparability of scientific units is direct since they form a closed set. Comparing non-scientific units is more involved. The inference rule used here is as follows: if the syntactic heads of the unit phrases match (i.e., there is an Is-A or synonymy relation in either direction), then the phrases are comparable. These comparisons are encoded as a function **comparableUnits**( $u_t, u_h$ ), which returns true if the units  $u_t$  and  $u_h$  are comparable, or else returns false.

If the units are comparable, the direction of monotonicity (i.e., the direction of the Is-A relation between the heads and the effects of any relevant modifiers) is verified. The function **checkMonotonicityOfUnits**( $u_t, u_h$ ) returns true, if  $u_t$  is more specific than  $u_h$ , false otherwise.

To compute the Is-A and synonymy relations we use WordNet [Miller et al., 1990], an ontology of words which contains these relations. We also augment WordNet with two lists from Wikipedia (specifically, lists of Nationalities and Jobs).

Next, we check whether the values of the quantities compared obey the monotonicity assumption; we say that  $v_t$  is more specific than  $v_h$  if  $v_t$  is a subset of  $v_h$ . (Note that  $v_t$  and  $v_h$  are both represented as sets and hence, checking subset relation is straightforward.) For example, “*more than 50*”  $\subseteq$  “*at least 10*”. This rule also applies to dates, e.g. “*03/18/1986*”  $\subseteq$  “*March 1986*”. Respecting scalar implicature, we assume that “5” is subset of “less than 10”, but not “10”. Similar to the case of units, we use the function

**checkMonotonicityOfValues**( $v_t, v_h$ ) which returns true, if  $v_t$  is more specific than  $v_h$ , and false otherwise.

A quantity which represents some form of change of a quantity cannot be derived from a quantity which does not represent change and vice versa. We set  $c_t = \text{true}$  if  $t$  denotes change in a quantity, otherwise we set  $c_t = \text{false}$ .

---

**Algorithm 2** QuantityComparison(  $t, h$  )

---

**Input:** Quantity-value triples  $t(v_t, u_t, c_t)$  and  $h(v_h, u_h, c_h)$

**Output:** Returns whether  $t$  entails, contradicts or has no relation with  $h$

```
1: if  $c_t \neq c_h$  then
2:   return no relation
3: end if
4: if comparableUnits(  $u_t, u_h$  )= false then
5:   return no relation
6: end if
7: if checkMonotonicityOfUnits(  $u_t, u_h$  )= true then
8:   if checkMonotonicityOfValues(  $v_t, v_h$  )= true then
9:     return entails
10:  end if
11: end if
12: return contradicts
```

---

### Implicit Quantity Production Rules

There are many relationships among quantities which can be the source of implicit information. The following is an incomplete, but relatively broad coverage list of common patterns:

1. Range may imply duration, e.g., “*John lived in Miami from 1980 to 2000*” implies that John lived in Miami for a duration of 20 years.
2. Compatible terms may be combined and abstracted. The sentence “*I bought 3 bananas, 2 oranges, and 1 apple*” implies that 6 fruits were purchased.
3. Ratios can imply percentages. The sentence “*9 out of the 10 dentists interviewed recommend brushing your teeth*” implies that 90% of the dentists interviewed recommend brushing.

4. Composition: Quantities and units may sometimes be composed. Consider the following examples, the phrase “*six Korean couples*” means that there are 12 people; the phrase “*John gave six 30-minute speeches*” implies that John spoke for 180 minutes.

The rules used for producing implicit quantities employed in our system are the following:

- **(a ratio b)** if a is a percentage, then multiply its value with the value of b to obtain a new quantity with the units of b.
- **(a ratio b)** if a is not percentage, divide its value with the value of b to obtain a new quantity with the units of b.
- **(a range b)** take the difference of the two values to obtain a new quantity with the appropriate change of units, e.g., time-stamp minus time-stamp results in units of time.

---

**Algorithm 3** QuantityEntailment(  $T, h$  )

---

**Input:** Text  $T$  and a quantity-value triples  $h(v_h, u_h, c_h)$

**Output:** Returns whether  $T$  *entails*, *contradicts* or has *no relation* with  $h$

```

1:  $Q \leftarrow$  QuantityExtraction(  $T$  )
2:  $Q' \leftarrow$  GenerateImplicitQuantities(  $Q$  )
3:  $Q \leftarrow Q \cup Q'$ 
4: contradict  $\leftarrow$  false
5: for all quantity-value triple  $q \in Q$  do
6:   if QuantityComparison(  $q, h$  )= entails then
7:     return entails
8:   end if
9:   if QuantityComparison(  $q, h$  )= contradicts then
10:    contradict  $\leftarrow$  true
11:   end if
12: end for
13: if contradict= true then
14:   return contradicts
15: else
16:   return no relation
17: end if

```

---

## Lightweight Logical Inference

The QE inference procedure simply applies each of the implicit quantity production rules to the Quantity-Value triples extracted from the passage  $T$ , until no more quantities are produced. Then it compares each quantity  $t$  extracted from  $T$  with the quantity  $h$ , according to the quantity comparison rules described in Algorithm 2. If any quantity in  $T$  entails  $h$ , then “entails” is reported; if there is no quantity in  $T$  which can explain  $h$ , but there exists one which contradicts  $h$ , then “contradiction” is reported; otherwise “no relation” is reported. The complete approach to Quantity Entailment is given in Algorithm 3.

### 4.5.2 Scope of QE Inference

Our current QE procedure is limited in several ways. In all cases, we attribute these limitations to subtle and deeper language understanding, which we delegate to the application module that will use our QE procedure as a subroutine. Consider the following examples:

$T$  : Adam has exactly 100 dollars in the bank.

$H_1$  : Adam has 50 dollars in the bank.

$H_2$  : Adam’s bank balance is 50 dollars.

Here,  $T$  implies  $H_1$  but not  $H_2$ . However for both  $H_1$  and  $H_2$ , QE will infer that “50 dollars” is a contradiction to sentence  $T$ , since it cannot make the subtle distinction required here.

$T$  : Ten students passed the exam, but six students failed it.

$H$  : At least eight students failed the exam.

Here again, QE will only output that  $T$  implies “At least eight students”, despite the second part of  $T$ . QE reasons about the quantities, and there needs to be an application specific module that understands which quantity is related to the predicate “failed”.

There also exists limitations regarding inferences with respect to events that could occur over a period of time. In “*It was raining from 5 pm to 7 pm*” one needs to infer that “*It was raining at 6 pm*” although “*6 pm*” is more specific than “*5 pm to 7 pm*”. There is a need to understand the role of associated verbs and entities, and the monotonicity of the passages to infer the global entailment decision. Many of these challenges will be handled in the later chapters of the thesis.

## 4.6 Experimental Study

In this section, we seek to validate our proposed modeling. We evaluate our system’s performance on four tasks: Quantity Segmentation, Quantity Entailment and Currency Range Search. We do not directly evaluate our system’s ability to map raw text segments into our representation, but instead evaluate this capability extrinsically, in the context of the aforementioned tasks, since good Standardization is necessary to perform quantitative inference.

### 4.6.1 Datasets

**QE:** Due to lack of related work, an adequately annotated corpus does not exist. Thus, in order to evaluate our system, we used two collections:

1. **Sub-corpus of the RTE Datasets** We choose text-hypothesis pairs from RTE2–RTE4 datasets, which have quantity mentions in the hypothesis. Overall, we selected 384 text-hypothesis pairs with quantities in the hypothesis.
2. **Newswire Text** 600 sentences of newswire text were selected, all containing quantity mentions.

Both these datasets were manually annotated with the phrase boundaries of quantity mentions and had an inter-annotator agreement of 0.91. We restricted annotation to contiguous segments of text. No instances of implicit quantities were annotated. We also did not annotate these mentions with QVRs.

Limiting the annotations to contiguous spans of text results in a few instances of quantities which contain missing information, such as missing or ambiguous units, and several range and ratio relationships which were not annotated (e.g., we do not annotate the range expressed in “*from [5 million] in [1995] to [6 million] in [1996]*”, but do so in “[*from 5 million to 6 million*]”).

In the RTE sub-corpus we also annotated entailment pairs with information about which quantities entail, in addition to the boundary information. For each quantity in the hypothesis we labeled it as either “entails”, “no relation”, or “contradicts”, with an inter-annotator agreement of 0.95. There were 309 entailing quantities, 71 contradicting quantities and 56 quantities which were unrelated to the corresponding text. We also maintained the information about general entailment, that is, whether the hypothesis can be explained by the text. An example of an annotated RTE example is shown below.



### Annotation Example for RTE sub-corpus

T: *A bomb in a Hebrew University cafeteria killed [five Americans] and [four Israelis].*

H: *A bombing at Hebrew University in Jerusalem killed [nine people], including [five Americans].*

“nine people” : entails

“five Americans” : entails

Global entailment decision : entails

Although we limit our scope to infer the entailment decision for individual quantities mentioned in hypothesis, we hope to see future approaches use these individual decisions and combine them appropriately to obtain the global entailment decision.

**Currency Search** We developed a new dataset for evaluating currency search. Queries of various amounts of money like “1000\$”, “USD 2 million”, etc. were made on a search engine, and paragraphs containing monetary mentions were taken from the top search results. We collected 100 paragraphs containing various mentions of monetary values, and labeled them with the amount mentioned in them. We restricted the denominations to US dollars. The inter-annotator agreement was 0.98.

## 4.6.2 Quantity Segmentation

We evaluate the phrase boundary recognizer on the annotated RTE and newswire datasets described in the previous section, using the phrase-based  $F_1$  score. We compare the accuracy and running times of the Semi-CRF model (SC) [Sarawagi and Cohen, 2004] and the bank of classifiers model (C+I) (PR) [Punyakanok and Roth, 2001], using 10-fold cross-validation. Note that the standardizer can often recover from mistakes made at the segmentation level. Therefore, this performance does not necessarily upper bound the performance of the next step in our pipeline.

The segmentation we are aiming for does not directly follow from syntactic structure of a sentence. For example, in the sentence “ *The unemployment rate increased 10%*”, we would like to segment together “*increased 10%*”, since this tells us that the quantity denotes a rise in value. Also, in the sentence “*Apple restores push email in Germany, nearly two years after Motorola shut it down*” we would like to segment together “*nearly two years after*”. We consider a quantity to be correctly detected only when we have the exact phrase that we want, otherwise we consider the segment to be undetected.

Model	P%	R%	F%	Train Time	Test Time
Semi-CRF (SC)	75.6	77.7	76.6	15.8	1.5
C+I (PR)	80.3	79.3	79.8	1.0	1.0

Table 4.1: 10-fold cross-validation results of segmentation accuracy and time required for segmentation, the columns for runtime have been normalized and expressed as ratios

Table 4.1 describes the segmentation accuracy, as well as the ratio between the time taken by both approaches. The bank of classifiers approach gives slightly better accuracy than the semi-CRF model, and is also significantly faster.

### 4.6.3 Quantity Entailment

We evaluate the complete Quantity Entailment system, determining the overall loss due to the segmentation, as well as the contribution of the Coreference Resolver and SRL. We show the performance of 4 systems.

1. GOLDSEG : Uses gold segmentation, and does not use SRL and Coreference Resolver.
2. GOLDSEG+SEM : Uses gold segmentation, and also uses SRL and Coreference Resolver to infer units.
3. PREDSEG : Performs segmentation, and does not use SRL and Coreference Resolver.
4. PREDSEG+SEM : Performs segmentation, and uses SRL and Coreference Resolver.

The baseline is an exact string matching algorithm. It answers “entails” if the quantity unit and value are present in the text, and answers “contradicts” if only the unit matches and the value does not. Otherwise, it returns “no relation”. The results are shown in Table 4.2. Note that exact match only supports 43.3% of the entailment decisions. It is also evident that the deeper semantic analysis using SRL and Coreference improves the quantitative inference.

Task	System	P%	R%	F%
Entailment	Baseline	100.0	43.3	60.5
	GOLDSEG	98.5	88.0	92.9
	+SEM	97.8	88.6	93.0
	PREDSEG	94.9	76.2	84.5
	+SEM	95.4	78.3	86.0
Contradiction	Baseline	16.6	48.5	24.8
	GOLDSEG	61.6	92.9	74.2
	+SEM	64.3	91.5	75.5
	PREDSEG	51.9	79.7	62.8
	+SEM	52.8	81.1	64.0
No Relation	Baseline	41.8	71.9	52.9
	GOLDSEG	81.1	76.7	78.8
	+SEM	80.0	78.5	79.3
	PREDSEG	54.0	75.4	62.9
	+SEM	56.3	72.7	63.5

Table 4.2: Results of QE; Adding Semantics(+SEM) consistently improves performance; Only 43.3% of entailing quantities can be recovered by simple string matching

#### 4.6.4 Currency Range Search

Table 4.3 shows the performance of our system in detecting currency phrases. We evaluate our system on the proportion of monetary mentions it recognized and standardized correctly from queried ranges of currency values, and report micro-averaged scores. Note that range search is a direct application of QE, where the quantity is a range of values, and the text is the corpus we want to search. All instances of “entails” correspond to search hits. The baseline here is also a string matching algorithm, which searches for numbers in the text.

System	P%	R%	F%
Baseline	72.0	69.2	70.5
PREDSEG+SEM	96.0	93.5	94.8

Table 4.3: Micro-averaged accuracy in detecting monetary mentions

### 4.6.5 Qualitative Analysis

The segmentation module made mistakes in detecting exact boundaries for uncommon phrases, e.g., “hundreds of thousands of people”, and “mid-1970’s”. Detection of missing units is problematic in cases like “Three eggs are better than two”. The SRL returns “Three eggs” as a candidate unit, which needs to be pruned appropriately to obtain the correct unit. The primary limitation of the reasoning system in both tasks is the lack of an extensive knowledge base. Wordnet based synsets prove to be insufficient to infer whether units are compatible. Also, there are certain reasoning patterns and various implicit relations between quantities which are not currently handled in the system. For example, inferring from the sentence “*Militants in Rwanda killed an [average of 8,000 people per day] for [100 days]*” that “around 800,000 people were killed”. Also, implication of ratios can be involved. For example, the sentence “[*One out of 100 participating students*] will get the award” implies that there were “*100 participating students*”, whereas “[*9 out of 10 dentists*] recommend brushing” does not imply there were 10 dentists.

## 4.7 Conclusion

We studied reasoning about quantities in natural language text, with respect to local context. We have identified and defined an interesting and useful slice of the Textual Entailment problem, the Quantity Entailment task. Since this problem involves inferences about individual quantities, it allows us to capture quantitative reasoning with respect to local context.

Our ability to support local context based quantitative reasoning builds on a method we proposed for detecting and normalizing quantities in unrestricted English text; we developed a framework to remove variability and ambiguity from unstructured text by mapping it into a representation which makes reasoning more tractable. Once quantities are mapped into our representation, we can support the reasoning required by Quantity Entailment.

# Chapter 5

## Equation Parsing

### 5.1 Introduction

We now expand our focus from the local context of numbers, to consider entire sentences. We investigate how numbers interact with the entities, verbs and other components of the sentence. In particular, we investigate how we can extract numeric relations expressed in sentences, along with the arguments for those relations.

Extracting numeric relations from sentences is a key requirement for natural language understanding . As an example, consider the news article statement in Example 5.1. Understanding it requires identifying relevant entities and the mathematical relations expressed among them in text, and determining how to compose them. Similarly, solving a math word problem with a sentence like Example 5.2, requires realizing that it deals with a *single* number, knowing the meaning of “*difference*” and composing the right equation – “25” needs to be subtracted from a number only after it is multiplied by 3.

<b>Example 5.1</b> <i>Emanuel’s campaign contributions total three times those of his opponents put together.</i>
---

<b>Example 5.2</b> <i>Twice a number equals 25 less than triple the same number.</i>
--

<b>Example 5.3</b> <i>Flying with the wind , a bird was able to make 150 kilometers per hour.</i>
---

<b>Example 5.4</b> <i>The sum of two numbers is 80.</i>
---

<b>Example 5.5</b> <i>There are 54 5-dollar and 10-dollar notes.</i>
--

As a first step towards understanding such relations, we introduce the Equation Parsing task - given a sentence expressing a mathematical relation, the goal is to generate an equation representing the relation, and to map the variables in the equation to their corresponding noun phrases. To keep the problem tractable, we restrict the final output equation form to have at most two (possibly coreferent) variables, and assume that each quantity mentioned in the sentence can be used at most once in the final equation.<sup>1</sup> In example 5.1, the gold output of an equation parse should be  $V_1 = 3 \times V_2$ , with  $V_1 =$  “Emanuel’s campaign contributions”

---

<sup>1</sup>We empirically found that around 97% of sentences describing a relation have this property.

and  $V_2 =$  “those of his opponents put together”.

The task can be seen as a form of semantic parsing [Goldwasser and Roth, 2011, Kwiatkowski et al., 2013] where instead of mapping a sentence to a logical form, we want to map it to an equation. However, there are some key differences that make this problem very challenging in ways that differ from the “standard” semantic parsing. In Equation Parsing, not all the components of the sentence are mapped to the final equation. There is a need to identify noun phrases that correspond to variables in the relations and determine that some are irrelevant and can be dropped. Moreover, in difference from semantic parsing into logical forms, in Equation Parsing multiple phrases in the text could correspond to the same variable, and identical phrases in the text could correspond to multiple variables.

We call the problem of mapping noun phrases to variables the problem of *grounding variables*. Grounding is challenging for various reasons, key among them are that: (i) The text often does not mention “*variables*” explicitly, e.g., the sentence in example 5.3 describes a mathematical relation between the speed of bird and the speed of wind, without mentioning “speed” explicitly. (ii) Sometimes, multiple noun phrases could refer to the same variable. For instance, in example 5.2, both “*a number*” and “*the same number*” refer to the same variable. On the other hand, the same noun phrase might refer to multiple variables, as in example 5.4, where the noun phrase “two numbers” refer to two variables.

In addition, the task involves deciding which of the quantities identified in the sentence are relevant to the final equation generation. In example 5.5, both “5” and “10” are not relevant for the final equation “ $V_1 + V_2 = 54$ ”. Finally, the equation needs to be constructed from a list of relevant quantities and grounded variables. Overall, the output space becomes exponential in the number of quantities mentioned in the sentence.

Determining the final equation that corresponds to the text is an inference step over a very large space. To address this, we define the concept of “projectivity” - a condition where the final equation can be generated by combining adjacent numbers or variables, and show that most sentences expressing mathematical relations exhibit the projectivity property. Finally, we restrict our inference procedure to only search over equations which have this property.

Our approach builds on a pipeline of structured predictors that identify irrelevant quantities, recognize coreferent variables, and, finally, generate equations. We also leverage a high precision lexicon of mathematical expressions and develop a greedy lexicon matching strategy to guide inference. We discuss and exemplify the advantages of this approach and, in particular, explain where the “standard” NLP pipeline fails to support equation parsing, and necessitates the new approach proposed here. Another contribution

of this work is the development of a new annotated data set for the task of equation parsing. We evaluate our method on this dataset and show that our method predicts the correct equation in 70% of the cases and that in 60% of the time we also ground all variables correctly.

The next section presents a discussion of related work. Next we formally describe the task of equation parsing. The following sections describe our equation representation and the concept of projectivity, followed by the description of our algorithm to generate the equations and variable groundings from text. We conclude with experimental results.

## 5.2 Related Work

The work most related to this work is [Madaan et al., 2016], which focuses on extracting relation triples where one of the arguments is a number. In contrast, our work deals with multiple variables and complex equations involving them. This work is also conceptually related to work on semantic parsing – mapping natural language text to a formal meaning representation [Wong and Mooney, 2007, Clarke et al., 2010, Cai and Yates, 2013, Kwiatkowski et al., 2013, Goldwasser and Roth, 2011]. However, as mentioned earlier, there are some significant differences in the task definition that necessitate the development of a new approach.

## 5.3 The Equation Parsing Task

Equation parsing takes as input a sentence  $x$  describing a single mathematical equation, comprising one or two variables and other quantities mentioned in  $x$ . Let  $N$  be the set of noun phrases in the sentence  $x$ . The output of the task is the mathematical equation described in  $x$ , along with a mapping of each variable in the equation to its corresponding noun phrase in  $N$ . We refer to this mapping as the “grounding” of the variable; the noun phrase represents what the variable stands for in the equation. Table 5.1 gives an example of an input and output for the equation parsing of the text in example 5.2. Since an equation can be written in various forms, we use the form which most agrees with text, as our target output. So, for example 5.1, we will choose  $V_1 = 3 \times V_2$  and not  $V_2 = V_1 \div 3$ . In cases where several equation forms seem to be equally likely to be the target equation, we randomly choose one of them, and keep this choice consistent across the dataset.

The Equation Parsing Task	
Input	<i>Twice a number equals 25 less than triple the same number.</i>
Output	$2 \times V_1 = (3 \times V_1) - 25$ (Equation) $V_1 = \text{“a number”}$ (Grounding)

Table 5.1: Input and output for Equation Parsing

### 5.3.1 Equation Parse Representation

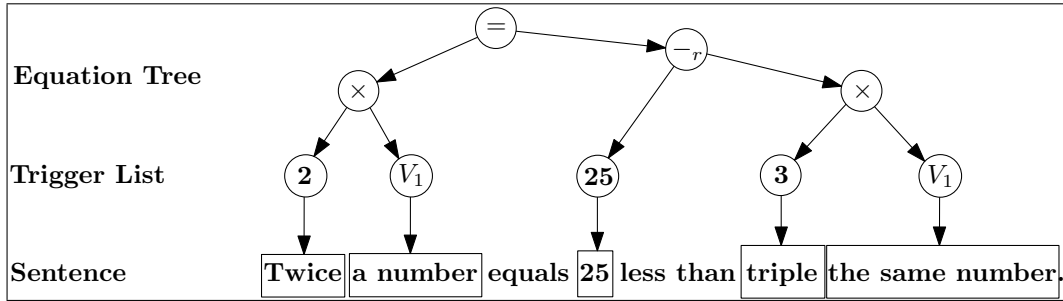


Figure 5.1: A sentence with its trigger list and equation tree.  $-_r$  indicates subtraction with order  $rl$ .

In this section, we introduce an equation parse for a sentence. An equation parse of a sentence  $x$  is a pair  $(T, E)$ , where  $T$  represents a set of *triggers* extracted from  $x$ , and  $E$  represents an *equation tree* formed with the set  $T$  as leaves. We now describe these terms in detail.

**Trigger** Given a sentence  $x$  mentioning a mathematical relation, a trigger can either be a *quantity trigger* expressed in  $x$ , or *variable trigger* which is a noun phrase in  $x$  corresponding to a variable. A *quantity trigger* is a tuple  $(q, s)$ , where  $q$  is the numeric value of the quantity mentioned in text, and  $s$  is the span of text from the sentence  $x$  which refers to the quantity. A *variable trigger* is a tuple  $(l, s)$ , where  $l$  represents the label of the variable, and  $s$  represents the noun phrase representing the variable. For example, for the sentence in Fig 5.1, the spans “Twice”, “25”, and “triple” generate quantity triggers, whereas “a number” and “the same number” generate variable triggers, with label  $V_1$ .

**Trigger List** The trigger list  $T$  for a sentence  $x$  contains one trigger for each variable mention and each numeric value used in the final equation expressed by the sentence  $x$ . The trigger list might consist of multiple triggers having the same label, or extracted from the same span of text. In the example sentence in Fig 5.1,



the trigger list comprises two triggers having the same label  $V_1$ . The final trigger list for the example in Fig 5.1 is  $\{(2, \text{"2"}), (V_1, \text{"a number"}), (25, \text{"25"}), (3, \text{"triple"}), (V_1, \text{"the same number"})\}$ . Note that there can be multiple valid trigger lists. In our example, we could have chosen both variable triggers to point to the same mention "a number". Quantity triggers in the trigger list form the *quantity trigger list*, and the variable triggers in trigger list form the *variable trigger list*.

Notation	Definition
Quantity Trigger	Mention of a quantity in text
Variable Trigger	Noun phrase coupled with variable label
Trigger	Quantity or variable trigger
Quantity Trigger List	List of quantity triggers, one for each number mention in equation
Variable Trigger List	List of variable triggers, one for each variable mention in equation
Trigger List	Union of quantity and variable trigger list
Equation Tree	Binary tree representation of equation
$lc(n), rc(n)$	Left and right child of node $n$
$EXPR(n)$	Expression represented by node $n$
$\odot(n)$	Operation at node $n$
$ORDER(n)$	Order of operation at node $n$
$LOCATION(n)$	Character offset of trigger representing leaf node $n$
$SPAN-START(n), SPAN-END(n)$	Start and end character offsets of span covered by node $n$

Table 5.2: Summary of notations used in this chapter

**Equation Tree** An equation tree of a sentence  $x$  is a binary tree whose leaves constitute the trigger list of  $x$ , and internal nodes (except the root) are labeled with one of the following operations – *addition*, *subtraction*, *multiplication*, *division*. In addition, for nodes which are labeled with subtraction or division, we maintain a separate variable to determine order of its children. The root of the tree is always labeled with the operation *equal*.

An equation tree is a natural representation for an equation. Each node  $n$  in an equation tree represents an expression  $EXPR(n)$ , and the label of the parent node determines how the expressions of its children are to be composed to construct its own expression. Let us denote the label for a non-leaf node  $n$  to be  $\odot(n)$ , where  $\odot(n) \in \{+, -, \times, \div, =\}$  and the order of a node  $n$ 's children by  $ORDER(n)$  (defined only for subtraction and division nodes), which takes values *lr* (Left-Right) or *rl* (Right-Left). For a leaf node  $n$ ,

the expression  $\text{EXPR}(n)$  represents the variable label, if  $n$  is a variable trigger, and the numeric value of the quantity, if it is a quantity trigger. Finally, we use  $lc(n)$  and  $rc(n)$  to represent the left and right child of node  $n$ , respectively. The equation represented by the tree can be generated as follows. For all non-leaf nodes  $n$ , we have

$$\text{EXPR}(n) = \begin{cases} \text{EXPR}(lc(n)) \odot(n) \text{EXPR}(rc(n)) & \text{if } \odot(n) \in \{+, \times, =\} \\ \text{EXPR}(lc(n)) \odot(n) \text{EXPR}(rc(n)) & \text{if } \odot(n) \in \{-, \div\} \wedge \text{ORDER}(n) = lr \\ \text{EXPR}(rc(n)) \odot(n) \text{EXPR}(lc(n)) & \text{if } \odot(n) \in \{-, \div\} \wedge \text{ORDER}(n) = rl \end{cases} \quad (5.1)$$

Given an equation tree  $\mathcal{T}$  of a sentence, the equation represented by it is the expression generated by the root of  $\mathcal{T}$  (following Equation 5.1). Referring to the equation tree in Fig 5.1, the node marked “ $-_r$ ” represents  $(3 \times V_1) - 25$ , and the root represents the full equation  $2 \times V_1 = (3 \times V_1) - 25$ .

## 5.4 Projectivity

For each leaf  $n$  of an equation tree  $T$ , we define a function  $\text{LOCATION}(\cdot)$ , to indicate the position of the corresponding trigger in text. We also define for each node  $n$  of equation tree  $T$ , functions  $\text{SPAN-START}(n)$  and  $\text{SPAN-END}(n)$  to denote the minimum span of text containing the leaves of the subtree rooted at  $n$ . We define them as follows:

$$\text{SPAN-START}(n) = \begin{cases} \text{LOCATION}(n) & \text{if } n \text{ is a leaf} \\ \min(\text{SPAN-START}(lc(n)), \text{SPAN-START}(rc(n))) & \text{otherwise} \end{cases}$$

$$\text{SPAN-END}(n) = \begin{cases} \text{LOCATION}(n) & \text{if } n \text{ is a leaf} \\ \max(\text{SPAN-END}(lc(n)), \text{SPAN-END}(rc(n))) & \text{otherwise} \end{cases}$$

An equation tree  $T$  is called *projective* iff for every node  $n$  of  $T$ , either  $\text{SPAN-END}(lc(n)) \leq \text{SPAN-START}(rc(n))$  or  $\text{SPAN-END}(rc(n)) \leq \text{SPAN-START}(lc(n))$ . In other words, the span of the left child and the right child cannot intersect in a projective equation tree<sup>2</sup>.

The key observation, as our corpus analysis indicates, is that for most sentences, *there exists* a trigger list, such that the equation tree representing the relation in the sentence is projective. However this might

<sup>2</sup>This is more general than the definition of projective trees used in dependency parsing [McDonald et al., 2005].

involve mapping two mentions of the same variable to different noun phrases. Figure 5.1 shows an example of a projective equation tree, which requires different mentions of  $V_1$  to be mapped to different noun phrases. If we had mapped both mentions of  $V_1$  to same noun phrase “a number”, the resulting equation tree would not have been projective. We collected 385 sentences which represent an equation with one or two mentions of variables, and each number in the sentence used at most once in the equation. We found that only one sentence among these could not generate a projective equation tree. (See Section 5.6.1 for details on dataset creation). Therefore, we develop an algorithmic approach for predicting projective equation trees, and show empirically that it compares favorably with ones which do not make the projective assumption.

## 5.5 Predicting Equation Parse

Equation parsing of a sentence involves predicting three components – Quantity Trigger List, Variable Trigger List and Equation Tree. We develop three structured prediction modules to predict each of the above components.

All our prediction modules take a similar form: given input  $x$  and output  $y$ , we learn a scoring function  $f_w(x, y)$ , which scores how likely is the output  $y$  given input  $x$ . The scoring function  $f_w(x, y)$  is linear,  $f_w(y) = w^T \phi(x, y)$ , where  $\phi(x, y)$  is a feature vector extracted from  $x$  and  $y$ . The inference problem, that is, the prediction  $y^*$  for an input  $x$  is then:  $y^* = \arg \max_{y \in \mathcal{Y}} f_w(y)$ , where  $\mathcal{Y}$  is the set of all allowed values of  $y$ .

### 5.5.1 Predicting Quantity Trigger List

Given input text and the quantities mentioned in it, the role of this step is to identify, for each quantity in the text, whether it should be part of the final equation. For instance, in example 5.5 in Section 5.1, both “5” and “10” are not relevant for the final equation “ $V_1 + V_2 = 54$ ”. Similarly, in example 5.4, the number “two” is irrelevant for the equation “ $V_1 + V_2 = 80$ ”.

We define for each quantity  $q$  in the sentence, a boolean value  $\text{RELEVANCE}(q)$ , which is set to *true* if  $q$  is relevant for the final equation, and to *false* otherwise. For the structured classification, the input  $x$  is the sentence along with a set of recognized quantities mentioned in it, and the output  $y$  is the relevance values for all quantities in the sentence. We empirically found that predicting all relevance values jointly performs better than having a binary classifier predict each one separately. The feature function  $\phi(x, y)$  used for the classification generates the following features :

1. **Neighborhood features** : For each quantity  $q$  in the input sentence, we add unigrams and bigrams generated from a window around  $q$ , part of speech tags of neighborhood tokens of  $q$ . We conjoin these features with  $\text{RELEVANCE}(q)$ .
2. **Quantity Features** : For each quantity  $q$ , we add unigrams and bigrams of the phrase representing the quantity. Also, we add a feature indicating whether the number is associated with number one or two, and whether it is the only number present in the sentence. These features are also conjoined with  $\text{RELEVANCE}(q)$ .

### 5.5.2 Predicting Variable Trigger List

The goal of this step is to predict the variable trigger list for the equation. Our structured classifier takes as input the sentence  $x$ , and the output  $y$  is either one or two noun-phrases, representing variables in the final equation. As we pointed out earlier, multiple groundings might be valid for any given variable, hence there can be multiple valid variable trigger lists. For every sentence  $x$ , we construct a set  $Y$  of valid outputs. Each element in  $Y$  corresponds to a valid variable trigger list. Finally, we aim to output only one of the elements of  $Y$ .

We modified the standard structured prediction algorithm to consider “superset supervision” and take into account multiple gold structures for an input  $x$ . We assume access to  $N$  training examples of the form :  $(x_1, Y_1), (x_2, Y_2), \dots, (x_N, Y_N)$ , where each  $Y_i$  is a set of valid outputs for the sentence  $x_i$ . Since we want to output only one variable trigger list, we want to score at least one  $y$  from  $Y_i$  higher than all other possible outputs, for each  $x_i$ . We use a modified latent structured SVM to learn the weight vector  $w$ . The algorithm treats the best choice among all of  $Y_i$  as a latent variable. At each iteration, for all  $x_i$ , the algorithm chooses the best choice  $y_i^*$  from the set  $Y_i$ , according to the weight vector  $w$ . Then,  $w$  is updated by learning on all  $(x_i, y_i^*)$  by a standard structured SVM algorithm. The details of the algorithm are in Algorithm 4.

---

**Algorithm 4** Structural SVM with Superset Supervision

---

**Input:** Training data  $T = \{(x_1, Y_1), (x_2, Y_2), \dots, (x_N, Y_N)\}$

**Output:** Trained weight vector  $w$

```
1:  $w \leftarrow w_0$ 
2: repeat
3:    $T' \leftarrow \emptyset$ 
4:   for all  $(x_i, Y_i) \in T$  do
5:      $y_i^* \leftarrow \arg \max_{y \in Y_i} w^T \phi(x_i, y)$ 
6:      $T' \leftarrow T' \cup \{(x_i, y_i^*)\}$ 
7:   end for
8:   Update  $w$  by running standard Structural SVM algorithm on  $T'$ 
9: until convergence
10: return  $w$ 
```

---

The distinction from standard latent structural SVM is in line 5 of Algorithm 4. In order to get the best choice  $y_i^*$  for input  $x_i$ , we search only inside  $Y_i$ , instead of all of  $\mathcal{Y}$ . A similar formulation can be found in [Björkelund and Kuhn, 2014]. The features  $\phi(x, y)$  used for variable trigger prediction are as follows:

1. **Variable features** : Unigrams and bigrams generated from the noun phrase representing variables, part of speech tags of tokens in noun phrase representing variables.
2. **Neighborhood Features** : Unigrams and POS tags from neighborhood of variables.

All the above features are conjoined with two labels, one denoting whether  $y$  has two variables or one, and the second denoting whether  $y$  has two variables represented by the same noun phrase.

If the output of the classifier is a pair of noun phrases, we use a rule based variable coreference detector, to determine whether both noun phrases should have the same variable label or not. The rules for variable coreference are as follows :

1. If both noun phrases are the same, and they do not have the token “two” or “2”, they have the same label.
2. If the noun phrases are different, and the noun phrase appearing later in the sentence contains tokens “itself”, “the same number”, they have the same label.
3. In all other cases, they have different labels.

Finally, each noun phrase contributes one variable trigger to the variable trigger list.

### 5.5.3 Predicting Equation Tree

It is natural to assume that the syntactic parse of the sentence could be very useful in addressing all the predictions we are making in the equation parsing tasks. However, it turns out that this is not the case – large portions of the syntactic parse will not be part of the equation parse, hence we need the aforementioned modules to address this. Nevertheless, in the next task of predicting the equation tree, we attempted to constraint the output space using guidance from the syntactic tree; we found, though, that even enforcing this weak level of output expectation is not productive. This was due to the poor performance of current syntactic parsers on the equation data (eg., in 32% of sentences, the Stanford parser made a mistake which does not allow recovering the correct equation).

The tree prediction module receives the trigger list predicted by the previous two modules, and the goal is to create an equation tree using the trigger list as the leaves of that tree. The input  $x$  is the sentence and the trigger list, and the output  $y$  is the equation tree representing the relation described in the sentence. We assume that the output will be a projective equation tree. For features  $\phi(x, y)$ , we extract for each non-leaf node  $n$  of the equation tree  $y$ , the following:

1. **Neighborhood Features** : Unigrams, bigrams and POS tags from neighborhood of  $\text{SPAN-START}(lc(n))$ ,  $\text{SPAN-START}(rc(n))$ ,  $\text{SPAN-END}(lc(n))$  and  $\text{SPAN-END}(rc(n))$ , conjoined with  $\odot(n)$  and  $\text{ORDER}(n)$ .
2. **Connecting Text Features** : Unigrams, bigrams and part of speech tags between  $\min(\text{SPAN-END}(lc(n)), \text{SPAN-END}(rc(n)))$  and  $\max(\text{SPAN-START}(lc(n)), \text{SPAN-START}(rc(n)))$ , conjoined with  $\odot(n)$  and  $\text{ORDER}(n)$ .
3. **Number Features** : In case we are combining two leaf nodes representing quantity triggers, we add a feature signifying whether one number is larger than the other.

The projectivity assumption implies that the final equation tree can be generated by combining only adjacent nodes, once the set of leaves is sorted based on  $\text{SPAN-START}(\cdot)$  values. This allows us to use CKY algorithm for inference. A natural approach to further reduce the output space is to conform to the projective structure of the syntactic parse of the sentence. However, we found this to adversely affect performance, due to the poor performance of syntactic parser on equation data.

**Lexicon** To bootstrap the equation parsing process, we developed a high precision lexicon to translate mathematical expressions to operations and orders, like “sum of A and B” translates to “A+B”, “A minus B” translates to “A-B”, etc. (where A and B denote placeholder numbers or expressions). At each step of CKY, while constructing a node  $n$  of the equation tree, we check for a lexicon text expression corresponding to node  $n$ . If found, we allow only the corresponding operation (and order) for node  $n$ , and do not explore other operations or orders. We show empirically that reducing the space using this greedy lexicon matching help improve performance. We found that using the lexicon rules as features instead of hard constraints do not help as much. Note that our lexicon comprises only generic math concepts, and around 50% of the sentences in our dataset do not contain any pattern from the lexicon. Details of the lexicon are added to the appendix.

Finally, given input sentence, we first predict the quantity trigger and the variable trigger lists. Given the complete trigger list, we predict the equation tree relating the components of the trigger list.

#### 5.5.4 Alternatives

A natural approach could be to jointly learn to predict all three components, to capture the dependencies among them. To investigate this, we developed a structured SVM which predicts all components jointly, using the union of the features of each component. We use approximate inference, first enumerating possible trigger lists, and then equation trees, and find the best scoring structure. However, this method did not outperform the pipeline method. The worse performance of joint learning is due to: (1) search space being too large for the joint model to do well given our dataset size of 385, and (2) our independent classifiers being good enough, thus supporting better joint inference. This tradeoff is strongly supported in the literature [Punyakanok et al., 2005b, Sutton and McCallum, 2007].

Another option is to enforce constraints between trigger list predictions, such as, variable triggers should not overlap with the quantity triggers. However, we noticed that often noun phrases returned by the Stanford parser were noisy, and would include neighboring numbers within the extracted noun phrases. This prevented us from enforcing such constraints.

## 5.6 Experimental Results

We now describe the data set, and the annotation procedure used. We then evaluate the system’s performance on predicting trigger list, equation tree, and the complete equation parse.

### 5.6.1 Dataset

We created a new dataset consisting of 385 sentences extracted from algebra word problems and financial news headlines. For algebra word problems, we used the MIT dataset [Kushman et al., 2014], and two high school mathematics textbooks, Elementary Algebra (College of Redwoods) and Beginning and Intermediate Algebra (Tyler Wallace). Financial news headlines were extracted from The Latest News feed of MarketWatch, over the month of February, 2015. All sentences with information describing a mathematical relation among at most two (possibly coreferent) variables, were chosen. Next, we pruned sentences which require multiple uses of a number to create the equation. This only removed a few time related sentences like “*In 10 years, John will be twice as old as his son.*”. We empirically found that around 97% of sentences describing a relation fall under the scope of our dataset. Some statistics of the dataset are provided in Table 5.3.

Source	#Sentences
MIT	245
Redwoods	25
Wallace	40
MarketWatch	75

Table 5.3: Statistics of dataset

The annotators were shown each sentence paired with the normalized equation representing the relation in the sentence. For each variable in the equation, the annotators were asked to mark spans of text which best describe what the variable represents. They were asked to annotate associated entities if exact variable description was not present. For instance, in example 5.3 (Section 1), the relation holds between the speed of bird and the speed of wind. However, “*speed*” is not explicitly mentioned in the sentence. In such cases, the annotators were asked to annotate the associated entities “*the wind*” and “*a bird*” as representing variables.

The guidelines also directed annotators to choose the longest possible mention, in case they feel the mention boundary is ambiguous. As a result, in the sentence, “*City Rentals rent an intermediate-size car for 18.95 dollars plus 0.21 per mile.*”, the phrase “*City Rentals rent an intermediate-size car*” was annotated as representing variable. We allow multiple mentions to be annotated for the same variable. In example 5.2 (Section 1), both “*a number*” and “*the same number*” were annotated as representing the same variable.

We wanted to consider only noun phrase constituents for variable grounding. Therefore, for each annotated span, we extracted the noun phrase with maximum overlap with the span, and used it to represent the variables. Finally, a tuple with each variable being mapped to one of the noun phrases representing it,



forms a valid output grounding (variable trigger list). We computed inter-annotator agreement on the final annotations where only noun phrases represent variables. The agreement ( $\kappa$ ) was 0.668, indicating good agreement. The average number of mention annotations per sentence was 1.74.

### 5.6.2 Equation Parsing Modules

In this section, we evaluate the performance of the individual modules of the equation parsing process. We report Accuracy - the fraction of correct predictions. Tables 5.4, 5.5 and 5.6 show the 5-fold cross validation accuracy of the various modules. In each case, we also report accuracy by removing each feature group, one at a time. In addition, for equation tree prediction, we also show the effect of lexicon, projectivity, conforming to syntactic parse constraints, and using lexicon as features instead of hard constraints. For all our experiments, we use the Stanford Parser [Chen and Manning, 2014], the Illinois POS tagger [Roth and Zelenko, 1998], the Illinois shallow parser [Punyakanok and Roth, 2001] and the Illinois-SL structured prediction package [Chang et al., 2015].

	Accuracy
All features	<b>95.3</b>
No Neighborhood features	42.5
No Quantity features	93.2

Table 5.4: Performance of Quantity Trigger List Prediction

	Accuracy
All features	<b>75.5</b>
No Variable features	58.6
No Neighborhood features	70.3

Table 5.5: Performance of Variable Trigger List Prediction

	Accuracy
All features	<b>78.9</b>
No Neighborhood features	64.3
No Connecting Text features	70.2
No Number features	77.6
No Lexicon	72.7
No Projectivity	72.8
Conform with Syntactic Parse	70.2
Lexicon as Features	74.5

Table 5.6: Performance of Equation Tree Prediction

### 5.6.3 Equation Parsing Results

In this section, we evaluate the performance of our system on the overall equation parsing task. We report Equation Accuracy - the fraction of sentences for which the system got the equation correct, and Equation+Grounding Accuracy - the fraction of sentences for which the system got both the equation and the grounding of variables correct. Table 5.7 shows the overall performance of our system, on a 5-fold cross validation. We compare against Joint Learning - a system which jointly learns to predict all relevant components of an equation parse (Section 5.5.4). We also compare with SPF [Artzi and Zettlemoyer, 2013], a publicly available semantic parser, which can learn from sentence-logical form pairs. We train SPF with sentence-equation pairs and a seed lexicon for mathematical terms (similar to ours), and report equation accuracy. Our structured predictors pipeline approach is shown to be superior to both Joint Learning and SPF.

Source	Equation Accuracy	Equation + Grounding Accuracy
Our System	<b>71.3 ± 3.7</b>	<b>61.2 ± 5.4</b>
Joint Learning	60.9 ± 9.1	50.0 ± 5.7
SPF	3.1	N/A

Table 5.7: Performance on equation parsing

SPF gets only a few sentences correct. We attribute this to the inability of SPF to handle overlapping mentions (like in Example 5.4), as well as its approach of parsing the whole sentence to the final output form. The developers of SPF also confirmed <sup>3</sup> that it is not suitable for equation parsing and that these results are expected. Since equation parsing is a more involved process, a slight adaptation of SPF does not seem possible, necessitating a more involved process, of the type we propose. Our approach, in contrast to SPF, can handle overlapping mentions, selects triggers from text, and parses the trigger list to form equations.

#### 5.6.4 Error Analysis

For variable trigger list prediction, around 25% of the errors were due to the predictor choosing a span which is contained within the correct span, e.g., when the target noun phrase is “The cost of a child’s ticket”, our predictor chose only “child’s ticket”. Although this choice might be sufficient for downstream tasks, we consider it to be incorrect in our current evaluation. Another 25% of the errors were due to selection of entities which do not participate in the relation. For example, in “A rancher raises 5 times as many cows as horses.”, our predictor chose “A rancher” and “cows” as variables, whereas the relation exists between “cows” and “horses”. For the prediction of the equation tree, we found that 35% of the errors were due to rare math concepts expressed in text. For example, “7 dollars short of the price” represents 7 dollars should be subtracted from the price. These errors can be handled by carefully augmenting the lexicon. Another 15% of the errors were due to lack of world knowledge, requiring understanding of time, speed, and distance.

### 5.7 Conclusion

In this chapter, we investigate methods that identify and understand mathematical relations expressed in text. We introduce the equation parsing task, which involves generating an equation from a sentence and identifying what the variables represent. We define the notion of projectivity, and construct a high precision lexicon, and use these to reduce the equation search space. Our experimental results are quite satisfying and raise a few interesting issues. In particular, it suggests that predicting equation parses using a pipeline of structured predictors performs better than jointly trained alternatives. As discussed, it also points out the limitation of the current NLP tools in supporting these tasks. Code and dataset are available at [http://cogcomp.cs.illinois.edu/page/publication\\_view/800](http://cogcomp.cs.illinois.edu/page/publication_view/800).

---

<sup>3</sup>Private communication

# Chapter 6

## Arithmetic Word Problems

### 6.1 Introduction

In this chapter, we expand our focus on quantitative reasoning over multiple numbers mentioned across different sentences. As a test bed, we choose the task of automatically solving arithmetic word problems. We found these problems to be a good abstraction of the challenges faced in quantitative reasoning across multiple sentences. Word problems arise naturally when reading the financial section of a newspaper, or following election coverage. These problems pose an interesting challenge to the NLP community, due to its concise and relatively straightforward text, and seemingly simple semantics. Arithmetic word problems are usually directed towards elementary school students, and can be solved by combining the numbers mentioned in text with basic operations (addition, subtraction, multiplication, division). They are simpler than algebra word problems which require students to identify variables, and form equations with these variables to solve the problem.

Initial methods to address arithmetic word problems have mostly focused on subsets of problems, restricting the number or the type of operations used [Roy et al., 2015, Hosseini et al., 2014] but could not deal with multi-step arithmetic problems involving all four basic operations. The template based method of [Kushman et al., 2014], on the other hand, can deal with all types of problems, but implicitly assumes that the solution is generated from a set of predefined equation templates.

In this chapter, we present a novel approach which can solve a general class of arithmetic problems without predefined equation templates. In particular, it can handle multiple step arithmetic problems as shown in Example 6.1.

Example 6.1

*Gwen was organizing her book case making sure each of the shelves had exactly 9 books on it. She has 2 types of books - mystery books and picture books. If she had 3 shelves of mystery books and 5 shelves of picture books, how many books did she have in total?*

The solution involves understanding that the number of shelves needs to be summed up, and that the total number of shelves needs to be multiplied by the number of books each shelf can hold. In addition, one has to understand that the number “2” is not a direct part of the solution of the problem.

While a solution to these problems eventually requires composing multi-step numeric expressions from text, we believe that directly predicting this complex expression from text is not feasible.

At the heart of our technical approach is the novel notion of an *Expression Tree*. We show that the arithmetic expressions we are interested in can always be represented using an Expression Tree that has some unique decomposition properties. This allows us to decompose the problem of mapping the text to the arithmetic expression to a collection of simple prediction problems, each determining the lowest common ancestor operation between a pair of quantities mentioned in the problem. We then formulate the decision problem of composing the final expression tree as a joint inference problem, via an objective function that consists of all these decomposed prediction problems, along with legitimacy and background knowledge constraints.

Learning to generate the simpler decomposed expressions allows us to support effectively support quantitative reasoning over multiple numbers, and allows generalization across problems types. In particular, our system could solve Example 6.1 even though it has never seen a problem that requires both addition and multiplication operations.

We also introduce a second concept, that of *quantity schema*, that allows us to focus on the information relevant to each quantity mentioned in the text. We show that features extracted from quantity schemas help reasoning effectively about the solution. Moreover, quantity schemas help identify unnecessary text snippets in the problem text. For instance, in Example 6.2, the information that “Tom washed cars over the weekend” is irrelevant; he could have performed any activity to earn money. In order to solve the problem, we only need to know that he had \$76 last week, and now he has \$86.

Example 6.2
<i>Last week Tom had \$74. He washed cars over the weekend and now has \$86. How much money did he make from the job?</i>

We combine the classifiers’ decisions using a constrained inference framework that allows for incorporating world knowledge as constraints. For example, we deliberately incorporate the information that, if the problems asks about an “amount”, the answer must be positive, and if the question starts with “how many”, the answer will most likely be an integer.

Our system is evaluated on two existing datasets of arithmetic word problems, achieving state of the art performance on both. We also create a new dataset of multistep arithmetic problems, and show that our system achieves competitive performance in this challenging evaluation setting.

The next section describes the related work in the area of automated math word problem solving. We then present the theory of expression trees and our decomposition strategy that is based on it. Sec. 4 presents the overall computational approach, including the way we use quantity schemas to learn the mapping from text to expression tree components. Finally, we discuss our experimental study and conclude.

## 6.2 Related Work

Work in automated arithmetic problem solvers has focused on a restricted subset of problems. The system described in [Hosseini et al., 2014, Mitra and Baral, 2016] handles only addition and subtraction problems, and requires additional annotated data for verb categories. In contrast, our system does not require any additional annotations and can handle a more general category of problems. The approach in [Roy et al., 2015] supports all four basic operations, and uses a pipeline of classifiers to predict different properties of the problem. However, it makes assumptions on the number of quantities mentioned in the problem text, as well as the number of arithmetic steps required to solve the problem. In contrast, our system does not have any such restrictions, effectively handling problems mentioning multiple quantities and requiring multiple steps. Kushman’s approach to automatically solving algebra word problems [Kushman et al., 2014] might be the most related to ours. It tries to map numbers from the problem text to predefined equation templates. However, they implicitly assume that similar equation forms have been seen in the training data. In contrast, our system can perform competitively, even when it has never seen similar expressions in training.

There is a recent interest in understanding text for the purpose of solving scientific and quantitative

problems of various kinds. Our approach is related to work in understanding and solving elementary school standardized tests [Clark, 2015]. The system described in [Berant et al., 2014] attempts to automatically answer biology questions, by extracting the structure of biological processes from text. There has also been efforts to solve geometry questions by jointly understanding diagrams and associated text [Seo et al., 2014].

Our constrained inference module falls under the general framework of Constrained Conditional Models (CCM) [Chang et al., 2012]. In particular, we use the  $L+I$  scheme of CCMs, which predicts structured output by independently learning several simple components, combining them at inference time. This has been successfully used to incorporate world knowledge at inference time, as well as getting around the need for large amounts of jointly annotated data for structured prediction [Roth and Yih, 2005, Punyakanok et al., 2005a, Punyakanok et al., 2008, Clarke and Lapata, 2006, Barzilay and Lapata, 2006, Roy et al., 2015].

### 6.3 Expression Tree and Problem Decomposition

We address the problem of automatically solving arithmetic word problems. The input to our system is the problem text  $P$ , which mentions  $n$  quantities  $q_1, q_2, \dots, q_n$ . Our goal is to map this problem to a read-once arithmetic expression  $E$  that, when evaluated, provides the problem’s solution. We define a read-once arithmetic expression as one that makes use of each quantity at most once. We say that  $E$  is a *valid* expression, if it is such a Read-Once arithmetic expression, and we only consider in this work problems that *can be* solved using valid expressions (it’s possible that they can be solved also with invalid expressions).

An expression tree  $\mathcal{T}$  for a valid expression  $E$  is a binary tree whose leaves represent quantities, and each internal node represents one of the four basic operations. For a non-leaf node  $n$ , we represent the operation associated with it as  $\odot(n)$ , and its left and right child as  $lc(n)$  and  $rc(n)$  respectively. The numeric value of the quantity associated with a leaf node  $n$  is denoted as  $Q(n)$ . Each node  $n$  also has a value associated with it, represented as  $VAL(n)$ , which can be computed in a recursive way as follows:

$$VAL(n) = \begin{cases} Q(n) & \text{if } n \text{ is a leaf} \\ VAL(lc(n)) \odot(n) VAL(rc(n)) & \text{otherwise} \end{cases} \quad (6.1)$$

For any expression tree  $\mathcal{T}$  for expression  $E$  with root node  $n_{root}$ , the value of  $VAL(n_{root})$  is exactly equal to the numeric value of the expression  $E$ . Therefore, this gives a natural representation of numeric expressions, providing a natural parenthesization of the numeric expression. Fig 6.1 shows an example of an arithmetic

problem with solution expression and an expression tree for the solution expression.

<p><b>Problem</b></p> <p><i>Gwen was organizing her book case making sure each of the shelves had exactly 9 books on it. She has 2 types of books - mystery books and picture books. If she had 3 shelves of mystery books and 5 shelves of picture books, how many books did she have total?</i></p>
<p><b>Solution</b> <math>(3 + 5) \times 9 = 72</math></p>
<p><b>Expression Tree of Solution</b></p> <pre>graph TD;     A((x)) --&gt; B((+));     A --&gt; C((9));     B --&gt; D((3));     B --&gt; E((5));</pre>

Figure 6.1: An arithmetic word problem, solution expression and the corresponding expression tree

**Definition** An expression tree  $\mathcal{T}$  for a valid expression  $E$  is called *monotonic* if it satisfies the following conditions:

1. If an addition node is connected to a subtraction node, then the subtraction node is the parent.
2. If a multiplication node is connected to a division node, then the division node is the parent.
3. Two subtraction nodes cannot be connected to each other.
4. Two division nodes cannot be connected to each other.

Fig 6.2 shows two different expression trees for the same expression. Fig 6.2b is monotonic whereas fig 6.2a is not.

Our decomposition relies on the idea of monotonic expression trees. We try to predict for each pair of quantities  $q_i, q_j$ , the operation at the lowest common ancestor (LCA) node of the monotonic expression tree for the solution expression. We also predict for each quantity, whether it is relevant to the solution. Finally, an inference module combines all these predictions.



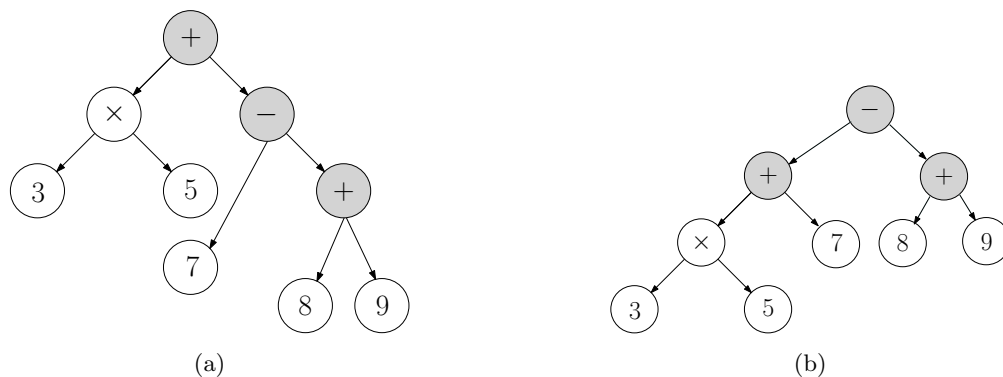


Figure 6.2: Two different expression trees for the numeric expression  $(3 \times 5) + 7 - 8 - 9$ . The right one is monotonic, whereas the left one is not.

In the rest of the section, we show that for any pair of quantities  $q_i, q_j$  in the solution expression, any monotonic tree for the solution expression has the same LCA operation. Therefore, predicting the LCA operation becomes a multiclass classification problem.

The reason that we consider the monotonic representation of the expression tree is that different trees could otherwise give different LCA operation for a given pair of quantities. For example, in Fig 6.2, the LCA operation for quantities 5 and 8 can be + or -, depending on which tree is considered.

**Definition** We define an ***addition-subtraction chain*** of an expression tree to be the maximal connected set of nodes labeled with addition or subtraction.

The nodes of an addition-subtraction (AS) chain  $C$  represent a set of terms being added or subtracted. These terms are sub-expressions created by subtrees rooted at neighboring nodes of the chain. We call these terms the ***chain terms*** of  $C$ , and the whole expression, after node operations have been applied to the chain terms, the ***chain expression*** of  $C$ . For example, in fig 6.2, the shaded nodes form an addition-subtraction chain. The chain expression is  $(3 \times 5) + 7 - 8 - 9$ , and the chain terms are  $3 \times 5$ , 7, 8 and 9. We define a ***multiplication-division (MD) chain*** in a similar way.

**Theorem 6.3.1.** *Every valid expression can be represented by a monotonic expression tree.*

*Proof.* The proof is procedural, that is, we provide a method to convert any expression tree to a monotonic expression tree for the same expression. Consider a non-monotonic expression tree  $E$ , and without loss of generality, assume that the first condition for monotonicity is not valid. Therefore, there exists an addition node  $n_i$  and a subtraction node  $n_j$ , and  $n_i$  is the parent of  $n_j$ . Consider an addition-subtraction chain  $C$  which includes  $n_i, n_j$ . We now replace the nodes of  $C$  and its subtrees in the following way. We add a single subtraction node  $n_-$ . The left subtree of  $n_-$  has all the addition chain terms connected by addition nodes,

and the right subtree of  $n_-$  has all the subtraction chain terms connected by addition nodes. Both subtrees of  $n_-$  only require addition nodes, hence monotonicity condition is satisfied. We can construct the monotonic tree in Fig 6.2b from the non-monotonic tree of Fig 6.2a using this procedure. The addition chain terms are  $3 \times 5$  and 7, and the subtraction chain terms are 8 and 9. As as was described above, we introduce the root subtraction node in Fig 6.2b and attach the addition chain terms to the left and the subtraction chain terms to the right. The same line of reasoning can be used to handle the second condition with multiplication and division replacing addition and subtraction, respectively.

□

**Theorem 6.3.2.** *Consider two valid expression trees  $\mathcal{T}_1$  and  $\mathcal{T}_2$  for the same expression  $E$ . Let  $C_1, C_2$  be the chain containing the root nodes of  $\mathcal{T}_1$  and  $\mathcal{T}_2$  respectively. The chain type (addition-subtraction or multiplication-division) as well as the the set of chain terms of  $C_1$  and  $C_2$  are identical.*

*Proof.* We first prove that the chains containing the roots are both AS or both MD, and then show that the chain terms are also identical.

We prove by contradiction that the chain type is same. Let  $C_1$ 's type be “addition-subtraction” and  $C_2$ 's type be “multiplication-division” (without loss of generality). Since both  $C_1$  and  $C_2$  generate the same expression  $E$ , we have that  $E$  can be represented as sum (or difference) of two expressions as well as product(or division) of two expressions. Transforming a sum (or difference) of expressions to a product (or division) requires taking common terms from the expressions, which imply that the sum (or difference) had duplicate quantities. The opposite transformation adds same term to various expressions leading to multiple uses of the same quantity. Therefore, this will force at least one of  $C_1$  and  $C_2$  to use the same quantity more than once, violating validity.

We now need to show that individual chain terms are also identical. Without loss of generality, let us assume that both  $C_1$  and  $C_2$  are “addition-subtraction” chains. Suppose the chain terms of  $C_1$  and  $C_2$  are not identical. The chain expression for both the chains will be the same (since they are root chains, the chain expressions has to be the same as  $E$ ). Let the chain expression for  $C_1$  be  $\sum_i t_i - \sum_i t'_i$ , where  $t_i$ 's are the addition chain terms and  $t'_i$  are the subtraction chain terms. Similarly, let the chain expression for  $C_2$  be  $\sum_i s_i - \sum_i s'_i$ . We know that  $\sum_i t_i - \sum_i t'_i = \sum_i s_i - \sum_i s'_i$ , but the set of  $t_i$ 's and  $t'_i$ 's is not the same as the set of  $s_i$  and  $s'_i$ 's. However it should be possible to transform one form to the other using mathematical manipulations. This transformation will involve taking common terms, or multiplying two terms, or both. Following previous explanation, this will force one of the expressions to have duplicate quantities, violating validity. Hence, the chain terms of  $C_1$  and  $C_2$  are identical.

□

Consider an expression tree  $\mathcal{T}$  for a valid expression  $E$ . For a distinct pair of quantities  $q_i, q_j$  participating in expression  $E$ , we denote by  $n_i, n_j$  the leaves of the expression tree  $\mathcal{T}$  representing  $q_i, q_j$ , respectively. Let  $n_{LCA}(q_i, q_j; \mathcal{T})$  to be the lowest common ancestor node of  $n_i$  and  $n_j$ . We also define  $order(q_i, q_j; \mathcal{T})$  to be true if  $n_i$  appears in the left subtree of  $n_{LCA}(q_i, q_j; \mathcal{T})$  and  $n_j$  appears in the right subtree of  $n_{LCA}(q_i, q_j; \mathcal{T})$  and set  $order(q_i, q_j; \mathcal{T})$  to false otherwise. Finally we define  $\odot_{LCA}(q_i, q_j; \mathcal{T})$  for a pair of quantities  $q_i, q_j$  as follows :

$$\odot_{LCA}(q_i, q_j, \mathcal{T}) = \begin{cases} + & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = + \\ \times & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = \times \\ - & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = - \text{ and } order(q_i, q_j; \mathcal{T}) = true \\ -_{reverse} & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = - \text{ and } order(q_i, q_j; \mathcal{T}) = false \\ \div & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = \div \text{ and } order(q_i, q_j; \mathcal{T}) = true \\ \div_{reverse} & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = \div \text{ and } order(q_i, q_j; \mathcal{T}) = false \end{cases} \quad (6.2)$$

**Definition** Given two expression trees  $\mathcal{T}_1$  and  $\mathcal{T}_2$  for the same expression  $E$ ,  $\mathcal{T}_1$  is *LCA-equivalent* to  $\mathcal{T}_2$  if for every pair quantities  $q_i, q_j$  in the expression  $E$ , we have  $\odot_{LCA}(q_i, q_j, \mathcal{T}_1) = \odot_{LCA}(q_i, q_j, \mathcal{T}_2)$ .

**Theorem 6.3.3.** *All monotonic expression trees for an expression are LCA-equivalent to each other.*

*Proof.* We prove by induction on the number of quantities used in an expression. For all expressions  $E$  with 2 quantities, there exists only one monotonic expression tree, and hence, the statement is trivially true. This satisfies our base case.

For the inductive case, we assume that for all expressions with  $k < n$  quantities, the theorem is true. Now, we need to prove that any expression with  $n$  nodes will also satisfy the property.

Consider a valid (as in all cases) expression  $E$ , with monotonic expression trees  $\mathcal{T}_1$  and  $\mathcal{T}_2$ . From theorem 6.3.2, we know that the chains containing the roots of  $\mathcal{T}_1$  and  $\mathcal{T}_2$  have identical type and terms. Given two quantities  $q_i, q_j$  of  $E$ , the lowest common ancestor of both  $\mathcal{T}_1$  and  $\mathcal{T}_2$  will either both belong to the chain containing the root, or both belong to one of the chain terms. If the LCA node is part of the chain for both  $\mathcal{T}_1$  and  $\mathcal{T}_2$ , monotonic property ensures that the LCA operation will be identical. If the LCA node is part of a chain term (which is an expression tree of size less than  $n$ ), the property is satisfied by induction hypothesis.

□

The theory just presented suggests that it is possible to uniquely decompose the overall problem to simpler steps and this will be exploited in the next section.

## 6.4 Mapping Problems to Expression Trees

Given the uniqueness properties proved in Sec. 6.3, it is sufficient to identify the operation between any two relevant quantities in the text, in order to determine the unique valid expression. In fact, identifying the operation between *any* pair of quantities provides much needed redundancy given the uncertainty in identifying the operation from text, and we exploit it in our final joint inference.

Consequently, our overall method proceeds as follows: given the problem text  $P$ , we detect quantities  $q_1, q_2, \dots, q_n$ . We then use two classifiers, one for relevance and other to predict the LCA operations for a monotonic expression tree of the solution. Our training makes use of the notion of quantity schemas, which we describe in Section 6.4.2. The distributional output of these classifiers is then used in a joint inference procedure that determines the final expression tree.

Our training data consists of problem text paired with a monotonic expression tree for the solution expression and alignment of quantities in the expression to quantity mentions in the problem text. Both the relevance and LCA operation classifiers are trained on gold annotations.

### 6.4.1 Global Inference for Expression Trees

In this subsection, we define the scoring functions corresponding to the decomposed problems, and show how we combine these scores to perform global inference. For a problem  $P$  with quantities  $q_1, q_2, \dots, q_n$ , we define the following scoring functions:

1.  $\text{PAIR}(q_i, q_j, op)$  : Scores the likelihood of  $\odot_{LCA}(q_i, q_j, \mathcal{T}) = op$ , where  $\mathcal{T}$  is a monotone expression tree of the solution expression of  $P$ . A multiclass classifier trained to predict LCA operations (Section 6.4.4) can provide these scores.
2.  $\text{IRR}(q)$  : Scores the likelihood of quantity  $q$  being an irrelevant quantity in  $P$ , that is,  $q$  is not used in creating the solution. A binary classifier trained to predict whether a quantity  $q$  is relevant or not (Section 6.4.3), can provide these scores.

For an expression  $E$ , let  $\mathcal{I}(E)$  be the set of all quantities in  $P$  which are not used in expression  $E$ . Let  $\mathcal{T}$  be a monotonic expression tree for  $E$ . We define  $\text{SCORE}(E)$  of an expression  $E$  in terms of the above scoring functions and a scaling parameter  $w_{\text{IRR}}$  as follows:

$$\text{SCORE}(E) = w_{\text{IRR}} \sum_{q \in \mathcal{I}(E)} \text{IRR}(q) + \sum_{q_i, q_j \notin \mathcal{I}(E)} \text{PAIR}(q_i, q_j, \odot_{\text{LCA}}(q_i, q_j, \mathcal{T})) \quad (6.3)$$

Our final expression tree is an outcome of a constrained optimization process, following [Roth and Yih, 2004, Chang et al., 2012]. Our objective function makes use of the scores returned by  $\text{IRR}(\cdot)$  and  $\text{PAIR}(\cdot)$  to determine the expression tree and is constrained by legitimacy and background knowledge constraints, detailed below.

1. **Positive Answer:** Most arithmetic problems asking for amounts or number of objects usually have a positive number as an answer. Therefore, while searching for the best scoring expression, we reject expressions generating negative answer.
2. **Integral Answer:** Problems with questions such as ‘how many’ usually expect integral solutions. We only consider integral solutions as legitimate outputs for such problems.

Let  $\mathcal{C}$  be the set of valid expressions that can be formed using the quantities in a problem  $P$ , and which satisfy the above constraints. The inference algorithm now becomes the following:

$$\arg \max_{E \in \mathcal{C}} \text{SCORE}(E) \quad (6.4)$$

The space of possible expressions is large, and we employ a beam search strategy to find the highest scoring constraint satisfying expression [Chang et al., 2012]. We construct an expression tree using a bottom up approach, first enumerating all possible sets of irrelevant quantities, and next over all possible expressions, keeping the top  $k$  at each step. We give details below.

1. **Enumerating Irrelevant Quantities:** We generate a state for all possible sets of irrelevant quantities, ensuring that there is at least two relevant quantities in each state. We refer to each of the relevant quantities in each state as a term. Therefore, each state can be represented as a set of terms.
2. **Enumerating Expressions:** For generating a next state  $S'$  from  $S$ , we choose a pair of terms  $t_i$  and  $t_j$  in  $S$  and one of the four basic operations, and form a new term by combining terms  $t_i$  and  $t_j$  with the operation. Since we do not know which of the possible next states will lead to the optimal

goal state, we enumerate all possible next states (that is, enumerate all possible pairs of terms and all possible operations); we prune the beam to keep only the top  $k$  candidates. We terminate when all the states in the beam have exactly one term.

Once we have a top  $k$  list of candidate expression trees, we choose the highest scoring tree which satisfies the constraints. However, there might not be any tree in the beam which satisfies the constraints, in which case, we choose the top candidate in the beam. We use  $k = 200$  in our experiments.

In order to choose the value for the  $w_{\text{IRR}}$ , we search over the set  $\{10^{-6}, 10^{-4}, 10^{-2}, 1, 10^2, 10^4, 10^6\}$ , and choose the parameter setting which gives the highest accuracy on the training data.

### 6.4.2 Quantity Schema

In order to generalize across problem types as well as over simple manipulations of the text, it is necessary to train our system only with relevant information from the problem text. E.g., for the problem in example 6.2, we do not want to take decisions based on how Tom earned money. Therefore, there is a need to extract the relevant information from the problem text. To this end, we introduce the concept of a *quantity schema* which we extract for each quantity in the problem’s text. Along with the question asked, the quantity schemas provides all the information needed to solve most arithmetic problems.

A quantity schema for a quantity  $q$  in problem  $P$  consists of the following components.

1. **Associated Verb** For each quantity  $q$ , we detect the verb associated with it. We traverse up the dependency tree starting from the quantity mention, and choose the first verb we reach. We used the easy first dependency parser [Goldberg and Elhadad, 2010].
2. **Subject of Associated Verb** We detect the noun phrase, which acts as subject of the associated verb (if one exists).
3. **Unit** We use a shallow parser to detect the phrase  $p$  in which the quantity  $q$  is mentioned. All tokens of the phrase (other than the number itself) are considered as unit tokens. Also, if  $p$  is followed by the prepositional phrase “of” and a noun phrase (according to the shallow parser annotations), we also consider tokens from this second noun phrase as unit tokens. Finally, if no unit token can be extracted, we assign the unit of the neighboring quantities as the unit of  $q$  (following previous work [Hosseini et al., 2014]).
4. **Related Noun Phrases** We consider all noun phrases which are connected to the phrase  $p$  containing

quantity  $q$ , with NP-PP-NP attachment. If only one quantity is mentioned in a sentence, we consider all noun phrases in it as related.

5. **Rate** We determine whether quantity  $q$  refers to a *rate* in the text, as well as extract two unit components defining the rate. For example, “7 kilometers per hour” has two components “kilometers” and “hour”. Similarly, for sentences describing unit cost like “Each egg costs 2 dollars”, “2” is a rate, with units “dollars” and “egg”.

In addition to extracting the quantity schemas for each quantity, we extract the surface form text which poses the question. For example, in the question sentence, “How much will John have to pay if he wants to buy 7 oranges?”, our extractor outputs “How much will John have to pay” as the question.

### 6.4.3 Relevance Classifier

We train a binary SVM classifier to determine, given problem text  $P$  and a quantity  $q$  in it, whether  $q$  is needed in the numeric expression generating the solution. We train on gold annotations and use the score of the classifier as the scoring function  $\text{IRR}(\cdot)$ .

#### Features

The features are extracted from the quantity schemas and can be broadly categorized into three groups:

1. **Unit features:** Most questions specifically mention the object whose amount needs to be computed, and hence questions provide valuable clue as to which quantities can be irrelevant. We add a feature for whether the unit of quantity  $q$  is present in the question tokens. Also, we add a feature based on whether the units of other quantities have better matches with question tokens (based on the number of tokens matched), and one based on the number of quantities which have the maximum number of matches with the question tokens.
2. **Related NP features:** Often units are not enough to differentiate between relevant and irrelevant quantities. Consider the following:

Example 6.3
Problem : <i>There are 8 apples in a pile on the desk. Each apple comes in a package of 11. 5 apples are added to the pile. How many apples are there in the pile?</i>
Solution : $(8 + 5) = 13$

The relevance decision depends on the noun phrase “the pile”, which is absent in the second sentence. We add a feature indicating whether a related noun phrase is present in the question. Also, we add a feature based on whether the related noun phrases of other quantities have better match with the question. Extraction of related noun phrases is described in Section 6.4.2.

3. **Miscellaneous Features:** When a problem mentions only two quantities, both of them are usually relevant. Hence, we also add a feature based on the number of quantities mentioned in text.

We include pairwise conjunction of the above features.

#### 6.4.4 LCA Operation Classifier

In order to predict LCA operations, we train a multiclass SVM classifier. Given problem text  $P$  and a pair of quantities  $p_i$  and  $p_j$ , the classifier predicts one of the six labels described in Eq. 6.2. We consider the confidence scores for each label supplied by the classifier as the scoring function  $\text{PAIR}(\cdot)$ .

##### Features

We use the following categories of features:

1. **Individual Quantity features:** Dependent verbs have been shown to play significant role in solving addition and subtraction problems [Hosseini et al., 2014]. Hence, we add the dependent verb of the quantity as a feature. Multiplication and division problems are largely dependent on rates described in text. To capture that, we add a feature based on whether the quantity is a rate, and whether any component of rate unit is present in the question. In addition to these quantity schema features, we add selected tokens from the neighborhood of the quantity mention. Neighborhood of quantities are often highly informative of LCA operations, for example, “He got 80 more marbles”, the term “more” usually indicates addition. We add as features adverbs and comparative adjectives mentioned in a window of size 5 around the quantity mention.
2. **Quantity Pair features:** For a pair  $(q_i, q_j)$  we add features to indicate whether they have the same dependent verbs, to indicate whether both dependent verbs refer to the same verb mention, whether the units of  $q_i$  and  $q_j$  are the same and, if one of them is a rate, which component of the unit matches with the other quantity’s unit. Finally, we add a feature indicating whether the value of  $q_i$  is greater than the value of  $q_j$ .



3. **Question Features:** Finally, we add a few features based on the question asked. In particular, for arithmetic problems where only one operation is needed, the question contains signals for the required operation. Specifically, we add indicator features based on whether the question mentions comparison-related tokens (e.g., “more”, “less” or “than”), or whether the question asks for a rate (indicated by tokens such as “each” or “one”).

We include pairwise conjunction of the above features. For both classifiers, we use the Illinois-SL package <sup>1</sup> under default settings.

## 6.5 Experimental Results

	AI2		IL		CC	
	Relax	Strict	Relax	Strict	Relax	Strict
All features	<b>88.7</b>	<b>85.1</b>	<b>75.7</b>	<b>75.7</b>	60.0	25.8
No Individual Quantity features	73.6	67.6	52.0	52.0	29.2	0.0
No Quantity Pair features	83.2	79.8	63.6	63.6	49.3	16.5
No Question features	86.8	83.9	73.3	73.3	<b>60.5</b>	<b>28.3</b>

Table 6.1: Performance of LCA Operation classifier on the datasets AI2, IL and CC.

In this section, we evaluate the proposed method on publicly available datasets of arithmetic word problems. We evaluate separately the relevance and LCA operation classifiers, and show the contribution of various features. Lastly, we evaluate the performance of the full system, and quantify the gains achieved by the constraints.

### 6.5.1 Datasets

We evaluate our system on three datasets, each of which comprise a different category of arithmetic word problems.

1. **AI2 Dataset:** This is a collection of 395 addition and subtraction problems, released by [Hosseini et al., 2014]. They performed a 3-fold cross validation, with every fold containing problems

<sup>1</sup> <http://cogcomp.cs.illinois.edu/page/software.view/Illinois-SL>

from different sources. This helped them evaluate robustness to domain diversity. We follow the same evaluation setting.

2. **IL Dataset:** This is a collection of arithmetic problems released by [Roy et al., 2015]. Each of these problems can be solved by performing one operation. However, there are multiple problems having the same template. To counter this, we perform a few modifications to the dataset. First, for each problem, we replace the numbers and nouns with the part of speech tags, and then we cluster the problems based on unigrams and bigrams from this modified problem text. In particular, we cluster problems together whose unigram-bigram similarity is over 90%. We next prune each cluster to keep at most 5 problems in each cluster. Finally we create the folds ensuring all problems in a cluster are assigned to the same fold, and each fold has similar distribution of all operations. We have a final set of 562 problems, and we use a 5-fold cross validation to evaluate on this dataset.
3. **Commoncore Dataset:** In order to test our system’s ability to handle multi-step problems, we create a new dataset of multi-step arithmetic problems. The problems were extracted from *www.commoncoresheets.com*. In total, there were 600 problems, 100 for each of the following types:
  - (a) Addition followed by Subtraction
  - (b) Subtraction followed by Addition
  - (c) Addition and Multiplication
  - (d) Addition and Division
  - (e) Subtraction and Multiplication
  - (f) Subtraction and Division

This dataset had no irrelevant quantities. Therefore, we did not use the relevance classifier in our evaluations.

In order to test our system’s ability to generalize across problem types, we perform a 6-fold cross validation, with each fold containing all the problems from one of the aforementioned categories. This is a more challenging setting relative to the individual data sets mentioned above, since we are evaluating on multi-step problems, without ever looking at problems which require the same set of operations.

### 6.5.2 Relevance Classifier

Table 6.2 evaluates the performance of the relevance classifier on the AI2 and IL datasets. We report two accuracy values: Relax - fraction of quantities which the classifier got correct, and Strict - fraction of math problems, for which all quantities were correctly classified. We report accuracy using all features and then removing each feature group, one at a time.

	AI2		IL	
	Relax	Strict	Relax	Strict
All features	94.7	89.1	<b>95.4</b>	<b>93.2</b>
No Unit features	88.9	71.5	92.8	91.0
No NP features	<b>94.9</b>	<b>89.6</b>	95.0	91.2
No Misc. features	92.0	85.9	93.7	89.8

Table 6.2: Performance of Relevance classifier on the datasets AI2 and IL.

We see that features related to units of quantities play the most significant role in determining relevance of quantities. Also, the related NP features are not helpful for the AI2 dataset.

### 6.5.3 LCA Operation Classifier

Table 6.1 evaluates the performance of the LCA Operation classifier on the AI2, IL and CC datasets. As before, we report two accuracies - Relax - fraction of quantity pairs for which the classifier correctly predicted the LCA operation, and Strict - fraction of math problems, for which all quantity pairs were correctly classified. We report accuracy using all features and then removing each feature group, one at a time.

The strict and relaxed accuracies for IL dataset are identical, since each problem in IL dataset only requires one operation. The features related to individual quantities are most significant; in particular, the accuracy goes to 0.0 in the CC dataset, without using individual quantity features. The question features are not helpful for classification in the CC dataset. This can be attributed to the fact that all problems in CC dataset require multiple operations, and questions in multi-step problems usually do not contain information for each of the required operations.

## 6.5.4 Global Inference Module

Table 6.3 shows the performance of our system in correctly solving arithmetic word problems. We show the impact of various constraints, and also compare against previously best known results on the AI2 and IL datasets. We also show results using each of the two constraints separately, and using no constraints at all.

	AI2	IL	CC
All constraints	72.0	<b>73.9</b>	<b>45.2</b>
Positive constraint	<b>78.0</b>	72.5	36.5
Integral constraint	71.8	73.4	39.0
No constraint	77.7	71.9	29.6
[Hosseini et al., 2014]	77.7	-	-
[Roy et al., 2015]	-	52.7	-
[Kushman et al., 2014]	64.0	73.7	2.3

Table 6.3: Accuracy in correctly solving arithmetic problems. First four rows represent various configurations of our system. We achieve state of the art results in both AI2 and IL datasets.

The previously known best result in the AI2 dataset is reported in [Hosseini et al., 2014]. Since we follow the exact same evaluation settings, our results are directly comparable. We achieve state of the art results, without having access to any additional annotated data, unlike [Hosseini et al., 2014], who use labeled data for verb categorization. For the IL dataset, we acquired the system of [Roy et al., 2015] from the authors, and ran it with the same fold information. We outperform their system by an absolute gain of over 20%. We believe that the improvement was mainly due to the dependence of the system of [Roy et al., 2015] on lexical and neighborhood of quantity features. In contrast, features from quantity schemas help us generalize across problem types. Finally, we also compare against the template based system of [Kushman et al., 2014]. [Hosseini et al., 2014] mentions the result of running the system of [Kushman et al., 2014] on AI2 dataset, and we report their result here. For IL and CC datasets, we used the system released by [Kushman et al., 2014].

The integrality constraint is particularly helpful when division is involved, since it can lead to fractional answers. It does not help in case of the AI2 dataset, which involves only addition and subtraction problems. The role of the constraints becomes more significant in case of multi-step problems and, in particular, they contribute an absolute improvement of over 15% over the system without constraints on the CC dataset.

The template based system of [Kushman et al., 2014] performs on par with our system on the IL dataset. We believe that it is due to the small number of equation templates in the IL dataset. It performs poorly on the CC dataset, since we evaluate on unseen problem types, which do not ensure that equation templates in the test data will be seen in the training data.

### 6.5.5 Discussion

The leading source of errors for the classifiers are erroneous quantity schema extraction and lack of understanding of unknown or rare verbs. For the relevance classifier on the AI2 dataset, 25% of the errors were due to mistakes in extracting the quantity schemas and 20% could be attributed to rare verbs. For the LCA operation classifier on the same dataset, 16% of the errors were due to unknown verbs and 15% were due to mistakes in extracting the schemas. The erroneous extraction of accurate quantity schemas is very significant for the IL dataset, contributing 57% of the errors for the relevance classifier and 39% of the errors for the LCA operation classifier. For the operation classifier on the CC dataset, 8% of the errors were due to verbs and 16% were due to faulty quantity schema extraction. Quantity Schema extraction is challenging due to parsing issues as well as some non-standard rate patterns, and it will be one of the future work targets. For example, in the sentence, “How many 4-dollar toys can he buy?”, we fail to extract the rate component of the quantity 4.

## 6.6 Conclusion

In this chapter, we present a novel method for understanding and solving a general class of arithmetic word problems. Our approach can solve all problems whose solution can be expressed by a read-once arithmetic expression, where each quantity from the problem text appears at most once in the expression. We develop a novel theoretical framework, centered around the notion of monotone expression trees, and showed how this representation can be used to get a unique decomposition of the problem. This theory naturally leads to a computational solution that we have shown to uniquely determine the solution - determine the arithmetic operation between any two quantities identified in the text. This theory underlies our algorithmic solution - we develop classifiers and a constrained inference approach that exploits redundancy in the information, and show that this yields strong performance on several benchmark collections. In particular, our approach achieves state of the art performance on two publicly available arithmetic problem datasets and can support natural generalizations for quantitative reasoning over multiple quantities

in text. Specifically, our approach performs competitively on multistep problems, even when it has never observed the particular problem type before. The datasets used in this work are available for download at [http://cogcomp.cs.illinois.edu/page/resource\\_view/98](http://cogcomp.cs.illinois.edu/page/resource_view/98).

# Chapter 7

## Unit Dependency Graphs

### 7.1 Introduction

Most statistical word problem solvers till now (including the one described in the last chapter) were entirely data-driven. They depend on training examples to learn all the concepts needed for math word problem solving. However, often domain knowledge is needed for quantitative reasoning, which is not easy to learn from training data. Dimensional analysis is an example of this kind of domain knowledge. Applying the knowledge of dimensional analysis to the units of quantities can inform several quantitative inferences as exemplified below.

Let us look at the arithmetic word problem in Example 7.1. The units of “66” and “10” are both “flowers”, which indicate they can be added or subtracted. Although unit of “8” is also “flower”, it is associated with a rate, indicating the number of flowers in each bouquet. As a result, “8” effectively has unit “flowers per bouquet”. Detecting such rate units and applying dimensional analysis help understand that “8” will more likely be multiplied or divided to arrive at the solution. Finally, the question asks for the number of “bouquets”, indicating “8” will likely be divided, and not multiplied. Knowing such interactions could help understand the situation and perform better quantitative reasoning. In addition, given that unit extraction is a noisy process, this can make it more robust via global reasoning. This is exactly the focus of this chapter – to integrate domain knowledge about dimensional analysis into statistical methods for word problem solving.

Example 7.1
Isabel picked 66 flowers for her friends wedding. She was making bouquets with 8 flowers in each one. If 10 of the flowers wilted before the wedding, how many bouquets could she still make?

We introduce the concept of *unit dependency graph* (UDG) for math word problems, to represent the relationships among the units of different numbers, and the question being asked. We also introduce a strategy

to extract annotations for unit dependency graphs, with minimal additional annotations. In particular, we use the answers to math problems, along with the rate annotations for a few selected problems, to generate complete annotations for unit dependency graphs. Finally, we develop a decomposed model to predict UDG given an input math word problem.

We augment the arithmetic word problem solver of [Roy and Roth, 2015] to predict a unit dependency graph, along with the solution expression of the input arithmetic word problem. Forcing the solver to respect the dependencies of the unit dependency graph enables us to improve unit extractions, as well as leverage the domain knowledge about unit dependencies in math reasoning. The introduction of unit dependency graphs reduced the error of the solver by over 10%, while also making it more robust to reduction in lexical and template overlap of the dataset.

## 7.2 Unit Dependency Graph

We first introduce the idea of a generalized rate, and its unit representation. We define **rate** to be any quantity which is some measure corresponding to one unit of some other quantity. This includes explicit rates like “40 miles per hour”, as well as implicit rates like the one in “Each student has 3 books”. Consequently, units for rate quantities take the form “*A per B*”, where A and B refer to different entities. We refer to A as Num Unit (short for Numerator Unit), and B as Den Unit (short for denominator unit). Table 7.1 shows examples of Num and Den Units for various rate mentions.

Mention	Num Unit	Den Unit
<i>40 miles per hour</i>	mile	hour
<i>Each student has 3 books.</i>	book	student

Table 7.1: Units of rate quantities

A unit dependency graph (UDG) of a math word problem is a graph representing the relations among quantity units and the question asked. Fig. 7.1 shows an example of a math word problem and its unit dependency graph. For each quantity mentioned in the problem, there exists a vertex in the unit dependency graph. In addition, there is also a vertex representing the question asked. Therefore, if a math problem mentions  $n$  quantities, its unit dependency graph will have  $n + 1$  vertices. In the example in Fig 7.1, there is one vertex corresponding to each of the quantities 66, 8 and 10, and one vertex representing the question part “how many bouquets could she still make ?”.



### Problem

Isabel picked 66 flowers for her friends wedding. She was making bouquets with 8 flowers in each one. If 10 of the flowers wilted before the wedding, how many bouquets could she still make?

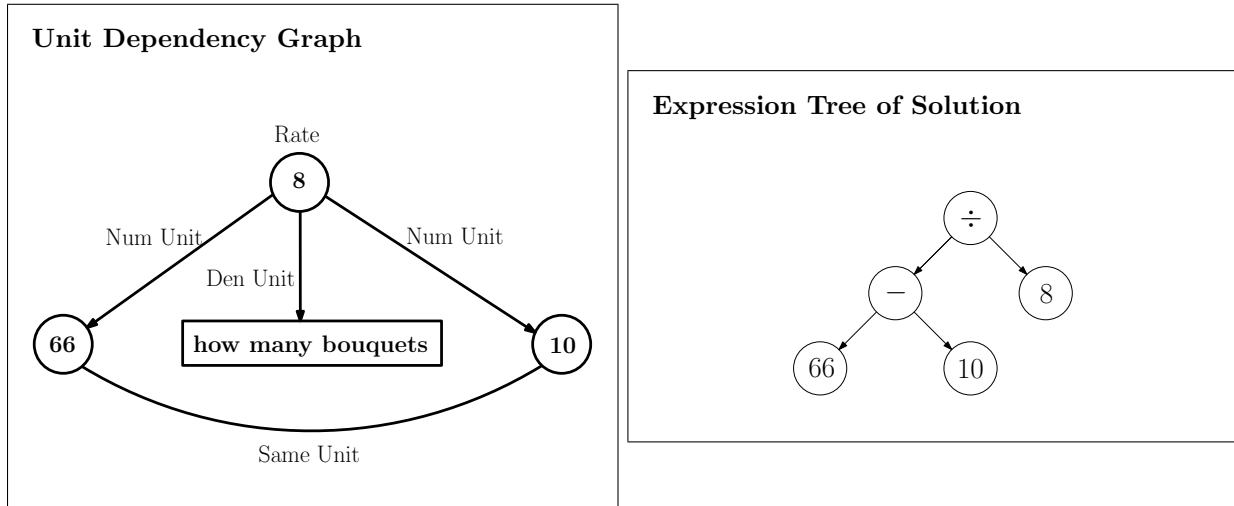


Figure 7.1: An arithmetic word problem, its UDG, and a tree representation of the solution  $(66 - 10)/8$ . Several dependencies exist between the UDG and the final solution of a problem. Here, “66” and “10” are connected via SAME UNIT edge, hence they can be added or subtracted, “8” is connected by DEN UNIT to the question, indicating that some expression will be divided by “8” to get the answer’s unit.

A vertex representing a number, is labeled RATE, if the corresponding quantity describes a rate relationship (according to the aforementioned definition). In fig 7.1, “8” is labeled as a RATE since it indicates the number of flowers in each bouquet. Similarly, a vertex corresponding to the question is marked RATE if the question asks for a rate.

Edges of a UDG can be directed as well as undirected. Each undirected edge has the label SAME UNIT, indicating that the connected vertices have the same unit. Each directed edge going from vertex  $u$  to vertex  $v$  can have one of the following labels:

1. **Num Unit** : Valid only for directed edges with source vertex  $u$  labeled as RATE, indicates that Num Unit of  $u$  matches the unit of the destination vertex  $v$ .
2. **Den Unit** : Valid only for directed edges with source vertex labeled as RATE, indicates that Den Unit of source vertex  $u$  matches the unit of the destination vertex  $v$ .

If no edge exists between a pair of vertices, they have unrelated units.

Several dependencies exist between the vertex and edge labels of the unit dependency graph of a problem,

and its solution expression. Sec 7.4.4 discusses these dependencies and how they can be leveraged to improve math problem solving.

## 7.3 Learning to Predict UDGs

Predicting UDG for a math word problem is essentially a structured prediction problem. However, since we have limited training data, we develop a decomposed model to predict parts of the structure independently, and then perform joint inference to enforce coherent predictions. This has been shown to be an effective method for structured prediction in the presence of limited data [Punyakanok et al., 2005b, Sutton and McCallum, 2007]. Empirically, we found our decomposed model to be superior to jointly trained alternatives (see Section 7.5).

Our decomposed model for UDG prediction uses the following two classifiers.

1. **Vertex Classifier** : This is a binary classifier, which takes a vertex of the UDG as input, and decides whether it denotes a rate.
2. **Edge Classifier** : This is a multiclass classifier, which takes as input a pair of nodes of the UDG, and predicts the properties of the edge connecting those nodes.

Finally, a constrained inference module combines the output of the two classifiers to construct a UDG. We provide details of the components in the following subsections.

### 7.3.1 Vertex Classifier

In order to detect rate quantities, we train a binary classifier. Given problem text  $P$  and a vertex  $v$  of the UDG, the classifier predicts whether  $v$  represents a rate. It predicts one of two labels - RATE or NOT RATE. The vertex  $v$  is either a quantity mentioned in  $P$ , or the question of  $P$ . The features used for the classification are as follows :

1. **Context Features**: We add unigrams, bigrams, part of speech tags, and their conjunctions from the neighborhood of  $v$ .
2. **Rule based Extraction Features**: We add a feature indicating whether a rule based approach can detect  $v$  as a rate.

### 7.3.2 Edge Classifier

We train a multiclass classifier to determine the properties of the edges of the UDG. Given problem text  $P$  and a pair of vertices  $v_i$  and  $v_j$  ( $i < j$ ), the classifier predicts one of the six labels :

1. SAME UNIT : Indicates that  $v_i$  and  $v_j$  should be connected by an undirected edge labeled SAME UNIT.
2. NO RELATION : Indicates no edge exists between  $v_i$  and  $v_j$ .
3.  $\text{RATE}_{Num}^{\rightarrow}$  : Indicates that  $v_i$  is a rate, and the Num Unit of  $v_i$  matches the unit of  $v_j$ .
4.  $\text{RATE}_{Num}^{\leftarrow}$  : Indicates that  $v_j$  is a rate, and the Num Unit of  $v_j$  matches the unit of  $v_i$ .
5. We similarly define  $\text{RATE}_{Den}^{\rightarrow}$  and  $\text{RATE}_{Den}^{\leftarrow}$ .

The features used for the classification are :

1. **Context Features:** For each vertex  $v$  in the query, we add the context features described for Vertex classifier.
2. **Rule based Extraction Features:** We add a feature indicating whether each of the queried vertices is detected as a rate by the rule based system. In addition, we also add features denoting whether there are common tokens in the units of  $v_i$  and  $v_j$ .

### 7.3.3 Constrained Inference

Our constrained inference module takes the scores of the Vertex and Edge classifiers, and combines them to find the most probable unit dependency graph for a problem. We define  $\text{VERTEX}(v, l)$  to be the score predicted by the Vertex classifier for labeling vertex  $v$  of a UDG with label  $l$ , where  $l \in \{\text{RATE}, \text{NOT RATE}\}$ . Similarly, we define  $\text{EDGE}(v_i, v_j, l)$  to be the score predicted by the Edge classifier for the assignment of label  $l$  to the edge between  $v_i$  and  $v_j$ . Here the label  $l$  is one of the six labels defined for the edge classifier.

Let  $G$  be a UDG with vertex set  $V$ . We define the score for  $G$  as follows:

$$\text{SCORE}(G) = \sum_{\substack{v \in V \\ \text{LABEL}(G, v) = \text{RATE}}} \text{VERTEX}(v, \text{RATE}) + \lambda \times \sum_{v_i, v_j \in V, i < j} \text{EDGE}(v_i, v_j, \text{LABEL}(G, v_i, v_j))$$

where  $\lambda$  is a scaling factor, and LABEL maps labels of the UDG, to the labels of the corresponding classifiers. LABEL( $G, v$ ) maps to RATE, if  $v$  is a rate, otherwise it maps to NOT RATE. Similarly, if no edge exists

between  $v_i$  and  $v_j$ ,  $\text{LABEL}(G, v_i, v_j)$  maps to NO RELATION, if Num Unit of  $v_i$  matches the unit of  $v_j$ ,  $\text{LABEL}(G, v_i, v_j)$  maps to  $\text{RATE}_{\vec{Num}}$ , and so on. Finally, the inference problem has the following form:

$$\arg \max_{G \in \text{GRAPHS}} \text{SCORE}(G)$$

where GRAPHs is the set of all valid unit dependency graphs for the input problem.

## 7.4 Joint Inference With An Arithmetic Solver

In this section, we describe our joint inference procedure to predict both a UDG and the solution of an input arithmetic word problem. Our model is built on the arithmetic word problem solver of [Roy and Roth, 2015], and we briefly describe it in the following sections. We first describe the concept of expression trees, and next describe the solver, which leverages expression tree representation of the solutions.

### 7.4.1 Monotonic Expression Tree

An **expression tree** is a binary tree representation of a mathematical expression, where leaves represent numbers, and all non-leaf nodes represent operations. Fig 7.1 shows an example of an arithmetic word problem and the expression tree of the solution mathematical expression. A **monotonic expression tree** is a normalized expression tree representation for math expressions, which restricts the order of combination of addition and subtraction nodes, and multiplication and division nodes. The expression tree in Fig 7.1 is monotonic.

### 7.4.2 Arithmetic Word Problem Solver

We now describe the solver pipeline of [Roy and Roth, 2015]. Given a problem  $P$  with quantities  $q_1, q_2, \dots, q_n$ , the solver uses the following two classifiers.

1. **Irrelevance Classifier** : Given as input, problem  $P$  and quantity  $q_i$  mentioned in  $P$ , the classifier decides whether  $q_i$  is irrelevant for the solution. The score of this classifier is denoted as  $\text{IRR}(q)$ .
2. **LCA Operation Classifier** : Given as input, problem  $P$  and a pair of quantities  $q_i$  and  $q_j$  ( $i < j$ ), the classifier predicts the operation at the lowest common ancestor (LCA) node of  $q_i$  and  $q_j$ , in the solution expression tree of problem  $P$ . The set of possible operations are  $+$ ,  $-$ ,  $-_r$ ,  $\times$ ,  $\div$  and  $\div_r$  (the subscript  $r$  indicates reverse order). Considering only monotonic expression trees for the solution

makes this operation unique for any pair of quantities. The score of this classifier for operation  $o$  is denoted as  $LCA(q_i, q_j, o)$ .

The above classifiers are used to gather irrelevance scores for each number, and LCA operation scores for each pair of numbers. Finally, constrained inference procedure combines these scores to generate the solution expression tree.

Let  $\mathcal{I}(T)$  be the set of all quantities in  $P$  which are not used in expression tree  $T$ , and  $\lambda_{\text{IRR}}$  be a scaling parameter. The score  $\text{SCORE}(T)$  of an expression tree  $T$  is defined as:

$$\text{SCORE}(T) = \lambda_{\text{IRR}} \sum_{q \in \mathcal{I}(T)} \text{IRR}(q) + \sum_{q_i, q_j \notin \mathcal{I}(T)} \text{LCA}(q_i, q_j, \odot_{LCA}(q_i, q_j, T))$$

where  $\odot_{LCA}(q_i, q_j, T)$  denotes the operation at the lowest common ancestor node of  $q_i$  and  $q_j$  in monotonic expression tree  $T$ . Let  $\text{TREES}$  be the set of valid expressions that can be formed using the quantities in a problem  $P$ , and also give positive solutions. The inference algorithm now becomes:

$$\arg \max_{T \in \text{TREES}} \text{SCORE}(T)$$

### 7.4.3 Joint Inference

We combine the scoring functions of UDG prediction and the ones from the solver of [Roy and Roth, 2015], so that we can jointly predict the UDG and the solution of the problem. For an input arithmetic word problem  $P$ , we score tuples  $(G, T)$  (where  $G$  is a candidate UDG for  $P$ , and  $T$  is a candidate solution expression tree of  $P$ ) as follows :

$$\begin{aligned} \text{SCORE}(G, T) = & \lambda_{\text{IRR}} \sum_{q \in \mathcal{I}(T)} \text{IRR}(q) + \sum_{q_i, q_j \notin \mathcal{I}(T)} \text{LCA}(q_i, q_j, \odot_{LCA}(q_i, q_j, T)) + \\ & \lambda_{\text{VERTEX}} \sum_{\substack{v \in V \\ \text{LABEL}(G, v) = \text{RATE}}} \text{VERTEX}(v, \text{RATE}) + \lambda_{\text{EDGE}} \sum_{v_i, v_j \in V, i < j} \text{EDGE}(v_i, v_j, \text{LABEL}(G, v_i, v_j)) \end{aligned}$$

where  $\lambda_{\text{IRR}}$ ,  $\lambda_{\text{VERTEX}}$  and  $\lambda_{\text{EDGE}}$  are scaling parameters. This is simply a scaled addition of the scores for UDG prediction and solution expression generation. Finally, the inference problem is

$$\arg \max_{(G, T) \in \text{TUPLES}} \text{SCORE}(G, T)$$

where TUPLES is the set of all tuples  $(G, T)$ , such that  $G \in \text{GRAPHS}$ ,  $T \in \text{TREES}$ , and  $G$  is a consistent UDG for the solution tree  $T$ .

#### 7.4.4 Consistent Rate Unit Graphs

We have a set of conditions to check whether  $G$  is a consistent UDG for monotonic tree  $T$ . Most of these conditions are expressed in terms of  $\text{Path}(T, v_i, v_j)$ , which takes as input a pair of vertices  $v_i, v_j$  of the UDG  $G$ , and a monotonic expression tree  $T$ , and returns the following.

1. If both  $v_i$  and  $v_j$  are numbers, and their corresponding leaf nodes in  $T$  are  $n_i$  and  $n_j$  respectively, then it returns the nodes in the path connecting  $n_i$  and  $n_j$  in  $T$ .
2. If only  $v_i$  denotes a number (implying  $v_j$  represents the question), the function returns the nodes in the path from  $n_i$  to the root of  $T$ , where  $n_i$  is the corresponding leaf node for  $v_i$ .

For the unit dependency graph and solution tree  $T$  of Fig 7.1,  $\text{Path}(T, 66, 8)$  is  $\{-, \div\}$ , whereas  $\text{Path}(T, 8, \text{question})$  is  $\{\div\}$ . Finally, the conditions for consistency between a UDG  $G$  and an expression tree  $T$  are as follows:

1. If  $v_i$  is the only vertex labeled RATE and it is the question, there should not exist a path from some leaf  $n$  to the root of  $T$  which has only addition, subtraction nodes. If that exists, it implies  $n$  can be added or subtracted to get the answer, that is, the corresponding vertex for  $n$  in  $G$  has same unit as the question, and should have been labeled RATE.
2. If  $v_i$  is labeled RATE and the question is not, the path from  $n_i$  (corresponding leaf node for  $v_i$ ) to the root of  $T$  cannot have only addition, subtraction nodes. Otherwise, the question will have same rate units as  $v_i$ .
3. We also check whether the edge labels are consistent with the vertex labels using Algorithm 5, which computes edge labels of UDGs, given the expression tree  $T$ , and vertex labels. It uses heuristics like if a rate  $r$  is being multiplied by a non-rate number  $n$ , the Den Unit of  $r$  should match the unit of  $n$ , etc.

---

**Algorithm 5** EDGELABEL

---

**Input:** Monotonic expression tree  $T$ , vertex pairs  $v_i, v_j$ , and their corresponding vertex labels

**Output:** Label of edge between  $v_i$  and  $v_j$

```
1: path  $\leftarrow$  Path( $T, v_i, v_j$ )
2: CountMulDiv  $\leftarrow$  Number of Multiplication and Division nodes in path
3: if  $v_i$  and  $v_j$  have same vertex label, and CountMulDiv = 0 then
4:   return SAME UNIT
5: end if
6: if  $v_i$  and  $v_j$  have different vertex labels, and CountMulDiv = 1 then
7:   if path contains  $\times$  and  $v_i$  is RATE then
8:     return RATE $_{Den}^{\rightarrow}$ 
9:   end if
10:  if path contains  $\times$  and  $v_j$  is RATE then
11:    return RATE $_{Den}^{\leftarrow}$ 
12:  end if
13:  if path contains  $\div$  and  $v_i$  is RATE then
14:    return RATE $_{Num}^{\rightarrow}$ 
15:  end if
16:  if path contains  $\div_r$  and  $v_j$  is RATE then
17:    return RATE $_{Num}^{\leftarrow}$ 
18:  end if
19: end if
20: return Cannot determine edge label
```

---

These consistency conditions prevent the inference procedure from considering any inconsistent tuples. They help the solver to get rid of erroneous solutions which involve operations inconsistent with all high scoring UDGs.

Finally, in order to find the highest scoring consistent tuple, we have to enumerate the members of TUPLES, and score them. The size of TUPLES however is exponential in the number of quantities in the problem. As a result, we perform beam search to get the highest scoring tuple. We first enumerate the members of TREES, and next for each member of TREES, we enumerate consistent UDGs.

## 7.5 Experiments

In this section, we evaluate our proposed method, and quantify the gains achieved by using UDGs in solving arithmetic word problems.

### 7.5.1 Dataset

We detected several issues in the existing datasets for arithmetic word problem solvers, and how they were used for evaluation. The evaluation in the last chapter was done separately on different types of arithmetic problems. This does not capture how well the systems can distinguish between these different problem types. Datasets released by [Roy and Roth, 2015] and [Koncel-Kedziorski et al., 2015] mention irrelevant quantities in words, and only the relevant quantities are mentioned in digits. This removes the challenge of detecting extraneous quantities. Also, the folds for Commoncore dataset were created in an adversarial manner to force the systems to only test on unseen equation forms. This seems too harsh for template based systems, who search the output space based on previously seen equation forms.

In order to address the aforementioned issues, we pooled arithmetic word problems from all available datasets [Hosseini et al., 2014, Roy and Roth, 2015, Koncel-Kedziorski et al., 2015], and normalized all mentions of quantities to digits. We next prune problems such that there do not exist a problem pair with over 80% match of unigrams and bigrams. The threshold of 80% was decided manually by determining that problems with around 80% overlap are sufficiently different. We finally ended up with 831 problems. We refer to this dataset as **AllArith**.

We also create subsets of **AllArith** using the MAWPS system [Koncel-Kedziorski et al., 2016]. MAWPS can generate subsets of word problems based on lexical and template overlap. Lexical overlap is a measure of reuse of lexemes among problems in a dataset. High lexeme reuse allows for spurious associations between the problem text and a correct solution [Koncel-Kedziorski et al., 2015]. Evaluating on low lexical overlap subset of the dataset can show the robustness of solvers to lack of spurious associations. Template overlap is a measure of reuse of similar equation templates across the dataset. Several systems focus on solving problems under the assumption that similar equation templates have been seen at training time. Evaluating on low template overlap subset can show the reliance of systems on the reuse of equation templates. We create two subsets of 415 problems each - one with low lexical overlap called **AllArithLex**, and one with low template overlap called **AllArithTmpl**.

We report random 5-fold cross validation results on all these datasets. For each fold, we choose 20% of the training data as development set, and tune the scaling parameters on this set. Once the parameters are



set, we retrain all the models on the entire training data. We use a beam size of 200 in all our experiments.

### 7.5.2 Data Acquisition

In order to learn the classifiers for predicting vertex and edge labels for UDGs, we need annotated data. However, gathering vertex and edge labels for UDGs of problems, can be expensive. In this section, we show that vertex labels for a subset of problems, along with annotations for solution expressions, can be sufficient to gather high quality annotations for vertex and edge labels of UDGs.

Given an arithmetic word problem  $P$ , annotated with the monotonic expression tree  $T$  of the solution expression, we try to acquire annotations for the UDG of  $P$ . First, we try to determine the labels for the vertices, and next the edges of the graph.

We check if  $T$  has any multiplication or division node. If no such node is present, we know that all the numbers in the leaves of  $T$  have been combined via addition or subtraction, and hence, none of them describes a rate in terms of the units of other numbers. This determines that none of  $T$ 's leaves is a rate, and also, the question does not ask for a rate. If a multiplication or division node is present in  $T$ , we gather annotations for the numbers in the leaves of  $T$  as well as the question of  $P$ . Annotators were asked to mark whether each number represents a rate relationship, and whether the question in  $P$  asks for a rate. This process determines the labels for the vertices of the UDG. Two annotators performed these annotations, with an agreement of 0.94(kappa).

Once we have the labels for the vertices of the UDG, we try to infer the labels for the edges using Algorithm 5. When the algorithm is unable to infer the label for a particular edge, we heuristically label that edge to be NO RELATION.

The above process allowed us to extract high quality annotations for UDGs with minimal manual annotations. In particular, we only had to annotate vertex labels for 300 problems, out of the 831 problems in **AllArith**. Obviously some of the extracted NO RELATION edge labels are noisy; this can be remedied by collecting annotations for these cases. However, in this work, we did not use any manual annotations for edge labels.

Features	Vertex Classifier			Edge Classifier		
	AllArith	AllArithLex	AllArithTmpl	AllArith	AllArithLex	AllArithTmpl
All features	96.7	96.2	97.5	87.1	84.3	86.6
No rule based features	93.2	92.5	92.6	79.3	75.4	78.0
No context features	95.1	94.1	95.3	78.6	70.3	75.5

Table 7.2: Performance of system components for predicting vertex and edge labels for unit dependency graphs

### 7.5.3 UDG Prediction

Table 7.2 shows the performance of the classifiers and the contribution of each feature type. The results indicate that rule-based techniques are not sufficient for robust extraction, there is a need to take context into account. Table 7.3 shows the performance of our decomposed model (DECOMPOSE) in correctly predicting UDGs, as well as the contribution of constraints in the inference procedure. Having explicit constraints for the graph structure provides 3-5% improvement in correct UDG prediction.

We also compare against a jointly trained model (JOINT), which learns to predict all vertex and edge labels together. Note that JOINT also uses the same set of constraints as DECOMPOSE in the inference procedure, to ensure it only predicts valid unit dependency graphs. We found that JOINT does not outperform DECOMPOSE, while taking significantly more time to train. The worse performance of joint learning is due to: (1) search space being too large for the joint model to do well given our relatively small dataset size, and (2) our independent classifiers being good enough, thus supporting better joint inference. This tradeoff is strongly supported in the literature [Punyakanok et al., 2005b, Sutton and McCallum, 2007].

Note, that all these evaluations are based on noisy edges annotations. This was done to reduce further annotation effort. Also, less than 15% of labels were noisy (indicated by fraction of NO RELATION labels), which makes this evaluation reasonable.

	AllArith	AllArithLex	AllArithTmpl
DECOMPOSE	73.6	67.7	68.7
- constraints	70.9	62.9	65.5
JOINT	72.9	66.7	68.4

Table 7.3: Performance in predicting UDGs

#### 7.5.4 Solving Arithmetic Word Problems

Here we evaluate the accuracy of our system in correctly solving arithmetic word problems. We refer to our system as UNITDEP. We compare against the following systems:

1. LCA++ : System of [Roy and Roth, 2015] with feature set augmented by neighborhood features, and with only positive answer constraint. We found that augmenting the released feature set with context features, and removing the integral answer constraint, were helpful. Our system UNITDEP also uses the augmented feature set for Relevance and LCA operation classifiers, and only positive constraint for final solution value.
2. TEMPLATE : Template based algebra word problem solver of [Kushman et al., 2014].
3. SINGLEEQ : Single equation word problem solver of [Koncel-Kedziorski et al., 2015].

In order to quantify the gains due to vertex and edge information of UDGs, we also run two variants of UNITDEP - one with  $\lambda_{\text{VERTEX}} = 0$ , and one with  $\lambda_{\text{EDGE}} = 0$ . Table 7.4 shows the performance of these systems on AllArith, AllArithLex and AllArithTmpl.

System	AllArith	AllArithLex	AllArithTmpl
TEMPLATE	73.7	65.5	71.3
SINGLEEQ	60.4	51.5	51.0
LCA++	79.4	63.6	74.7
UNITDEP	<b>81.7</b>	<b>68.9</b>	<b>79.5</b>
$\lambda_{\text{VERTEX}} = 0$	80.3	67.2	77.1
$\lambda_{\text{EDGE}} = 0$	79.9	64.1	75.7

Table 7.4: Performance in solving arithmetic word problems

UNITDEP outperforms all other systems across all datasets. Setting either  $\lambda_{\text{VERTEX}} = 0$  or  $\lambda_{\text{EDGE}} = 0$  leads to a drop in performance, indicating that both vertex and edge information of UDGs assist in math problem solving. Note that setting both  $\lambda_{\text{VERTEX}}$  and  $\lambda_{\text{EDGE}}$  to 0, is equivalent to LCA++. SINGLEEQ performs worse than other systems, since it does not handle irrelevant quantities in a problem.

In general, reduction of lexical overlap adversely affects the performance of most systems. The reduction of template overlap does not affect performance as much. This is due to the limited number of equation templates found in arithmetic problems. Introduction of UDGs make the system more robust to reduction of both lexical and template overlap. In particular, they provide an absolute improvement of 5% in both AllArithLex and allArithTpl datasets (indicated by difference of LCA++ and UNITDEP results).

For the sake of completeness, we also ran our system on the previously used datasets, achieving 1% and 4% absolute improvements over LCA++, in the Illinois dataset [Roy et al., 2015] and the Commoncore dataset [Roy and Roth, 2015] respectively.

### 7.5.5 Discussion

Most of gains of UNITDEP over LCA++ came from problems where LCA++ was predicting an operation or an expression that was inconsistent with the units. A small gain (10%) also comes from problems where UDGs help detect certain irrelevant quantities, which LCA++ cannot recognize. Table 7.5 lists some of the examples which UNITDEP gets correct but LCA++ does not.

Most of the mistakes of UNITDEP were due to extraneous quantity detection (around 50%). This was followed by errors due to the lack of math understanding (around 23%). This includes comparison questions like “How many more pennies does John have?”.

Problem	LCA++	UNITDEP
At lunch a waiter had 10 customers and 5 of them didn't leave a tip. If he got \$3.0 each from the ones who did tip, how much money did he earn?	$10.0-(5.0/3.0)$	$3.0*(10.0-5.0)$
The schools debate team had 26 boys and 46 girls on it. If they were split into groups of 9, how many groups could they make?	$9*(26+46)$	$(26+46)/9$
Melanie picked 7 plums and 4 oranges from the orchard . She gave 3 plums to Sam . How many plums does she have now ?	$(7+4)-3$	$(7-3)$
Isabellas hair is 18.0 inches long. By the end of the year her hair is 24.0 inches long. How much hair did she grow?	$(18.0*24.0)$	$(24.0-18.0)$

Table 7.5: Examples of problems which UNITDEP gets correct, but LCA++ does not.

## 7.6 Conclusion

In this chapter, we introduce the concept of unit dependency graphs, to model the dependencies among units of numbers mentioned in a math word problem, and the question asked. The dependencies of UDGs help improve performance of an existing arithmetic word problem solver, while also making it more robust to low lexical and template overlap of the dataset. Code and dataset are available at [http://cogcomp.cs.illinois.edu/page/publication\\_view/804](http://cogcomp.cs.illinois.edu/page/publication_view/804).

## Chapter 8

# Mapping to Declarative Knowledge

### 8.1 Introduction

Understanding and solving math word problems involves understanding several mathematical concepts, as well as their interaction with the physical world. Consider the arithmetic word problem shown in Fig 8.1, targeted towards elementary school students. To solve the problem, one needs to understand that “apple pies” and “pecan pies” are parts of “pies”, and hence, the number of apple pies and pecan pies needs to be added to get the total number of pies. Similarly, detecting that “5” represents “the number of pies per row” and applying dimensional analysis or unit compatibility knowledge, helps us infer that the total number of pies needs to be divided by 5 to get the answer. Besides part-whole relationship and dimensional analysis, there are several other concepts needed to support the reasoning in word problems. Some of them involve understanding comparisons, transactions, as well as application of math or physics formulas. We refer to any such formulas or concepts as “math domain knowledge”.

In the last chapter, we show how dimensional analysis knowledge can be integrated into math problem solving. In this chapter, we introduce a more general framework to incorporate any “math domain knowledge” into word problem solving. For combining a pair of numbers or math sub-expressions, our method first predicts *the type of knowledge* that is needed for it (like subset relationship, dimensional analysis, etc), and then predicts a *declarative rule* under that knowledge type to infer the mathematical operation. We model the selection of declarative rules as a latent variable, which removes the need for expensive annotations for the intermediate steps of our method.

The proposed approach has some clear advantages compared to existing work on word problem solving. First, it provides a way to relax the tight coupling between the “language understanding” component and the “Math knowledge” component. This relaxation facilitates better generalization, as we show, and, opens the ways to incorporate additional, more sophisticated math knowledge from different domains. Second, it provides interpretability of the solution, without expensive annotations. Our method predicts a declarative

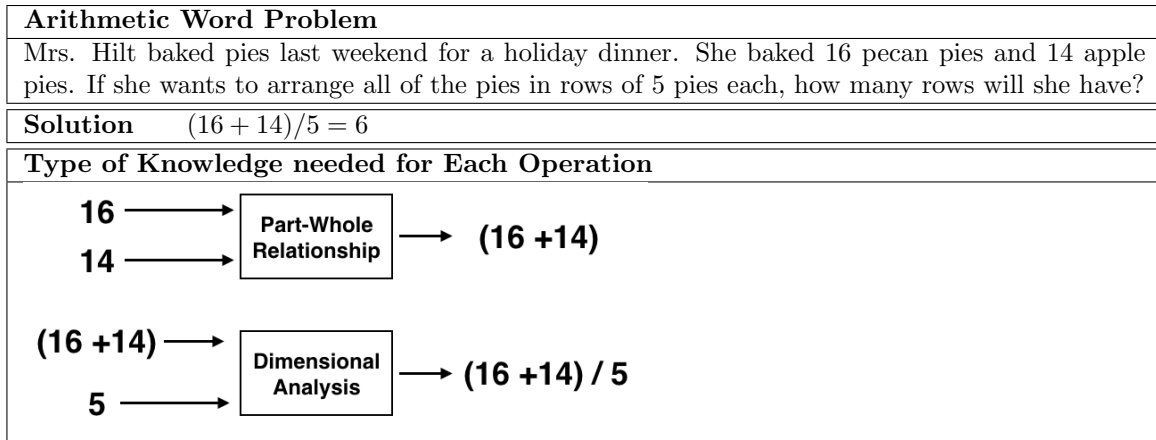


Figure 8.1: An example arithmetic word problem and its solution, along with the type of domain knowledge required to generate each operation of the solution

knowledge based inference rule for each operation needed in the solution. These rules provide an explanation for the operations performed. In particular, it learns to predict these rules without explicit annotations for them. Third, each individual operation in the solution expression can be generated independently by a separate knowledge type. This allows our method to use multiple knowledge types in the same problem.

We show that existing datasets of arithmetic word problems suffer from significant vocabulary biases and, consequently, existing solvers do not do well on conceptually similar problems that are not biased in the same way. Our method, on the other hand, learns the right abstractions even in the presence of biases in the data. We also introduce a novel approach to gather word problems without these biases, creating a new dataset of 1492 problems.

The next section introduces our representation of domain knowledge as well as the specific types of knowledge we use for arithmetic word problems. Section 8.3 describes our model to predict answers using domain knowledge, and provides details of our training paradigm. Finally, we provide experimental evaluation of our proposed method in Section 8.5, and then conclude with discussion of related work.

## 8.2 Knowledge Representation

We introduce our representation of domain knowledge in this section. We organize any knowledge hierarchically in two levels - knowledge types and declarative rules. A *knowledge type* is a concept or phenomenon which needs to be understood to apply reasoning over numbers. Examples of knowledge types include part-whole relations, dimensional analysis, etc. Under each knowledge type, there are a few declarative rules, which dictate which operation is needed in a particular context. An example of a declarative rule under

*part-whole knowledge type* can be that if two numbers quantify “parts” of a larger quantity, the operation between them must be addition. These rules often use knowledge type specific predicates, which we exemplify in the following subsections.

Since this work focuses on arithmetic word problems, we consider 4 knowledge types which are most common in these problems, as follows:

1. **Transfer:** This involves understanding the transfer of objects from one person to another. For example, the action described by the sentence “Tim gave 5 apples to Jim”, results in Tim losing “5 apples” and Jim gaining “5 apples”.
2. **Dimensional Analysis:** This involves understanding compatibility of units or dimensions of numbers. For example, “30 pies” can be divided by “5 pies per row” to get the number of rows.
3. **Part-Whole Relation:** This includes asserting that if two numbers quantify parts of a larger quantity, they are to be added. For example, the problem in Section ?? involves understanding “pecan pies” and “apple pies” are parts of “pies”, and hence must be added.
4. **Explicit Math:** Often word problems mention explicit math relationships among the quantities or entities in the problem. For example, “Jim is 5 inches taller than Tim”. This knowledge type captures the reasoning needed for such explicit math relationships.

Each of these knowledge types comprises a small number of declarative rules which determine the math operations; we describe them below.

### 8.2.1 Transfer

Consider the following excerpt of a word problem exhibiting a transfer phenomenon: “*Stephen owns 5 books. Daniel gave him 4 books.*” The goal of the declarative rules is to determine which operation is required between 5 and 4, given that we know that a transfer is taking place. We note that a transfer usually involves two entities, which occur as subject and indirect object in a sentence. Also, the direction of transfer is determined by the verbs associated with the entities. We define a set of variables to denote these properties; we define as Subj1, Verb1, IObj1 the subject, verb and indirect object associated with the first number, and as Subj2, Verb2, IObj2 the subject, verb and indirect object related to the second number. For the above example, the assignment of the variables are shown below:



[Stephen]<sub>Subj1</sub> [owns]<sub>Verb1</sub> 5 books. [Daniel]<sub>Subj2</sub> [gave]<sub>Verb2</sub> [him]<sub>IObj2</sub> 4 books.

In order to determine the direction of transfer, we require some classification of verbs. In particular, we classify each verb into one of five classes, namely HAVE, GET, GIVE, CONSTRUCT and DESTROY. The HAVE class constitutes all verbs which signify the state of an entity, such as “have”, “own”, etc. The GET class contains verbs which indicate the gaining of things for the subject. Examples of such verbs are “acquire”, “borrow”, etc. The GIVE class contains verbs which indicate the loss of things for the subject. Verbs like “lend”, “give” belong to this class. Finally CONSTRUCT class constitutes verbs indicating construction or creation, like “build”, “fill”, etc., while DESTROY indicates destruction related verbs like “destroy”, “eat”, “use”, etc. This verb classification is largely based on the work of [Hosseini et al., 2014].

Finally, the declarative rule applicable for our example has the following form:

$$[\text{Verb1} \in \text{HAVE}] \wedge [\text{Verb2} \in \text{GIVE}] \wedge [\text{Coref}(\text{Subj1}, \text{IObj2})] \Rightarrow \text{Addition}$$

where  $\text{Coref}(A, B)$  is true when  $A$  and  $B$  represent the same entity or are coreferent, and is false otherwise. Verb1 is “own” and hence  $[\text{Verb1} \in \text{HAVE}]$  is true. Verb2 is “give” and hence  $[\text{Verb2} \in \text{GIVE}]$  is true. Finally, Subj1 and IObj2 both refer to Stephen, so  $[\text{Coref}(\text{Subj1}, \text{IObj2})]$  returns true. As a result, the above declarative rule dictates that addition should be performed between 5 and 4.

We have 18 such inference rules for transfer, covering all combinations of verb classes and  $\text{Coref}()$  values. All these rules generate addition or subtraction operations.

## 8.2.2 Dimensional Analysis

We now look at the use of dimensional analysis knowledge in word problem solving. To use dimensional analysis, one needs to extract the units of numbers as well as the relations between the units of numbers. Consider the following excerpt of a word problem: “*Stephen has 5 bags. Each bag has 4 apples.*” Knowing that the unit of 5 is “bag” and the effective unit of 4 is “apples per bag”, allows us to infer that the numbers can be multiplied to obtain the total number of apples.

To capture these dependencies, we first introduce a few terms. Whenever a number has a unit of the form “A per B”, we refer to “A” as the unit of the number, and refer to “B” as *the rate component of the number*. In our example, the unit of 4 is “apple”, and the rate component of 4 is “bag”. We define variables Unit1 and Rate1 to denote the unit and the rate component of the first number respectively. We similarly

define Unit2 and Rate2. For the above example, the assignment of variables are shown below:

Stephen has 5 [bags]<sub>Unit1</sub>. Each [bag]<sub>Rate2</sub> has 4 [apples]<sub>Unit2</sub>.

Finally, the declarative rule applicable for our example has the following form:

[Coref(Unit1, Rate2)]  $\Rightarrow$  Multiplication

We have only 3 rules for dimensional analysis knowledge type. They generate multiplication or division operations.

### 8.2.3 Explicit Math

In this subsection, we want to capture the reasoning behind explicit math relationships expressed in word problems such as the one described in “*Stephen has 5 apples. Daniel has 4 more apples than Stephen*”. To detect such math relationships, we create a small list of patterns which indicate explicit math relationships, as well as, categorize these patterns based on the operation they imply. For example, the list for addition (ADD) has the patterns “more than”, “taller than”, “heavier than”, etc, and the list for subtraction (SUB) has patterns like “less than”, “shorter than”, etc. We define by *Math1* and *Math2* any explicit math term associated with the first and second numbers respectively. As was the case for transfers, we also define Subj1, IObj1, Subj2, and IObj2 to denote the entities participating in the math relationship. The assignment of these variables in our example is shown below.

[Stephen]<sub>Subj1</sub> has 5 apples. [Daniel]<sub>Subj2</sub> has 4 [more apples than]<sub>Math2</sub> [Stephen]<sub>IObj2</sub>.

Finally the declarative rule that applies is:

[Coref(Subj1, IObj2)]  $\wedge$  [Math2  $\in$  ADD]  $\Rightarrow$  Addition

We have only 7 rules for the explicit math knowledge type.

### 8.2.4 Part-Whole Relation

Understanding part-whole relationship entails understanding whether two quantities are hyponym, hypernym or siblings (that is, co-hyponym, or parts of the same quantity). For example, in the excerpt “*Mrs. Hilt*

has 5 pecan pies and 4 apple pies”, determining that pecan pies and apple pies are parts of all pies, helps inferring that addition is needed. We have 3 simple rules which directly map from Hyponym, Hypernym or Sibling detection to the corresponding math operation. For the above example, the applicable declarative rule is:

$$[\text{Sibling}(\text{Number1}, \text{Number2})] \Rightarrow \text{Addition}$$

The rules for part-whole knowledge type can generate addition and subtraction operations.

Note that all the declarative rules are designed to determine an operation between two numbers only. We introduce a strategy in Section 8.3, which facilitates combining sub-expressions with these declarative rules. A complete list of declarative rules is added to the appendix.

### 8.3 Mapping of Word Problems to Declarative Knowledge

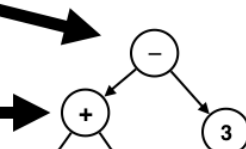
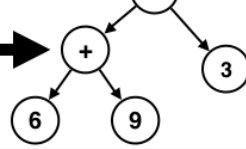
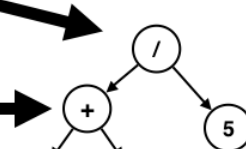
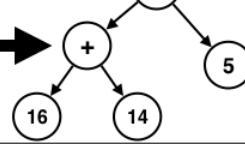
Word Problem	Tim 's cat had 6 kittens . He gave 3 to Jessica. Then Sara gave him 9 kittens . How many kittens does he now have ?
Knowledge based Answer Derivation	$6, 3 \Rightarrow \text{Transfer} \Rightarrow [\text{Verb1} \in \text{HAVE}] \wedge [\text{Verb2} \in \text{GIVE}] \wedge [\text{Coref}(\text{Subj1}, \text{Subj2})] \Rightarrow \text{Subtraction}$  $6, 9 \Rightarrow \text{Transfer} \Rightarrow [\text{Verb1} \in \text{HAVE}] \wedge [\text{Verb2} \in \text{GIVE}] \wedge [\text{Coref}(\text{Subj1}, \text{IObj2})] \Rightarrow \text{Addition}$ 
Word Problem	Mrs. Hilt baked pies last weekend for a holiday dinner. She baked 16 pecan pies and 14 apple pies. If she wants to arrange all of the pies in rows of 5 pies each, how many rows will she have?
Knowledge based Answer Derivation	$16, 5 \Rightarrow \text{Dimensional Analysis} \Rightarrow [\text{Coref}(\text{Unit1}, \text{Unit2})] \Rightarrow \text{Division}$  $16, 14 \Rightarrow \text{Part-Whole} \Rightarrow [\text{Sibling}(\text{Number1}, \text{Number2})] \Rightarrow \text{Addition}$ 

Table 8.1: Two examples of arithmetic word problems, and derivation of the answer. For each combination, first a knowledge type is chosen, and then a declarative rule from that knowledge type is chosen to infer the operation.

Given an input arithmetic word problem  $x$ , the goal is to predict the math expression  $y$ , which generates the correct answer. In order to derive the expression  $y$  from the word problem  $x$ , we leverage declarative

knowledge types and declarative rules that we introduced in Section 8.2. In order to combine two numbers mentioned in  $x$ , we first predict a domain knowledge type  $k$ , and then we choose a declarative knowledge rule  $r$  from  $k$ . The rule  $r$  generates the math operation needed to combine the two numbers. Consider the first example in Table 8.1. To combine 6 and 9, we first decide on the transfer knowledge type, and then choose an appropriate rule under transfer to generate the operation.

Next we need to combine the sub-expression  $(6 + 9)$  with the number 3. However, our inference rules were designed for the combination of two numbers only. In order to combine a sub-expression, we choose *a representative number* from the sub-expression, and use that number to determine the operation. In our example, we choose the number 6 as the representative number for  $(6 + 9)$ , and decide the operation between 6 and 3, following a similar procedure as before. This operation is now used to combine  $(6 + 9)$  and 3.

The representative number for a sub-expression is chosen such that it preserves the reasoning needed for the combination of this sub-expression with other numbers. We follow a simple heuristic to choose a representative number from a sub-expression:

1. For transfers and part-whole relationship, we choose the representative number of the left subtree.
2. In case of rate relationship, we choose the number which does not have a rate component.
3. In case of explicit math, we choose the number which is not directly associated with the explicit math expression.

### 8.3.1 Scoring Answer Derivations

Given the input word problem  $x$ , the solution math expression  $y$  is constructed by combining numbers in  $x$  with operations. We refer to the set of operations used in an expression  $y$  as  $\odot(y)$ . Each operation  $o$  in  $\odot(y)$  is generated by first choosing a knowledge type  $k^o$ , and then selecting a declarative rule  $r^o$  from that knowledge type.

In order to discriminate between multiple candidate solution expressions of a word problem  $x$ , we score them using a linear model over features extracted from the derivation of the solution. Our scoring function has the following form:

$$\text{SCORE}(x, y) = \sum_{o \in \odot(y)} w_k \phi_k(x, k^o) + w_r \phi_r(x, r^o)$$

where  $\phi_k(x, k^o)$  and  $\phi_r(x, r^o)$  are feature vectors related to knowledge type  $k^o$ , and declarative rule  $r^o$ ,

respectively, and  $w_k$  and  $w_r$  are the corresponding weight vectors. The term  $w_k \phi_k(x, k^o)$  is the score for the selection of  $k^o$ , and the term  $w_r \phi_r(x, r^o)$  is the score for the selection of  $r^o$ . Finally, the total score is the sum of the scores of all knowledge type and rule choices, over all operations of  $y$ .

### 8.3.2 Learning

We wish to estimate the parameters of the weight vectors  $w_k$  and  $w_r$ , such that our scoring function assigns a higher score to the correct math expression, and a lower score to other competing math expressions. For learning the parameters, we assume access to word problems paired with the correct math expression. We show in Section 8.4 that certain simple heuristics and rule component annotations can be used to create somewhat noisy annotations for knowledge types for operations. Hence, we will assume for our formulation, access to knowledge type supervision as well. We thus assume access to  $m$  examples of the following form:

$$\{(x_1, y_1, \{k^o\}_{o \in \odot(y_1)}), (x_2, y_2, \{k^o\}_{o \in \odot(y_2)}), \dots, (x_m, y_m, \{k^o\}_{o \in \odot(y_m)})\}.$$

We do not have any supervision for declarative rule selection, which we model as a latent variable.

**Two Stage Learning:** A straightforward solution for our learning problem could be to jointly learn  $w_k$  and  $w_r$  using latent structured SVM. However, we found that this model does not perform well. Instead, we chose a two stage learning protocol. At the first stage, we only learn  $w_r$ , the weight vector for scoring the declarative rule choice. Once learned, we fix the parameters for  $w_r$ , and then learn the parameters for  $w_k$ .

In order to learn the parameters for  $w_r$ , we solve the following optimization problem:

$$\min_{w_r} \frac{1}{2} \|w_r\|^2 + C \sum_{i=1}^m \sum_{o \in \odot(y_i)} \left[ \max_{\hat{r} \in k^o} w_r \cdot \phi_r(x, \hat{r}) - \max_{\hat{r} \in k^o, \hat{r} \Rightarrow o} w_r \cdot \phi_r(x, \hat{r}) \right]$$

where  $\hat{r} \in k^o$  implies that  $\hat{r}$  is a declarative rule of the knowledge type  $k^o$ , and  $\hat{r} \Rightarrow o$  signify that the declarative rule  $\hat{r}$  generates operation  $o$ . The above objective is similar to that of latent structured SVM. For each operation  $o$  in the solution expression  $y_i$ , the objective tries to minimize the difference between the highest scoring rule from its knowledge type  $k^o$ , and highest scoring rule from  $k^o$  which explains or generates the operation  $o$ .

Next we fix the parameters of  $w_r$ , and solve the following optimization:

$$\min_{w_k} \frac{1}{2} \|w_k\|^2 + C \sum_{i=1}^m \left[ \max_{y \in \mathcal{Y}} \text{SCORE}(x_i, y) - \text{SCORE}(x_i, y_i) \right]$$

This is equivalent to a standard structured SVM objective. Note that fixing the parameters of  $w_r$  determines the scores for rule selection, removing the need for any latent variables at this stage.

### 8.3.3 Inference

Given an input word problem  $x$ , inferring the best math expression involves computing the following:

$$\arg \max_{y \in \mathcal{Y}} \text{SCORE}(x, y)$$

where  $\mathcal{Y}$  is the set of all math expressions that can be created by combining the numbers in  $x$  with basic math operations.

The size of  $\mathcal{Y}$  is exponential in the number of quantities mentioned in  $x$ . As a result, it becomes computationally intractable to score each element of  $\mathcal{Y}$  and pick the highest scoring expression. We instead perform approximate inference using beam search.

We initialize the beam with the set  $E$  of all numbers mentioned in the problem  $x$ . At each step of the beam search, we choose two numbers (or sub-expressions)  $e_1$  and  $e_2$  from  $E$ , and then select a knowledge type and a declarative rule to infer an operation  $o$ . We create a new sub-expression  $e_3$  by combining the sub-expressions  $e_1$  and  $e_2$  with operation  $o$ . We finally create a new set  $E'$  from  $E$ , by removing  $e_1$  and  $e_2$  from it, and adding  $e_3$  to it. We remove  $E$  from the beam, and add all such modified sets  $E'$  to the beam. We continue this process till all sets in the beam have only one element in them. We choose the highest scoring expression among these elements as the solution expression.

## 8.4 Model and Implementation Details

We describe in this section several details of our model related to the supervision and features we use for training.

### 8.4.1 Supervision

Each word problem in our dataset is annotated with the solution math expression, along with alignment of numbers from the problem to the solution expression. In addition, we also have annotations for the numbers which possess a rate component. An example is shown in Fig 8.2.

<b>Problem:</b> Mrs. Hilt baked pies last weekend for a holiday dinner. She baked 16 pecan pies and 14 apple pies. If she wants to arrange all of the pies in rows of 5 pies each, how many rows will she have?
<b>Number List:</b> 16, 14, 5
<b>Solution:</b> $(16_{[1]} + 14_{[2]})/5_{[3]} = 6$
<b>Rates:</b> 5

Figure 8.2: Annotations in our dataset. Number List refers to the numbers detected in the problem. The subscripts in the solution indicate the position of the numbers in the number list.

This is the same level of supervision used in [Roy and Roth, 2017]. Many of the annotations can be extracted semi-automatically. The number list is extracted automatically by a number detector, the alignments require human supervision only when the same numeric value is mentioned multiple times in the problem. Most of the rate component annotations can also be extracted automatically, see [Roy and Roth, 2017] for details.

We apply a few heuristics to obtain noisy annotations for the knowledge types of operations. Consider the case for combining two numbers  $num1$  and  $num2$ , by operation  $o$ . We apply the following rules:

1. If we detect an explicit math pattern in the neighborhood of  $num1$  or  $num2$ , we assign knowledge type  $k^o$  to be Explicit Math.
2. If  $o$  is multiplication or division, and one of  $num1$  or  $num2$  has a rate component, we assign  $k^o$  to be Dimensional Analysis.
3. If  $o$  is addition or subtraction, we check if the dependent verb of both numbers are identical. If they are, we assign  $k^o$  to be Part-Whole relationship, otherwise, we assign it to be Transfer.

The annotations obtained via these rules are of course not perfect. However, we tested a small sample of these annotations, and found only around 3% of them to be wrong. As a result, we assume these annotations to be correct, and formulate our learning problem accordingly.

### 8.4.2 Features

We use two feature functions  $\phi_k(x, k^o)$  and  $\phi_r(x, r^o)$ , where  $x$  is the input word problem, and  $k^o$  and  $r^o$  are the knowledge type and the declarative rule for operation  $o$  respectively.  $\phi_r(x, r^o)$  constitutes the following features:

1. If  $r^o$  contains  $\text{Coref}(\cdot)$  function, we add features related to similarity of the arguments of  $\text{Coref}(\cdot)$ , for example, word overlap, whether they constitute a pronoun, etc.
2. For part-whole relationships, we add indicators for a list of words like “remaining”, “rest”, “either”, “overall”, “total”, conjoined with the part-whole function in  $r^o$  (Hyponymy, Hypernymy, Sibling).
3. Unigrams from the neighborhood of numbers being combined.

Finally,  $\phi_k(x, k^o)$  generates the following features:

1. If  $k^o$  is related to dimensional analysis, we add features indicating whether a rule based system detected a rate component in the combining numbers.
2. If  $k^o$  is part-whole, we add features indicating whether the verbs of combining numbers are identical.

Note that these features capture several interpretable functions like coreference, hyponymy, etc.

We do not learn three components of our system – verb classification for transfer knowledge, categorization of explicit math terms, and irrelevant number detection. For verb classification, we use a seed list of around 10 verbs for each category. Given a new verb  $v$ , we choose the most similar verb  $v'$  from the seed lists according to Glove vector [Pennington et al., 2014] based similarity . We assign  $v$  the category of  $v'$ . Note that this can be replaced by a learned component (as in [Hosseini et al., 2014]). However we found seed list based categorization to work well in most cases. For explicit math, we check for a small list of patterns to detect and categorize math terms. Note that for both the cases above, we still have to learn  $\text{Coref}(\cdot)$  function to determine the final operation. Finally, to detect irrelevant numbers (numbers which are not used in the solution), we use a set of rules based on the units of numbers. Again, this can be replaced by a learned model (as in [Roy and Roth, 2015]).

## 8.5 Experiments

In this section, we experimentally evaluate our proposed approach, and perform a detailed analysis of the results.

### 8.5.1 Results on Existing Dataset

We first evaluate our approach on the existing datasets of AllArith, AllArithLex, and AllArithTmpl



[Roy and Roth, 2017]. AllArithLex and AllArithTmpl are subsets of the AllArith dataset, created to test the robustness to new vocabulary, and new equation forms respectively. We compare to the top word problem solvers which can handle arithmetic word problems. They are as follows:

1. TEMPLATE : Template based algebra word problem solver of [Kushman et al., 2014].
2. LCA++ : System of [Roy and Roth, 2015] based on lowest common ancestors of math expression trees.
3. UNITDEP: Unit dependency graph based solver of [Roy and Roth, 2017].

We refer to our approach as KNOWLEDGE. The first few columns of Table 8.2 shows the performance of the systems on the aforementioned datasets. The performance of KNOWLEDGE is on par or lower than some of the existing systems.

We analyzed the performance of the systems, and found most of them to be not robust to perturbations of the problem text; Table 8.3 shows a few examples. We further analyzed the datasets, and identified several biases in the problems (in both train and test). Systems which remember these biases get an undue advantage in evaluation. For example, the verb “give” only appears with subtraction, and hence the models are learning an erroneous correlation of “give” with subtraction. Since the test also exhibit the same bias, these systems get all the “give”-related questions correct. However, they fail to solve the problem in Table 8.3, where “give” results in addition.

System	AllArith	AllArith Lex	AllArith Tmpl	Aggregate	Aggregate Lex	Aggregate Tmpl	Train on AllArith, Test on Perturb
TEMPLATE	71.96 ± 3.12	64.09 ± 5.56	70.64 ± 1.17	54.62 ± 2.32	45.05 ± 4.48	54.69 ± 2.22	24.2
LCA++	78.34 ± 2.59	66.99 ± 4.21	75.66 ± 4.72	65.21 ± 4.21	53.62 ± 5.01	63.0 ± 0.96	43.57
UNITDEP	<b>79.67</b> ± 1.95	71.33 ± 3.52	<b>77.11</b> ± 5.0	69.9 ± 4.57	57.51 ± 5.17	<b>68.64</b> ± 2.16	46.29
KNOWLEDGE	77.86 ± 1.27	<b>72.53</b> ± 2.99	74.7 ± 4.38	<b>73.32*</b> ± 2.31	<b>66.63*</b> ± 4.75	68.62 ± 4.98	<b>65.66*</b>

Table 8.2: Accuracy in solving arithmetic word problems. All columns except the last report 5-fold cross validation results. \* indicates statistically significant improvement ( $p = 0.05$ ) over second highest score in the column.

Problem	Systems which solved correctly	
	Trained on AllArith	Trained on Aggregate
Adam has 70 marbles . Adam gave 27 marbles to Sam . How many marbles does Adam have now ?	TEMPLATE, UNITDEP, LCA, KNOWLEDGE	LCA, UNITDEP, KNOWLEDGE
Adam has 70 marbles . Sam gave 27 marbles to Adam . How many marbles does Adam have now ?	KNOWLEDGE	TEMPLATE, KNOWLEDGE
Adam has 5 marbles . Sam has 6 more marbles than Adam . How many marbles does Sam have ?	LCA, UNITDEP, KNOWLEDGE	LCA, UNITDEP, KNOWLEDGE
Adam has 11 marbles . Adam has 6 more marbles than Sam . How many marbles does Sam have ?	TEMPLATE, KNOWLEDGE	TEMPLATE, KNOWLEDGE

Table 8.3: Pairs of perturbed problems, along with the systems which get them correct

### 8.5.2 New Dataset Creation

In order to remove the aforementioned biases from the dataset, we augment it with new word problems collected via a crowdsourcing platform. These new word problems are created by perturbing the original problems minimally, such that the answer is different from the original problem. For each word problem  $p$  with an answer expression  $a$  in our original dataset AllArith, we replace one operation in  $a$  to create a new math expression  $a'$ . We ask annotators to modify problem  $p$  minimally, such that  $a'$  is now the solution to the modified word problem. We create  $a'$  from  $a$  either by replacing an addition with subtraction or vice versa, or by replacing multiplication with division or vice versa. We do not replace addition and subtraction with multiplication or division, since there might not be an easy perturbation that supports this conversion. We manually pruned problems which did not yield the desired solution  $a'$ , or were too different from the input problem  $p$ . This procedure gave us a set of 661 new word problems, which we refer to as **Perturb**. Finally we augment **AllArith** with the problems of **Perturb**, and call this new dataset **Aggregate**. Aggregate has a total of 1492 problems.

The addition of **Perturb** problems ensures that the dataset now has problems with similar lexical items generating different answers. This minimizes the bias that we discussed in subsection 8.5.1. To quantify this, consider the probability distribution over operations for a quantity  $q$ , given that word  $w$  is present in the neighborhood of the number  $q$ . For an unbiased dataset, you will expect the entropy of this distribution to be high, since the presence of a single word in a number neighborhood will seldom be completely informative for the operation. We compute the average of this entropy value over all numbers and neighborhood words in our dataset. AllArith and Perturb have an average entropy of 0.34 and 0.32 respectively, whereas Aggregate’s average entropy is 0.54, indicating that, indeed, the complete data set is significantly less biased.

### 8.5.3 Generalization from Biased Dataset

First, we evaluate the ability of systems to generalize from biased datasets. We train all systems on AllArith, and test them on Perturb (which was created by perturbing AllArith problems). The last column of Table 8.2 shows the performance of systems in this setting. KNOWLEDGE outperforms all other systems in this setting with around 19% absolute improvement over UNITDEP. This shows that declarative knowledge allows the system to learn the correct abstractions, even from biased datasets.

### 8.5.4 Results on the New Dataset

Finally, we evaluate the systems on the Aggregate dataset. Following previous work [Roy and Roth, 2017], we compute two subsets of Aggregate comprising 756 problems each, using the MAWPS [Koncel-Kedziorski et al., 2016] system. The first, called AggregateLex, is one with low lexical repetitions, and the second called AggregateTmpl is one with low repetitions of equation forms. We also evaluate on these two subsets on a 5-fold cross-validation. Columns 4-6 of Table 8.2 show the performance of systems on this setting. KNOWLEDGE significantly outperforms other systems on Aggregate and AggregateLex, and is similar to UNITDEP on AggregateTmpl. There is a 9% absolute improvement on AggregateLex, showing that KNOWLEDGE is significantly more robust to low lexical overlap between train and test. The last column of Table 8.3 also shows that the other systems do not learn the right abstraction, even when trained on Aggregate.

### 8.5.5 Analysis

Table 8.4 shows examples of problems which KNOWLEDGE gets right, but UNITDEP does not. The gains can be attributed to the injection of declarative knowledge in our system. Earlier systems like UNITDEP try to learn the reasoning required for these problems from the data alone. This is often difficult in the presence of limited data, and noisy output from NLP tools. In contrast, we learn probabilistic models for interpretable functions like coreference, hyponymy, etc., and then use declarative knowledge involving these functions to perform reasoning. This considerably reduces the complexity of the target function to be learnt, and hence we end up with a more robust model.

A weakness of our method is the requirement to have all the relevant declarative knowledge during training. Many of the component functions (like coreference) are learnt through latent alignments with no explicit annotations. If too many problems are not explained by the declarative knowledge, the model will learn noisy alignments for the component functions.

Rachel was organizing her book case making sure each of the shelves had exactly 9 books on it. If she had 6 shelves of mystery books and 2 shelves of picture books, how many books did she have total?
Tim’s cat had kittens. He gave 3 to Jessica and 6 to Sara . He now has 9 kittens . How many kittens did he have to start with ?
Mrs. Snyder made 86 heart cookies. She made 36 red cookies, and the rest are pink. How many pink cookies did she make?

Table 8.4: Examples which KNOWLEDGE gets correct, but UNITDEP does not.

Table 8.5 shows the major categories of errors with examples. 26% of the errors are due to extraneous number detection. We use a rules based on units of numbers, to detect such irrelevant numbers. As a result, we fail to detect numbers which are irrelevant due to other factors, like associated entities, or associated verb. We can potentially expand our rule based system to detect those, or replace it by a learned module like [Roy and Roth, 2015]. Another major source of errors is parsing of rate components, that is, understanding “earns \$46 cleaning a home” should be normalized to “46\$ per home”. Although we learn a model for coreference function, we make several mistakes related to coreference. For the example in Table 8.5, we fail to detect the coreference between “team member” and “people”.

Irrelevant Number Detection (26%)	Sally had 39 baseball cards, and <u>9 were torn</u> . Sara bought 24 of Sally’s baseball cards . How many baseball cards does Sally have now ?
Parsing Rate Component (26%)	Mary earns <u>\$46 cleaning a home</u> . How many homes did she clean, if she made 276 dollars?
Coreference (22%)	There are <u>5 people</u> on the Green Bay High track team. If a relay race is 150 meters long, how far will each <u>team member</u> have to run?

Table 8.5: Examples of errors made by KNOWLEDGE

## 8.6 Related Work

Our work is primarily related to three major strands of research - automatic word problem solving, semantic parsing, as well as approaches incorporating background knowledge in learning.

**Automatic Word Problem Solving** There has been a growing interest in automatically solving math word problems, with various systems focusing on particular types of problems. [Hosseini et al., 2014] and

[Mitra and Baral, 2016] focus on addition, subtraction problems; [Roy and Roth, 2015], [Roy and Roth, 2017] as well as this work focus on arithmetic word problems; [Koncel-Kedziorski et al., 2015] looks at single equation problems, and finally [Kushman et al., 2014] handles general algebra word problems. Some of these approaches also try to incorporate some form of declarative or domain knowledge. [Hosseini et al., 2014] incorporates the transfer phenomenon by classifying verbs; [Mitra and Baral, 2016] maps problems to a set of formulas. Both require extensive annotations for intermediate steps (verb classification for [Hosseini et al., 2014], alignment of numbers to formulas for [Mitra and Baral, 2016], etc). In contrast, our method can handle a more general class of problems, while training only requires problem-equation pairs coupled with rate component annotations. [Roy and Roth, 2017] focus on only incorporating dimensional analysis knowledge, and handle the same class of problems as we do. In contrast, our method provides a general framework for incorporating any form of declarative knowledge, exemplified here by incorporating 4 types of domain knowledge.

**Semantic Parsing** Our work is also related to learning semantic parsers from indirect supervision [Clarke et al., 2010, Liang et al., 2011]. The general approach here is to learn mapping of sentences to logical forms, with the only supervision being the response of executing the logical form on a knowledge base. Similarly we learn to select declarative rules with only the final operation as supervision (and not which rule generated it). However, in contrast to the semantic parsing work, selection of each declarative rule usually requires reasoning across multiple sentences. Also, we do not require an explicit grounding of words or phrases to logical variables.

**Background Knowledge in Learning** Approaches to incorporate knowledge in learning started with Explanation based Learning (EBL) [DeJong, 1993, DeJong, 2014]. EBL uses domain knowledge based on observable predicates, whereas we learn to map text to predicates of our domain knowledge. More recent approaches tried to incorporate knowledge in the form of constraints on the output [Chang et al., 2007, Chang et al., 2012, Ganchev et al., 2010].

## 8.7 Conclusion

In this chapter, we introduce a framework for incorporating declarative knowledge in word problem solving. In order to combine numbers or math expressions, our system selects an appropriate knowledge type, and then an appropriate declarative rule that is used to infer the necessary math operation. The selection of the declarative rule for each operation provides interpretability for each operation selected. Our knowledge based approach outperforms all other systems, as well as learns better abstraction from biased datasets. Beyond better generalization, we believe that relaxing the coupling between the natural language component and

the “math” knowledge, also promises to eventually, simplify extending our approach to deal with a broader range of math phenomena. The current approach allows declarative rules that relate to two numbers or sub-expressions at a time. Future work will involve extending this approach to include rules involving multiple numbers or sub-expressions.

# Chapter 9

## Conclusion and Future Work

### 9.1 Summary

Quantities play an important role in everyday communication. Humans naturally perform a lot of quantitative reasoning while reading newspapers, discussing election results with friends, and planning their budget for a vacation, etc. As a result, quantitative reasoning is essential for complete language understanding. However, in spite of its abundance in everyday conversations and interactions, little work from NLP has focused on analyzing quantitative reasoning. In this thesis, we take the first few steps in that direction. We investigate several challenges for which standard NLP tools cannot be easily used for quantitative reasoning. We address each of these challenges, outlining several key technical ideas. Finally, we also build end to end robust quantitative reasoning systems, and make them available for the research community.

We begin by looking at quantities at a local level in Chapter 4. We develop a computational approach to extract quantities mentioned in free form text along with its related modifiers, and finally normalize the quantity to a standardized form. We show that correct detection and normalization can help perform several quantitative reasoning inferences, as exemplified by the Quantity Entailment task.

We investigate sentence level numeric relations in Chapter 5. An important challenge for capturing numeric relations is identifying the variables which participate in the relation. To capture this, we formulate the Equation Parsing task. We show that leveraging projectivity structures as well as using a pipeline of structured predictors can provide effective relation extraction systems. We also show that standard syntactic and semantic parsers cannot be effectively used in this domain, which motivates developing new approaches for these problems.

We develop a word problem solver for arithmetic problems in Chapter 6. The key idea is to effectively decompose the output structure in terms of the lowest common ancestors of expression trees. This decomposition allows us to perform reasoning for pairs of numbers, and effectively compose those pairwise answers, rather than considering all the numbers at a time. This helps us achieve state of the art results in arithmetic



word problem solving.

We address the challenge of incorporating domain knowledge in quantitative reasoning. First, to incorporate dimensional analysis knowledge, we introduce the concept of “unit dependency graphs”. We again show its effectiveness in the domain of arithmetic word problems. We show that learning to jointly predict unit dependency graphs along with the math solution, improves the robustness of the solver.

We further develop a general framework to integrate any declarative knowledge into math word problem solvers. We integrate several common math domain knowledge using this framework. We show that the knowledge based method learns robust models from limited data, as well as, learns the right abstraction from biased datasets.

These solutions address a wide range of challenges for quantitative reasoning. However there are a few limitations outlined in the following section.

## 9.2 Limitations

A key limitation of our math solver work is that we focus on arithmetic word problems. The assumption is that we can combine the numbers in the problem with basic operations to get to the answer. This is usually true for word problems seen in elementary school. However there are various other forms of word problems, which require building multiple equations with several variables. Unfortunately, considering multiple equations exponentially blows up the output space, which makes learning semantic structures from limited data impossible.

A similar limitation exists in our equation parsing work. We make a simplifying assumption that each sentence expresses a relation among at most two variables, and uses each of the numbers in the sentence at most once. Although we observed that most numeric relations follow this pattern, there are a few exceptions. Currently our model will not handle these cases, without some minor modifications.

## 9.3 Future Directions

The thesis shows directions to some promising areas of future research. We outline them in the following subsections.

### 9.3.1 Commonsense Quantitative Reasoning

The thesis looks at quantitative reasoning based on local neighborhood in quantity entailment, as well as across sentences in the context of math word problems. However there are several commonsense reasoning with respect to quantities, which are not captured by the tasks discussed in this thesis. The challenge is to acquire associations of several concepts with numbers. For example, “The water is hot” implies the temperature of the water must be a high value, “The water is slightly warm” implies that the temperature is a bit high, but not as high as the last case. Also, a sentence like “I could not afford the television, so I bought the radio.” implies that the cost of the television is higher than that of the radio. An interesting direction is to systematically study these inference problems. A reasonable approach can be to generalize the definition of the quantity entailment task to capture effects of the entire sentence, and not single quantities.

### 9.3.2 Unified Framework for Solvers

Currently, the arithmetic solvers that we have developed handles word problems from the elementary school level. An interesting direction is to use the ideas developed in this thesis to create solvers for math word problems from higher grade (like algebra word problems), or for word problems from other related domains (like physics problems). A research question here is that, is it necessary to always build a new system from scratch when we move to a new domain, or can we build some components based on our work, which are independent of the domain. The answer is likely to be affirmative. An effective approach can be to augment our knowledge based system with the appropriate inference rules for the particular domain. This approach has a good potential to generalize the work done in this thesis, and develop strategies to create solvers for any domain.

### 9.3.3 Automated Math Tutoring System

An interesting direction can be to develop automated math tutoring systems. Although there has been an increase in the number of available online courses, personalized feedback is usually lacking in these courses. However effective tutoring requires personalized interaction with students, and feedback on individual issues. An intelligent tutoring system can be the solution to this. Our work can be seen as a first step towards developing intelligent math tutoring systems. Moving ahead, we have to model the interaction with students, and how the solver can generate step by step explanations for the solution. Another direction is to automatically detect the weakness of individual students, and generate practice problems for them specifically targeting those weakness.

# References

- [Gun, 2015] (2015). 15 statistics that tell the story of gun violence this year. [Online; accessed 13-April-2017].
- [Syr, 2017] (2017). Car bomb hits syrian refugee camp on jordan border. [Online; accessed 13-April-2017].
- [Chi, 2017] (2017). Emanuel, allies spent at least \$22.8 million to win. [Online; accessed 10-April-2017].
- [Artzi and Zettlemoyer, 2013] Artzi, Y. and Zettlemoyer, L. (2013). UW SPF: The University of Washington Semantic Parsing Framework.
- [Bakman, 2007] Bakman, Y. (2007). Robust Understanding of Word Problems with Extraneous Information. *ArXiv Mathematics e-prints*.
- [Barwise and Cooper, 1981] Barwise, J. and Cooper, R. (1981). Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4(2):159–219.
- [Barzilay and Lapata, 2006] Barzilay, R. and Lapata, M. (2006). Aggregation via Set Partitioning for Natural Language Generation. In *Proc. of HLT/NAACL*.
- [Bengtson and Roth, 2008] Bengtson, E. and Roth, D. (2008). Understanding the value of features for coreference resolution. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- [Berant et al., 2014] Berant, J., Srikumar, V., Chen, P.-C., Linden, A. V., Harding, B., Huang, B., Clark, P., and Manning, C. D. (2014). Modeling biological processes for reading comprehension. In *Proceedings of EMNLP*.
- [Björkelund and Kuhn, 2014] Björkelund, A. and Kuhn, J. (2014). Learning structured perceptrons for coreference resolution with latent antecedents and non-local features. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 47–57, Baltimore, Maryland. Association for Computational Linguistics.
- [Bobrow, 1964] Bobrow, D. (1964). Natural language input for a computer problem solving system. Technical report, Cambridge, MA, USA.
- [Bundy et al., 1979] Bundy, A., Byrd, L., Luger, G., Mellish, C., Milne, R., and Palmer, M. (1979). *MECHO: A program to solve mechanics problems*. Department of Artificial Intelligence, University of Edinburgh.
- [Cai and Yates, 2013] Cai, Q. and Yates, A. (2013). Semantic Parsing Freebase: Towards Open-domain Semantic Parsing. In *Proceedings of the Second Joint Conference on Lexical and Computational Semantics (\*SEM)*.
- [Chang et al., 2015] Chang, K., Upadhyay, S., Chang, M., Srikumar, V., and Roth, D. (2015). Illinoisl: A JAVA library for structured prediction. *CoRR*, abs/1509.07179.

- [Chang et al., 2007] Chang, M., Ratinov, L., and Roth, D. (2007). Guiding semi-supervision with constraint-driven learning. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 280–287, Prague, Czech Republic. Association for Computational Linguistics.
- [Chang et al., 2010] Chang, M., Srikumar, V., Goldwasser, D., and Roth, D. (2010). Structured output learning with indirect supervision. In *Proc. of the International Conference on Machine Learning (ICML)*.
- [Chang et al., 2012] Chang, M.-W., Ratinov, L., and Roth, D. (2012). Structured learning with constrained conditional models. *Machine Learning*, 88(3):399–431.
- [Charniak, 1968] Charniak, E. (1968). Carps, a program which solves calculus word problems. Technical report, Cambridge, MA, USA.
- [Chen and Manning, 2014] Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *Empirical Methods in Natural Language Processing (EMNLP)*.
- [Clark, 2015] Clark, P. (2015). Elementary school science and math tests as a driver for ai: Take the aristo challenge! In *AAAI*.
- [Clarke et al., 2010] Clarke, J., Goldwasser, D., Chang, M., and Roth, D. (2010). Driving semantic parsing from the world’s response. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*.
- [Clarke and Lapata, 2006] Clarke, J. and Lapata, M. (2006). Constraint-based sentence compression: An integer programming approach. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 144–151, Sydney, Australia. ACL.
- [Collins, 2002] Collins, M. (2002). Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*.
- [Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20:273–297.
- [Dagan et al., 2006] Dagan, I., Glickman, O., and Magnini, B., editors (2006). *The PASCAL Recognising Textual Entailment Challenge.*, volume 3944. Springer-Verlag, Berlin.
- [Dagan et al., 2013] Dagan, I., Roth, D., Sammons, M., and Zanzoto, F. M. (2013). Recognizing textual entailment: Models and applications.
- [de Kleer, 1977] de Kleer, J. (1977). Multiples representations of knowledge in a mechanics problem-solver. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI’77*, pages 299–304.
- [de Marneffe et al., 2008] de Marneffe, M.-C., Rafferty, A. N., and Manning, C. D. (2008). Finding contradictions in text. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1039–1047, Columbus, Ohio. Association for Computational Linguistics.
- [DeJong, 1993] DeJong, G. (1993). *Investigating explanation-based learning.* Kluwer international series in engineering and computer science. Kluwer Academic Publishers.
- [DeJong, 2014] DeJong, G. (2014). Explanation-based learning. In Gonzalez, T., Diaz-Herrera, J., and Tucker, A., editors, *CRC computing handbook: Computer science and software engineering*, pages 66.1–66.26. CRC Press, Boca Raton.
- [Dellarosa, 1986] Dellarosa, D. (1986). A computer simulation of children’s arithmetic word-problem solving. *Behavior Research Methods, Instruments, & Computers*, 18(2):147–154.

- [Diane J. Briars, 1984] Diane J. Briars, J. H. L. (1984). An integrated model of skill in solving elementary word problems. *Cognition and Instruction*, 1(3):245–296.
- [Fletcher, 1985] Fletcher, C. R. (1985). Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods, Instruments, & Computers*, 17(5):565–571.
- [Forbus, 1984] Forbus, K. (1984). Qualitative process theory. *Artificial Intelligence*, 24:85–168.
- [Freund and Schapire, 1998] Freund, Y. and Schapire, R. (1998). Large margin classification using the Perceptron algorithm. In *Proceedings of the Annual ACM Workshop on Computational Learning Theory (COLT)*, pages 209–217.
- [Ganchev et al., 2010] Ganchev, K., Graça, J., Gillenwater, J., and Taskar, B. (2010). Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*.
- [Gelb, 1971] Gelb, J. P. (1971). Experiments with a natural language problem-solving system. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, IJCAI’71, pages 455–462.
- [Goldberg and Elhadad, 2010] Goldberg, Y. and Elhadad, M. (2010). An efficient algorithm for easy-first non-directional dependency parsing. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*.
- [Goldwasser and Roth, 2011] Goldwasser, D. and Roth, D. (2011). Learning from natural instructions. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*.
- [Hosseini et al., 2014] Hosseini, M. J., Hajishirzi, H., Etzioni, O., and Kushman, N. (2014). Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*.
- [Joachims et al., 2009] Joachims, T., Finley, T., and Yu, C.-N. (2009). Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59.
- [Kai-Wei Chang and Roth, 2013] Kai-Wei Chang, R. S. and Roth, D. (2013). A constrained latent variable model for coreference resolution. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- [Koncel-Kedziorski et al., 2015] Koncel-Kedziorski, R., Hajishirzi, H., Sabharwal, A., Etzioni, O., and Ang, S. (2015). Parsing Algebraic Word Problems into Equations. *TACL*.
- [Koncel-Kedziorski et al., 2016] Koncel-Kedziorski, R., Roy, S., Amini, A., Kushman, N., and Hajishirzi, H. (2016). Mawps: A math word problem repository. In *NAACL*.
- [Kuehne, 2004a] Kuehne, S. (2004a). On the representation of physical quantities in natural language text. In *Proceedings of Twenty-sixth Annual Meeting of the Cognitive Science Society*.
- [Kuehne, 2004b] Kuehne, S. (2004b). *Understanding natural language descriptions of physical phenomena*. PhD thesis, Northwestern University, Evanston, Illinois.
- [Kushman et al., 2014] Kushman, N., Zettlemoyer, L., Barzilay, R., and Artzi, Y. (2014). Learning to automatically solve algebra word problems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 271–281.
- [Kwiatkowski et al., 2013] Kwiatkowski, T., Choi, E., Artzi, Y., and Zettlemoyer, L. (2013). Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1545–1556, Seattle, Washington, USA. Association for Computational Linguistics.

- [Liang et al., 2011] Liang, P., Jordan, M. I., and Klein, D. (2011). Learning dependency-based compositional semantics. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
- [Maccartney and Manning, 2008] Maccartney, B. and Manning, C. D. (2008). Modeling semantic containment and exclusion in natural language inference. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 521–528, Manchester, UK. Coling 2008 Organizing Committee.
- [Madaan et al., 2016] Madaan, A., Mittal, A., Mausam, Ramakrishnan, G., and Sarawagi, S. (2016). Numerical relation extraction with minimal supervision. In *AAAI*.
- [Marshall, 1995] Marshall, S. P. (1995). *Schemas in problem solving*. Cambridge Univ Press.
- [McDonald et al., 2005] McDonald, R., Pereira, F., Ribarov, K., and Hajic, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics.
- [Miller et al., 1990] Miller, G., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1990). Wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–312.
- [Mitra and Baral, 2016] Mitra, A. and Baral, C. (2016). Learning to use formulas to solve simple arithmetic problems. In *ACL*.
- [Mukherjee and Garain, 2008] Mukherjee, A. and Garain, U. (2008). A review of methods for automatic understanding of natural language mathematical problems. *Artificial Intelligence Review*, 29(2):93–122.
- [Novak, 1976] Novak, G. S. (1976). *Computer understanding of physics problems stated in natural language*. PhD thesis, The University of Texas at Austin.
- [Oberem, 1987] Oberem, G. (1987). Albert: a physics problem solving monitor and coach. In *Proceedings of the first international conference on computer assisted learning (ICCAL87)*, ICCAL’87, pages 179–184.
- [Palmer et al., 2010] Palmer, M., Gildea, D., and Xue, N. (2010). *Semantic Role Labeling*, volume 3. Morgan & Claypool Publishers.
- [Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proc. of EMNLP*.
- [Punyakanok and Roth, 2001] Punyakanok, V. and Roth, D. (2001). The use of classifiers in sequential inference. In *Proc. of the Conference on Neural Information Processing Systems (NIPS)*, pages 995–1001. MIT Press.
- [Punyakanok et al., 2005a] Punyakanok, V., Roth, D., and Yih, W. (2005a). The necessity of syntactic parsing for semantic role labeling. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1117–1123.
- [Punyakanok et al., 2008] Punyakanok, V., Roth, D., and Yih, W. (2008). The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2).
- [Punyakanok et al., 2005b] Punyakanok, V., Roth, D., Yih, W., and Zimak, D. (2005b). Learning and inference over constrained output. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1124–1129.
- [Purdy, 1991] Purdy, W. (1991). A logic for natural language. *Notre Dame Journal of Formal Logic*, 32(3):409–425.

- [Roth and Yih, 2004] Roth, D. and Yih, W. (2004). A linear programming formulation for global inference in natural language tasks. In Ng, H. T. and Riloff, E., editors, *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*, pages 1–8. Association for Computational Linguistics.
- [Roth and Yih, 2005] Roth, D. and Yih, W. (2005). Integer linear programming inference for conditional random fields. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 737–744.
- [Roth and Zelenko, 1998] Roth, D. and Zelenko, D. (1998). Part of speech tagging using a network of linear separators. In *Coling-Acl, The 17th International Conference on Computational Linguistics*, pages 1136–1142.
- [Roy and Roth, 2015] Roy, S. and Roth, D. (2015). Solving general arithmetic word problems. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- [Roy and Roth, 2017] Roy, S. and Roth, D. (2017). Unit dependency graph and its application to arithmetic word problem solving. In *Proc. of the Conference on Artificial Intelligence (AAAI)*.
- [Roy et al., 2015] Roy, S., Vieira, T., and Roth, D. (2015). Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics (ACL)*, 3.
- [Sammons et al., 2010] Sammons, M., Vydiswaran, V., and Roth, D. (2010). Ask not what textual entailment can do for you... In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, Uppsala, Sweden. Association for Computational Linguistics.
- [Sarawagi and Cohen, 2004] Sarawagi, S. and Cohen, W. W. (2004). Semi-markov conditional random fields for information extraction. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 1185–1192.
- [Seo et al., 2014] Seo, M. J., Hajishirzi, H., Farhadi, A., and Etzioni, O. (2014). Diagram understanding in geometry questions. In *AAAI*.
- [Shalev-Shwartz et al., 2007] Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007). Pegasos: primal estimated sub-gradient solver for SVM. In Ghahramani, Z., editor, *Proceedings of the International Conference on Machine Learning (ICML)*, pages 807–814. Omnipress.
- [Sutton and McCallum, 2007] Sutton, C. and McCallum, A. (2007). Piecewise pseudolikelihood for efficient training of conditional random fields. In Ghahramani, Z., editor, *Proceedings of the International Conference on Machine Learning (ICML)*, pages 863–870. Omnipress.
- [Tsochantaridis et al., 2005] Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484.
- [Wong and Mooney, 2007] Wong, Y.-W. and Mooney, R. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 960–967, Prague, Czech Republic. Association for Computational Linguistics.
- [Yu and Joachims, 2009] Yu, C. and Joachims, T. (2009). Learning structural svms with latent variables. In *Proceedings of the International Conference on Machine Learning (ICML)*.

# Appendix A

## Lexicon for Equation Parsing

We construct a high precision list of rules, to parse sentences describing mathematical concepts, for example, “difference of”, “greater than”, etc for the equation parsing task described in Chapter 5. For each non-leaf node  $n$  of a projective equation tree, we define the following terms :

1.  $\text{MIDSPAN}(n)$  : The string from  $\min(\text{SPAN-END}(lc(n)), \text{SPAN-END}(rc(n)))$  to  $\max(\text{SPAN-START}(lc(n)), \text{SPAN-START}(rc(n)))$ .
2.  $\text{LEFTSPAN}(n)$  : The string ending at  $\min(\text{SPAN-START}(lc(n)), \text{SPAN-START}(rc(n)))$  and starting from the nearest trigger position on the left.
3.  $\text{RIGHTSPAN}(n)$  : The string starting at  $\max(\text{SPAN-END}(lc(n)), \text{SPAN-END}(rc(n)))$  and ending at the nearest trigger position on the right.
4.  $\text{LEFTTOKEN}(n)$  : Defined only for leaves, indicates the span of text for the trigger of  $n$ .

The rules in our lexicon are described using the above terms. They are as follows, ordered from low precedence to high precedence.

1. If  $\text{LEFTSPAN}(n)$  contains “sum of” and  $\text{MIDSPAN}(n)$  contains “and” or is the empty string,  $\odot(n)$  should be +.
2. If  $\text{MIDSPAN}(n)$  contains one of “added to”, “plus”, “more than” “taller than”, “greater than”, “larger than”, “faster than”, “longer than”, “increased”,  $\odot(n)$  should be +.
3. If  $\text{MIDSPAN}(n)$  contains one of “more than” “taller than”, “greater than”, “larger than”, “faster than”, “longer than”, and  $\text{RIGHTSPAN}(n)$  contains “by”,  $\odot(n)$  should be  $-$ , and  $\text{ORDER}(n)$  should be  $lr$ .
4. If  $\text{LEFTSPAN}(n)$  contains “difference of” and  $\text{MIDSPAN}(n)$  contains “and” or is the empty string,  $\odot(n)$  should be  $-$  and  $\text{ORDER}(n)$  should be  $lr$ .



5. If LEFTSPAN( $n$ ) contains one of “exceeds”, “minus”, “decreased”,  $\odot(n)$  should be  $-$ , and ORDER( $n$ ) should be  $lr$ .
6. If MIDSPAN( $n$ ) contains one of “subtracted” “shorter than”, “less than”, “slower than”, “smaller than”,  $\odot(n)$  should be  $-$ , and ORDER( $n$ ) should be  $rl$ .
7. If MIDSPAN( $n$ ) contains “multiplied by”,  $\odot(n)$  is  $\times$ .
8. If LEFTSPAN( $n$ ) contains “product of” and MIDSPAN( $n$ ) contains “and”,  $\odot(n)$  should be  $\times$ .
9. If LEFTSPAN( $n$ ) contains “ratio of”,  $\odot(n)$  should be  $\div$ , and ORDER( $n$ ) should be  $lr$ .
10. If LEFTTOKEN( $n$ ) contains one of “thrice”, “triple”, “twice”, “double”, “half”, or if MIDSPAN( $n$ ) contains “times”,  $\odot(n)$  is  $\times$ .
11. If LEFTTOKEN( $n$ ) contains one of “thrice”, “triple”, “twice”, “double”, “half”, or if MIDSPAN( $n$ ) contains “times”, and MIDSPAN( $n$ ) contains “as”, and RIGHTSPAN( $n$ ) contains “as”, operation at  $\odot(n)$  is  $\div$ , and ORDER( $n$ ) is  $rl$ .

# Appendix B

## Declarative Rules for Arithmetic Word Problems

Here, we list all the declarative rules for each knowledge type used in Chapter 8.

- **Transfer**

1.  $[\text{Verb1} \in \text{HAVE}] \wedge [\text{Verb2} \in \text{HAVE}] \wedge [\text{Coref}(\text{Subj1}, \text{Subj2})] \Rightarrow \text{Subtraction}$
2.  $[\text{Verb1} \in \text{HAVE}] \wedge [\text{Verb2} \in (\text{GET} \cup \text{CONSTRUCT})] \wedge [\text{Coref}(\text{Subj1}, \text{Subj2})] \Rightarrow \text{Addition}$
3.  $[\text{Verb1} \in \text{HAVE}] \wedge [\text{Verb2} \in (\text{GIVE} \cup \text{DESTROY})] \wedge [\text{Coref}(\text{Subj1}, \text{Subj2})] \Rightarrow \text{Subtraction}$
4.  $[\text{Verb1} \in (\text{GET} \cup \text{CONSTRUCT})] \wedge [\text{Verb2} \in \text{HAVE}] \wedge [\text{Coref}(\text{Subj1}, \text{Subj2})] \Rightarrow \text{Subtraction}$
5.  $[\text{Verb1} \in (\text{GET} \cup \text{CONSTRUCT})] \wedge [\text{Verb2} \in (\text{GET} \cup \text{CONSTRUCT})] \wedge [\text{Coref}(\text{Subj1}, \text{Subj2})] \Rightarrow \text{Addition}$
6.  $[\text{Verb1} \in (\text{GET} \cup \text{CONSTRUCT})] \wedge [\text{Verb2} \in (\text{GIVE} \cup \text{DESTROY})] \wedge [\text{Coref}(\text{Subj1}, \text{Subj2})] \Rightarrow \text{Subtraction}$
7.  $[\text{Verb1} \in (\text{GIVE} \cup \text{DESTROY})] \wedge [\text{Verb2} \in \text{HAVE}] \wedge [\text{Coref}(\text{Subj1}, \text{Subj2})] \Rightarrow \text{Addition}$
8.  $[\text{Verb1} \in (\text{GIVE} \cup \text{DESTROY})] \wedge [\text{Verb2} \in (\text{GET} \cup \text{CONSTRUCT})] \wedge [\text{Coref}(\text{Subj1}, \text{Subj2})] \Rightarrow \text{Subtraction}$
9.  $[\text{Verb1} \in (\text{GIVE} \cup \text{DESTROY})] \wedge [\text{Verb2} \in (\text{GIVE} \cup \text{DESTROY})] \wedge [\text{Coref}(\text{Subj1}, \text{Subj2})] \Rightarrow \text{Addition}$

We also have another rule for each rule above, which states that if  $\text{Coref}(\text{Subj1}, \text{Obj2})$  or  $\text{Coref}(\text{Subj2}, \text{Obj1})$  is true, and none of the verbs is CONSTRUCT or DESTROY, the final operation is changed from addition to subtraction, or vice versa. To determine the order of subtraction, we always subtract the smaller number from the larger number.

- **Dimensional Analysis**

1.  $[\text{Coref}(\text{Unit1}, \text{Rate2})] \wedge [\text{Coref}(\text{Unit2}, \text{Rate1})] \Rightarrow \text{Multiplication}$

2.  $[\text{Coref}(\text{Unit1}, \text{Unit2})] \wedge [\text{Rate2} \neq \text{null}] \Rightarrow \text{Division}$
3.  $[\text{Coref}(\text{Unit1}, \text{Unit2})] \wedge [\text{Rate1} \neq \text{null}] \Rightarrow \text{Division (Reverse order)}$

• **Explicit Math**

1.  $[\text{Coref}(\text{Subj1}, \text{IObj2}) \parallel \text{Coref}(\text{Subj2}, \text{IObj1})] \wedge [\text{Math1} \in \text{ADD} \parallel \text{Math2} \in \text{ADD}] \Rightarrow \text{Addition}$
2.  $[\text{Coref}(\text{Subj1}, \text{IObj2}) \parallel \text{Coref}(\text{Subj2}, \text{IObj1})] \wedge [\text{Math1} \in \text{SUB} \parallel \text{Math2} \in \text{SUB}] \Rightarrow \text{Subtraction}$
3.  $[\text{Coref}(\text{Subj1}, \text{Subj2})] \wedge [\text{Math1} \in \text{ADD} \parallel \text{Math2} \in \text{ADD}] \Rightarrow \text{Subtraction}$
4.  $[\text{Coref}(\text{Subj1}, \text{Subj2})] \wedge [\text{Math1} \in \text{SUB} \parallel \text{Math2} \in \text{SUB}] \Rightarrow \text{Addition}$
5.  $[\text{Coref}(\text{Subj1}, \text{Subj2})] \wedge [\text{Math1} \in \text{MUL}] \Rightarrow \text{Division (Reverse order)}$
6.  $[\text{Coref}(\text{Subj1}, \text{Subj2})] \wedge [\text{Math2} \in \text{MUL}] \Rightarrow \text{Division}$
7.  $[\text{Coref}(\text{Subj1}, \text{IObj2}) \parallel \text{Coref}(\text{Subj2}, \text{IObj1})] \wedge [\text{Math1} \in \text{MUL} \parallel \text{Math2} \in \text{MUL}] \Rightarrow \text{Multiplication}$

• **Part-Whole Relationship**

1.  $[\text{Sibling}(\text{Number1}, \text{Number2})] \Rightarrow \text{Addition}$
2.  $[\text{Hyponym}(\text{Number1}, \text{Number2})] \Rightarrow \text{Subtraction}$
3.  $[\text{Hypernym}(\text{Number1}, \text{Number2})] \Rightarrow \text{Subtraction}$