A MATHEMATICAL WORD PROBLEM SOLVER FOR ADDITION AND SUBTRACTION PROBLEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUSTAFA EROLCAN ER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COGNITIVE SCIENCE

SEPTEMBER 2020

A MATHEMATICAL WORD PROBLEM SOLVER FOR ADDITION AND SUBTRACTION PROBLEMS

submitted by **MUSTAFA EROLCAN ER** in partial fulfillment of the requirements for the degree of **Master of Science in Cognitive Science Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, **Graduate School of Informatics** _____

Prof. Dr. Cem Bozşahin
Head of Department, **Cognitive Science** _____

Prof. Dr. Cem Bozşahin
Supervisor, **Cognitive Science, METU** _____

Examining Committee Members:

Assist. Prof. Dr. Murat Perit Çakır
Cognitive Science, METU _____

Prof. Dr. Cem Bozşahin
Cognitive Science, METU _____

Assist. Prof. Dr. Burcu Can
Research Group of Computational Linguistics,
University of Wolverhampton _____

Date: 07 September 2020

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: MUSTAFA EROLCAN ER

Signature :

ABSTRACT

A MATHEMATICAL WORD PROBLEM SOLVER FOR ADDITION AND SUBTRACTION PROBLEMS

ER, MUSTAFA EROLCAN
MSc., Department of Cognitive Science
Supervisor: Prof. Dr. Cem Bozşahin

September 2020, 53 pages

Mathematical Word Problems are mathematical problems written in natural languages. These problems are crucial for mathematics education. This thesis aims to develop a model to solve addition and subtraction math word problems automatically. At first, an intermediate representation is introduced to mimic the human representation computationally. Dependency structures and part-of-speech tags are used to provide the auxiliary linguistic information about the problem. Circumscription idea is used to fill the blanks in intermediate representation. Then, an automata-based system has been designed to generate the equations for given problems with the help of domain-specific verb categories. The model has been developed and evaluated with different datasets. In conclusion chapter, possible future studies that may use our system were discussed.

Keywords: Mathematical Word Problems, Dependency Structures, POS Tagging, Educational Technology, Language Technologies

ÖZ

TOPLAMA VE ÇIKARMA PROBLEMLERİ İÇİN BİR MATEMATİKSEL KELİME PROBLEMİ ÇÖZÜCÜSÜ

ER, MUSTAFA EROLCAN
Yüksek Lisans, Bilişsel Bilimler Bölümü
Tez Yöneticisi: Prof. Dr. Cem Bozşahin

Eylül 2020 , 53 sayfa

Matematiksel Kelime Problemleri, doğal dilde yazılan matematik problemleridir. Bu problemler matematik eğitiminin önemli bir parçasıdır. Bu tez, toplama ve çıkarma matematiksel kelime problemlerini otomatik olarak çözecek bir model geliştirmeyi amaçlamaktadır. İlk olarak, insan temsilini taklit etmek amacıyla bir ara temsil tanıtılmaktadır. Bağımlılık yapıları ve sözcük türü etiketleri, problem hakkında yardımcı dilbilimsel bilgileri elde etmek için kullanılmaktadır. Çevreleme fikri, ara temsildeki boşlukları doldurmak için kullanılmaktadır. Daha sonra, otomata tabanlı bir sistem verilen problemler için alana özgü fiil kategorileri yardımıyla denklemler türetmek için tasarlanmıştır. Model, farklı verisetleri ile geliştirilmiş ve değerlendirilmiştir. Sonuç bölümünde, bizim sistemimizi kullanabilecek olası gelecek çalışmalar tartışılmaktadır.

Anahtar Kelimeler: Matematiksel Kelime Problemleri, Bağımlılık Yapıları, Sözcük Türü Etiketleri, Eğitim Teknolojisi, Dil Teknolojileri

To My Mother

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor Prof. Dr. Cem Bozşahin for his invaluable advices and supports. He expanded my horizon through his critical questions. I learned most of the things what I know about language technologies from him.

I would also express my gratitude to all faculty members of the cognitive science department. I learned a lot from them.

I am grateful to my mother for her support along this journey. This thesis would not exist without her support and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

MWP	Mathematical Word Problem
IR	Intermediate Representation
POS	Part of Speech
CL	Computational Linguistics
NLP	Natural Language Processing
HMM	Hidden Markov Model
MEMM	Maximum Entropy Markov Model

CHAPTER 1

INTRODUCTION

1.1 Thesis

Mathematics education is crucial both for societies and for individuals who want to have a good place in society. Some empirical results support this claim (Murnane et al., 2001).

However, this practical importance of mathematics education may not be well understood by the people. Mathematics education consisting of only symbols and formulas might push people to this idea. Thus, a considerable part of mathematics education in many parts of the world proceeds over math problems written in natural language. These math problems are referred to as Mathematical Word Problem in the literature. Problem texts narrated through the stories we may encounter in daily life are given at MWPs and mathematical concepts are expected to be used for their solutions.

Cognitive science is a discipline that tries to understand the concept of mind and how it works with the contributions of different disciplines. Computer science, linguistics, psychology, philosophy, and educational sciences are the major disciplines that cognitive science works with. Therefore, concept of MWP is a good candidate to be an experimental area for cognitive science with the disciplines it is connected to.

The aim of this thesis is exploring the internal mechanism of MWPs from cognitive science perspective. We will explore the minimum required linguistic information to express the mathematical equations from a story text. Then, we will develop an intermediate representation to model this internal mechanism in line with children's problem solving strategies.

Choosing the right representation system is critical in this domain. Even for humans, different representations might change the difficulty of same problem significantly. For instance, we can represent numbers with the Arabic numeric system or the Roman numeric system. Nevertheless, we cannot have the same power in arithmetic operations using these two representations. This difference is explained with an obvious example below:

It is easy to add, to subtract and even to multiply if the Arabic or binary representations are used, but it is not at all easy to do these things -especially multiplication- with Roman numerals. This is a key reason why the roman culture failed to develop mathematics in the way the earlier Arabic cultures had. (Marr, 1982)

In this thesis, we will develop a 6-tuple IR for MWPs. The main purpose of this representation method is to model the MWP components computationally. Then,

the answer for the MWP can be generated step by step compositionally. Although similar representation systems have been developed before, these systems are either too simple to capture the meaning of an MWP (Fletcher, 1985) or they are complex mechanisms with too many components (Koncel-Kedziorski et al., 2015). IR that we will introduce in this thesis fits MWPs better than previous systems.

1.2 Motivation

In this thesis, we have three motivations. First, we approach MWP solvers from the cognitive science perspective. We have analyzed MWP solving interviews with children from (Carpenter et al., 1983) and construct an intermediate representation for MWPs that mimics children's problem solving strategy. We will also show a potential application of this representation in the chapter 7.

Secondly, we use an approach similar to Hosseini et al. (2014) to generate an equation from an intermediate representation. It is called verb categorization. However, we implement a new algorithm to use verb categories in MWP pipeline. With the help of this approach, we do not set verb categories to one action. Actions will be chosen in accordance with other verbs' categories and their positions in the problem. Details will be explained in the chapter 4.

Lastly, we do not use any statistically trained model while generating problem representation from MWPs or generating the final equation from representation. The only two statistical models that we have used were POS tagging and dependency parsing model. These models are not our contributions in this thesis. These are auxiliary linguistic features.

All unique contributions in this thesis were designed as a rule-based model by hand. Nowadays, this is not a widely used approach for problem solvers. However, our primary aim in this thesis is to explore the internal mechanism of MWPs. Therefore, instead of using statistical models, we will examine human problem solving strategies and represent them.

1.3 Thesis Statement

This thesis introduces a complete problem solver for MWPs. This problem solver uses NLP algorithms to determine the linguistic attributes of raw problem text, and rule based procedures to generate IR from these raw texts. Finally, these IRs are processed by another set of procedures to generate the final answer automatically.

Our evaluation results show that our model using IR has an accuracy close to the state-of-the-art statistical models. However, statistical models tend to be overfitting because of the similarity between training dataset and test dataset. We separated our development dataset and test dataset carefully to avoid this problem. Thus, our model has more reliable results than the statistical models. This result and our unique IR mechanism are the main contributions of this thesis.

1.4 Outline

This thesis has six main chapters: Literature Review, Data, Implementation, Results, Discussion, Conclusion and Further Studies.

In the Literature Review chapter, we provide a background for our thesis. This chapter summarizes pioneer MWP solvers, dependency parsing, and part of speech tagging. In the Data chapter, we explain how to prepare development and test dataset for our system. The Implementation chapter describes our MWP solver. In the Results chapter, we evaluate our system's accuracy and compare this with other studies. Also, we will show that previous studies' datasets might cause overfitting in this chapter. We will discuss our motivations in the Discussion chapter. Finally, we conclude the thesis and talk about potential future studies in the last chapter.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

MWP is a mathematics education method that covers pure mathematical relations with the help of stories from real life. This method is available in most countries' school curriculum. In general, MWPs start with basic addition/subtraction problems in elementary schools aims to capture more difficult subjects like probability, logarithms, calculus within years.

The main purpose of practicing mathematics with word problems might be examined with two perspectives:

1. Showing that learning mathematics is useful in students' daily life.
2. Helping students to learn mathematical concepts better.

From the first perspective, some researchers think that giving a realistic background to mathematics might motivate the students to learn mathematics. Because they may understand why mathematics is important in their daily life. This perspective is explained below:

Word problems may also be a means of convincing students that they really will need the mathematics they are learning in school when they grow up and move into the world. Through experience in doing these problems, students should become confident that once they have learned the techniques, they will be able to apply them in the future not only in school but also out of school in real situations.
(Verschaffel et al., 2000)

Secondly, some researchers think that students are more familiar with daily life concepts than the pure mathematical world. Therefore, stories from daily life can make mathematics easier to understand for students.

Modeling aims, among other things, at providing students with a better comprehension of mathematical concepts, teaching them to formulate and to solve specific situation-problems, awaking their critical and creative senses, and shaping their attitude towards mathematics and their picture of it. (Blum et al., 2006)

Artificial intelligence is trying to model human intelligence and there are many attempts for automated MWP solvers in this area. In contrast to humans, computers are good at calculation tasks, while even easier MWPs are tough to be understood by computers. There are various reasons for this contradiction.

First, word problems are natural language texts. Thus, computer-based models have to cope with natural language understanding problems like ambiguities and coreference resolution. Therefore, we have to separate natural language texts related to mathematics into the two categories. These are mathematical texts and story texts.

Mathematical texts are explanatory texts for symbolic mathematics. Their meanings are not independent of their mathematical context.

The square root of 2 is irrational. Might be expressed more concisely as $\sqrt{2}$ *is irrational. Or even as* $\sqrt{2} \notin \mathbb{Q}$. (Ganesalingam, 2013)

According to Ganesalingam (2013), mathematical texts differ from other natural language texts in three ways. Two of them are also valid for our categorization:

1. Since mathematical texts are dealing with timeless facts, they do not need to consider tenses. Also, mathematical truths do not change in context. They are true in all possible worlds. Thus, there is no modality. Because of these reasons, mathematical texts are much more restricted than natural language.
2. Since mathematical assertions are either true or false, mathematical texts are free from vagueness. Also, mathematical texts do not include metaphor or figurative meaning. Thus, they can be captured by standard semantic representations easily.

These attributes of mathematical texts make them easier for linguistic analysis. However, MWPs are not these kinds of texts. They include stories from daily life. This fact makes MWPs more complex than other mathematical texts. Their meanings for stories might be independent of their mathematical meanings, and they might include metaphor or figurative meaning. Also, we need to consider the tenses while solving MWPs. Mathematical meanings of the problems may change in accordance with the tenses.

Also, there are some specific problems in understanding MWPs. Sentences of MWPs are dependent on each other more than other natural language texts. Thus, understanding the whole problem cannot be handled by understanding the components independently. Also, some sentences or pieces of information might be entirely irrelevant for solutions. Detecting them is not easy for computers.

(2.1) "There are 9 pencils and 4 rulers in the drawer. Sally took 4 pencils out of the drawer. How many pencils are there now?"

In the example taken from Hosseini et al. (2014) above, the number of rulers in the drawer is redundant information for the solution. A successful MWP solver should handle this.

MWPs are story texts in accordance with our classification explained above. Thus, they are complex problems from computational perspective. In this thesis, we will develop a simple mechanism to solve these complex problems as suggested by Bozşahin (2018).

Although designing an MWP solver is a challenging task, there are various studies for MWP solver systems since the 1960s. In the next two sections, we will mention some pioneer studies in the literature briefly.

2.2 Early Attempts for Mathematical Word Problem Solvers

The first attempt for an automatic MWP solver was the STUDENT which is developed by Bobrow (1964). This system was programmed in LISP. Behind the STUDENT, a simple pattern matching algorithm was working. The system converted MWPs to kernel sentences via using keywords in MWPs. Then, keywords like times, square, add were mapping to their mathematical symbols. As a final step, this kernel sentence was processed to find the final answer.

After STUDENT, Charniak (1968) applied a similar idea to the rate problems. He developed an MWP solver in LISP that is called CARPS. In CARPS, the input problem is converted to simple sentences with basic pattern matching rules. After this step, simple sentences were translated into an intermediate representation, which was a hierarchical tree that represented the problem structurally. In the next step, an equation template to solve the problem were retrieved from memory. Finally, the equation for the input question was manipulated with pieces of information from intermediate representation. Even though it was one of the first MWP solvers, CARPS could solve only 14 rate problems. Thus, it does not have a wide application area.

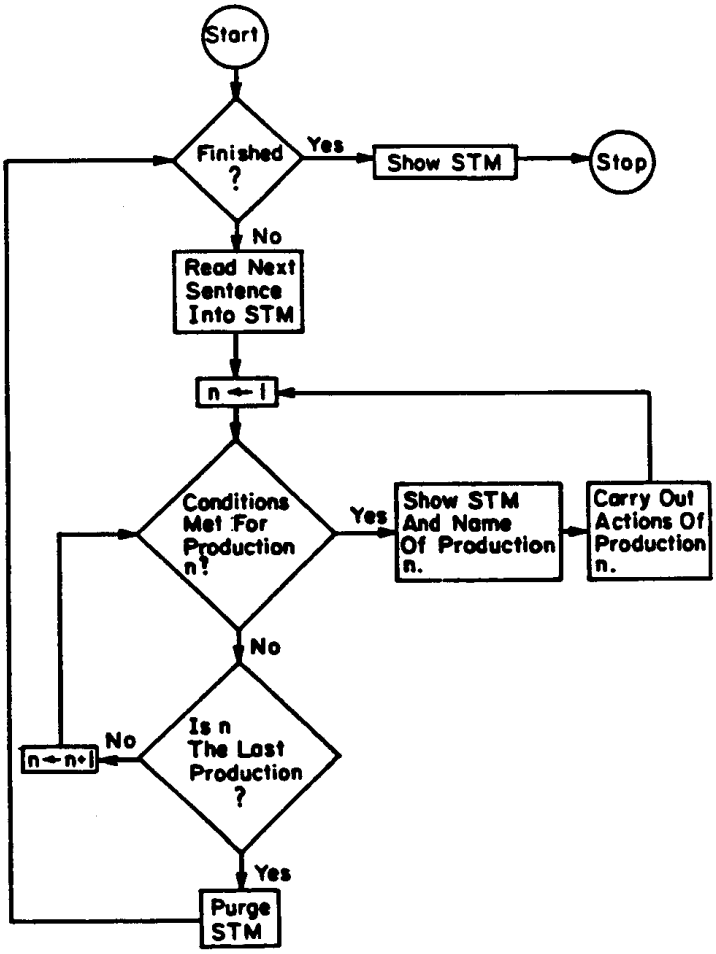

Figure 2.1: Basic Control Structure of WORDPRO.

In 1985, WORDPRO was developed by Fletcher (1985). This system was taking inputs as a set of propositions instead of raw problem text. The system was converting these propositions to an intermediate representation with three slots. These slots were object slot, quantity slot, and specification slot. As a next step, WORDPRO was checking the rule list (STM) to find the matched rules with the current proposition. If a rule were found, the action part of the rule was carried out via the pieces of information from intermediate representation. This system could solve only 14 specific questions too. Thus, this is a very narrow and ad-hoc solution.

The last entirely rule based MWP solver as far as we know was ROBUST (Bakman, 2007). ROBUST could handle more problem types than WORDPRO and STUDENT. This system was using change schemas to generate the answer. These schemas taken from Bakman (2007) are shown in Table 2.1:

Table 2.1: Change schemas for ROBUST.

Name of Change Schema	Formula Associated with the Schema
Transfer-In-Place	There were X objects in the place. Y objects were transferred into the place. There are Z objects in the place now.
Transfer-Out-Place	There were X objects in the place. Y objects were transferred out of the place. There are Z objects in the place now.
Transfer-In-Ownership	The owner had R objects. The owner received S objects. The owner has T objects now.
Transfer-Out-Ownership	The owner had R objects. The owner forfeited S objects. The owner has T objects now.
Creation (Ownership)	The owner had R objects. The owner created S objects. The owner has T objects now.
Creation (Place)	There were X objects in the place. Y objects were created in the place. There are Z objects in the place now.
Termination (Ownership)	The owner had R objects. The owner terminated S objects. The owner has T objects now.
Termination (Place)	There were X objects in the place. Y objects were terminated in the place. There are Z objects in the place now.

In ROBUST, input problems were converted to propositions firstly. Also, change schemas were found in this step. Secondly, propositions were split into their components. In the final step, these components filled the schema slots and solutions were generated from these schemas. ROBUST also was the first system that tried to handle redundant sentences and information.

In addition to mathematics, there were similar attempts for mechanic and chemistry based word problems. (Novak Jr, 1976), (Bundy et al., 1979), (Novak Jr & Bulko, 1993) are some examples of them. We will not mention these studies' inner mechanisms here, but these studies are good examples of different applications of word problem solvers.

2.3 New Attempts for Mathematical Word Problem Solvers

After ROBUST, there was not a prominent study on MWPs for a while. However, two studies published in 2014 brought MWP solvers to the agenda of the NLP community again. These studies were (Hosseini et al., 2014) and (Kushman et al., 2014).

Hosseini et al. (2014) developed a three-stage MWP solver that is called ARIS. At first in ARIS, Stanford dependency parser (Chen & Manning, 2014), named entity recognizer (Finkel et al., 2005), POS tagger (Klein & Manning, 2003), and coreference resolution system (Recasens et al., 2013) was applied to the problem text to express relevant information like verbs, quantities, entities, and other linguistic attributes.

Secondly, a model was trained to learn verb categories. They used linear similarity (Lin et al., 1998), WordNet (Miller, 1995), and structural features as features of their machine learning model. As an output, all verbs in their dataset were mapped to one of seven verb categories. These categories are summarized in Table 2.2 below. In this thesis, we have used five of them for state transition.

Table 2.2: Different verb categories in sentences taken from ARIS.

Category	**Example**
Observation	There **were** 28 bales of hay in the *barn*.
Positive	*Joan* **went** to 4 football games this year.
Negative	*John* **lost** 3 of the violet balloons.
Positive Transfer	*Mike's dad* **borrowed** 7 nickels from *Mike*.
Negative Transfer	*Jason* **placed** 131 erasers in the *drawer*.
Construct	*Karen* **added** 1/4 of a cup of walnuts to a batch of trail mix.
Destroy	The *rabbits* **ate** 4 of *Dan's* potatoes.

In the last step, all information from the first two steps were used to generate the final equation. ARIS used state progression in this step. For any MWP, states were constructed with pieces of information from the first step, and numbers on these states were updated by verb categories that were learned in the second step.

ARIS was one of the most successful and pioneer MWP solver for addition and subtraction problems. This system was evaluated on a dataset with 395 MWPs. Nowadays, this dataset is one of the most used dataset in the literature. It is also our test dataset in this thesis.

In the same year, Kushman et al. (2014) introduced an MWP solver that is called KAZB. KAZB processes input problems in two steps to generate an equation.

First, an equation template is chosen through a machine learning model to represent the input problem. Then, the template is filled with numeric values and nouns from the input problem. This model was the first wide-coverage MWP solver. The system could solve multiplication and division problems in addition to addition and subtraction problems.

Derivation 1	
Word problem	An amusement park sells 2 kinds of tickets. Tickets for children cost $ 1.50. Adult tickets cost $ 4. On a certain day, 278 people entered the park. On that same day the admission fees collected totaled $ 792. How many children were admitted on that day? How many adults were admitted?
Aligned template	$u_1^1 + u_2^1 - n_1 = 0$ $\quad$ $n_2 \times u_1^2 + n_3 \times u_2^2 - n_4 = 0$
Instantiated equations	$x + y - 278 = 0$ $\quad$ $1.5x + 4y - 792 = 0$
Answer	$x = 128$ $y = 150$
Derivation 2	
Word problem	A motorist drove 2 hours at one speed and then for 3 hours at another speed. He covered a distance of 252 kilometers. If he had traveled 4 hours at the first speed and 1 hour at the second speed, he would have covered 244 kilometers. Find two speeds?
Aligned template	$n_1 \times u_1^1 + n_2 \times u_2^1 - n_3 = 0$ $\quad$ $n_4 \times u_1^2 + n_5 \times u_2^2 - n_6 = 0$
Instantiated equations	$2x + 3y - 252 = 0$ $\quad$ $4x + 1y - 244 = 0$
Answer	$x = 48$ $y = 52$

Figure 2.2: Two questions that are solved by KAZB.

Another pioneering work in the literature is ALGES (Koncel-Kedziorski et al., 2015). ALGES uses an intermediate representation to model the problem. This IR is called Quantified set (Qset), and it is a tuple with seven elements. These elements are entity, quantity, adjective, location, verb, syntactic information, and container.

In the second step, integer linear programming is used to generate all syntactically correct possible equation trees with the help of Qset. Finally, a joint probability distribution is used for determining how well the equation trees capture the problem. The best tree is chosen to extract the final equation.

This model also captures all types of arithmetic problems like KAZB. However, ALGES has far better accuracy compared to KAZB. Since their equation trees are similar to equation templates, the possible reason for better accuracy is their intermediate representation. Qset is a very detailed way to express the MWP computationally.

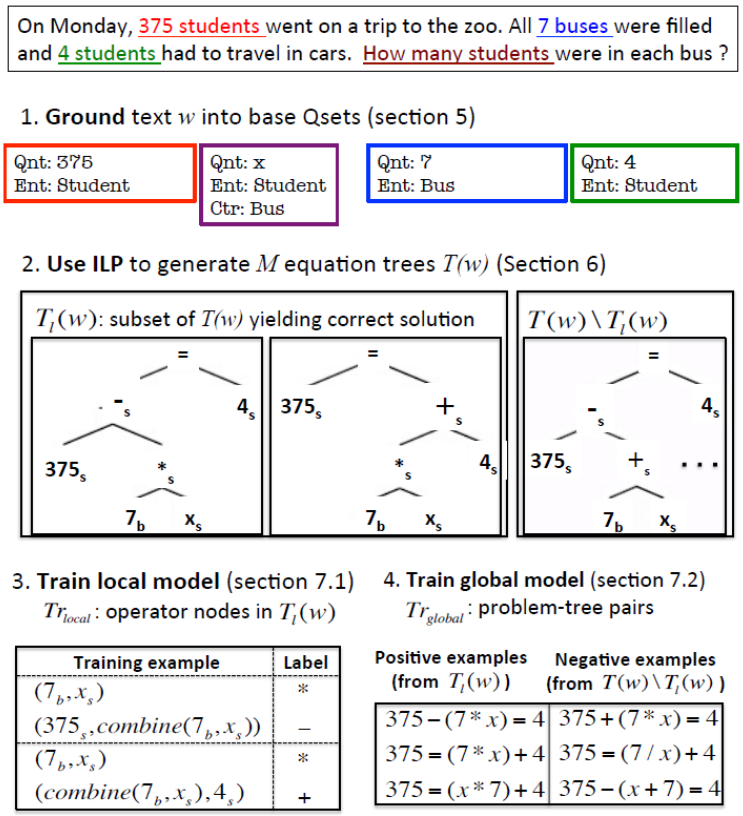

Figure 2.3: Overview of ALGES system.

Another Pioneer work in the MWP literature is developed by Mitra and Baral (2016). In this study, MWP P represented formally as below:

- sequence of k words: $<w_1, \ldots, w_k>$ that constructs the MWP.
- set of variables: $V = \{v_0, v_1, \ldots, v_{n-1}, x\}$ where $v_0, v_1, \ldots, v_{n-1}$ are numbers in P and x is the unknown.
- P_{addsub} is the set of all problems, and each problem can be represented and solved by an equation E constructed by combining the set of variables and binary operators O = {+,-}.

After introducing the representation, they introduce mathematical formulas that they used. These formulas are $C = \{C_{partwhole}, C_{change}, C_{comparison}\}$. In this study, solving MWP is finding the right equation E through applying the right formula C to the set of variables V. Elements of C is explained as below:

$C_{partwhole}$: This formula is applied when there is part whole relationship in the problem. This formula has two slots. The first one is whole, the second one is parts while whole is equal to sum of the parts.

C_{change}: This formula is applied when there is increase or decrease relationship between the numbers in the problem text. This formula has four slots. The first slot is a *start* slot that denotes the original value of a variable. Second one is *end* slot which, denotes the final value of the variable. The last two slots are *gains* and *losses* that

keep the set of increments and decrements on the original value. Given these slots, the equation is generated as follows:

$$val_{start} + \sum_{g \in gains} val_g = \sum_{l \in losses} val_l + val_{end}$$

$C_{comparison}$: When two quantities are compared, these questions are accepted as comparison problems. Comparison formula is applied to these kinds of problems. This formula has three slots. These are *large quantity*, *small quantity* and their *difference*. Given these slots, problem is mapped to *large quantity* = *small quantity* + *difference* equation.

They trained a machine learning method to choose the formula from the set of C. All the features they used in this model were using linguistic attributes like POS tags and dependency labels. Since they were using k-fold cross validation to train their model, their training method was vulnerable to overfitting problem.

Final MWP solver that we will examine here is developed by Shi et al. (2015). Differently from other MWP solvers, this one can solve only number word problems for arithmetic. An example number word problem can be seen below:

(2.3) "7 plus 5 is 6 more than a number. What is the number?"

A number word problem is an arithmetic problem that is explained by a text. However, this text does not have a story in it. There are only mathematical terms in these texts. They are mathematical texts that we explained at the beginning of this chapter. For instance, a problem solver needs only two attributes to solve a number word problem shown above:

1. a list of keywords for arithmetic operations.
2. a method to determine the word for unknown element.

There are two reasons to mention this study here. The first one is that their different approach for solving word problems automatically. A domain-specific language is developed only for this study. Since MWPs are natural language texts, designing a language to solve them is a smart idea. The second reason is the system's success. This solver can solve number word problems with %95.4 precision. According to the developers of the system, such a high precision shows that the system understands the problem instead of guessing its answer.

The reason for such a high precision is that, by transforming NL text to DOL trees, the system "understands" the problem (or has structured and accurate information about quantity relations). Therefore it is more likely to generate correct results than statistical methods who simply "guess" according to features. By examining the problems in the dev set that we cannot generate answers, we find that most of them are due to empty parsing results. (Shi et al., 2015)

This model has three modules. Domain-specific language DOL, semantic parser, and reasoning module. To solve any number word problem, the input problem is parsed by the semantic parser to express a DOL representation tree. Then, the reasoning

module outputs the equation by using DOL. This part can be thought of as a code generation step of a compiler.

```
vf.be.equ($1,$2) → {$1} be equal to {$2}
                 | {$1} equal {$2}
                 | {$1} be {$2}
vf.give($1,$2,$3) → {$1} give {$2} to {$3}
                  | {$1} give {$3} {$2}
nf.math.sum!1($1,$2) → {$1} plus {$2}
                     | {$2} added to {$1}
nf.math.sum!2($1) → sum of {$1}
                  | addition of {$1}
nf.math.power($1,$2)
    → {$1} raised to the {power|exponent} of {$2}
nf.list($1,$2) → {$2} {$1}
mf.number.even → even
mf.condition.if($1) → if {$1}
mf.approximately → approximately
                 | roughly
education.university → university
math.number → number
math.integer → integer
```

Figure 2.4: Example grammar for connecting DOL and Natural Language.

Despite high precision, recall for this system is %60.2, which is low for number word problems. Also, number word problems are a tiny and easy subset of MWPs, as we explained at the beginning of chapter 2.

In addition to the studies that were explained above, there are other MWP solvers in the literature. For instance, Roy and Roth (2016) has developed a system that uses expression trees. Recently, deep learning methods are used too. (Ling et al., 2017) and (Y. Wang et al., 2017) are some example studies that train deep neural networks for solving MWPs. Also, (L. Wang et al., 2018) is an example of deep reinforcement learning for the same purpose.

2.4 Common Features in Mathematical Word Problem Solvers

Even though there are several methods to design an MWP solver, some common attributes appear in most of them. Below, we will explain these attributes:

1. The first phase is linguistic analysis. Almost all MWP solvers process input problem linguistically before working on it.
2. As a second phase, MWP solvers used formal representation to describe MWPs. These representations use linguistic analysis results in general. They make problems easier to be understood computationally.
3. In last phase, MWP solvers use solution generator to express the answer from formal representations.

These stages can be seen in most MWP solvers that we have summarized above. For instance, in STUDENT, finding keywords is linguistic analysis part; converting the input problem to kernel sentences is the representation part, and generating answers from kernel sentences is the solution generator part.

In ARIS, applying dependency parsing, named entity recognition, and coreference resolution to the problem text is linguistic analysis part; constructing states using linguistic information is the representation part and using state progression to generate equations is the last part.

2.5 Dependency Grammar

Dependency Grammar is a grammar formalism that is based on dependency relations. Modern approaches to dependency grammar are based on Tesnière (1959) in general. In this grammar, parsing is done by expressing the relations between the words that make up the sentence. A visual example in Figure 2.5 which is taken from Jurafsky and Martin (2014) might be helpful to understand this formalism:

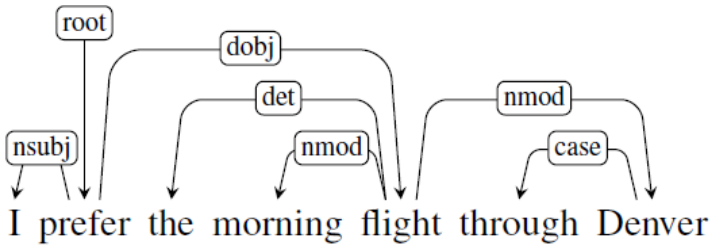

Figure 2.5: An example dependency parsing for a sentence.

The relations in this formalism do not only show syntactic connections but also semantic connections in the sentence. Which words are related to which words can be seen in the dependency grammar trees as opposed to constituent-based grammar trees. Example trees taken from Jurafsky and Martin (2014) are shown in Figure 2.6:

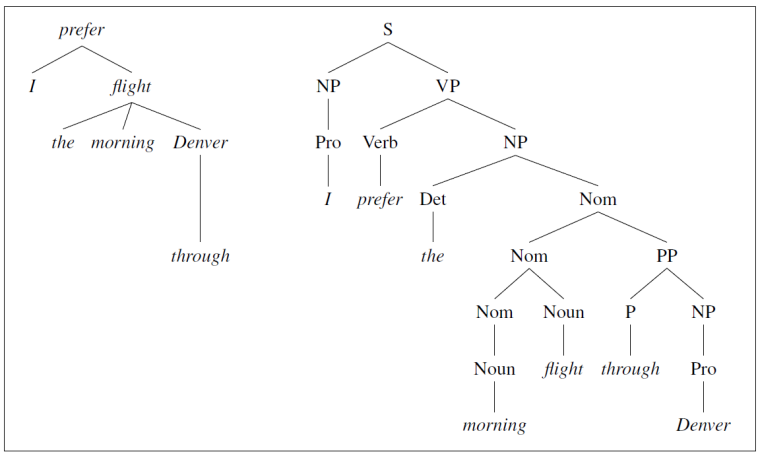

Figure 2.6: Dependency Grammar and CFG parse trees for the same sentence.

In this thesis, we will use a transition-based approach for dependency parsing. Transition-based dependency parsing is a heuristic search algorithm that uses statistical scoring methods to produce dependency trees (Nivre, 2003). Two popular systems in the literature are arc-standard system (Nivre, 2004) and arc-eager system (Nivre, 2003).

Arc-standard system consists of a stack (s), a buffer (b), and a set of dependency arcs (A). Initially, there is a sentence $w_1 \ldots\ldots w_n$, a stack s = [ROOT], a buffer b = $[w_1 \ldots\ldots w_n]$ and, an empty set A = [empty set] in the system.

This process terminates when buffer is empty and stack has only ROOT node in it. After termination of the process, parse tree can be expressed from the set A. In arc-standard system, there are three types of transition:

- LEFT-ARC appends an arc from the word at the top of the stack to the second word of the stack. It deletes the second word from the stack.
- RIGHT-ARC appends an arc from the second word of stack to the top of buffer stack. It deletes the word at the top of the stack.
- SHIFT deletes the element from the front of the buffer and appends it to the stack.

In Dependency Parsing, an oracle is used to determine the correct transition given s, b, and A. Then, this transition is applied to the current state. This process continues until the terminal state shows up. Trace for a transition-based approach using the arc-standard system taken from Jurafsky and Martin (2014) is shown in Figure 2.7 below:

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Figure 2.7: Trace for a sample sentence that uses transition-based approach with arc-standard system.

Oracle is a classifier algorithm that determines the type of transition in the parsing algorithm. Oracle uses the current stack and set of dependency relations. It also uses a reference parse that includes a set of vertices and a set of dependency relations. Given these, oracle decides transitions as follows:

- LEFT-ARC, if it generates a correct head-dependent relation through reference parse and current configuration.
- RIGHT-ARC, if it generates a correct head-dependent relation through reference parse and all dependents have been mapped for the word at the top of the stack.

- SHIFT, for all other cases.

Although the arc-standard system is effective, the arc-eager system (Nivre, 2003) and tree-constrained arc-eager system (Nivre & Fernández-González, 2014) allow a more robust dependency parsing algorithm. Arc-eager system can detect rightward relationships much earlier than arc-standard system. It is possible to do this by changing LEFT-ARC and RIGHT-ARC operations and adding a new operation called REDUCE. Redesigned transitions for arc-eager system are shown below:

- LEFT-ARC appends an arc from the first word of buffer to the top of stack and pop the stack if top of the stack has no head.
- RIGHT-ARC appends an arc from the top of stack to the first word of buffer and moves the first word of the buffer to the stack.
- SHIFT deletes the element from the front of the buffer and appends it to the stack.
- REDUCE pop the stack.

However, the arc-eager system does not guarantee that only the projective dependency tree will be produced during the parsing phase. Since dependency trees are generated from phrase-structure treebanks, they should be projective because phrase-structure treebanks use context-free grammars. Thus, any sentences with non-projective arc increase the error of a dependency parser. This is a bottleneck for a successful dependency parsing.

An arc is projective if there is a path from the head to all words between the head and the dependent. If all arcs are projective in a dependency tree, then this tree is being projective too. To overcome this problem, a tree constrained arc-eager system was developed by Nivre and Fernández-González (2014). In this system, a new transition operation was added to the arc-eager system. This transition operation is called UNSHIFT:

- UNSHIFT moves the top of the stack back to the buffer. This transition is allowed if the top of the stack has no head and the buffer is empty.

In this thesis, we used the non-monotonic dependency parsing model developed by Honnibal et al. (2013) that works with a tree-constrained arc-eager system (Nivre & Fernández-González, 2014). The code for this parsing model is taken from the spaCy library (Honnibal & Johnson, 2015) for efficiency.

2.6 Part of Speech Tagging

In linguistics, part of speech (POS) is a term for classifying words. Words showing the same grammatical features are evaluated under the same POS class. Noun, verb, adjective, pronoun, and preposition are some examples of commonly used POS.

POS tagging is the process that matches all the words in the input text with a part of speech. In this thesis, we will use Penn Treebank tag set for POS Tagging introduced by Marcus et al. (1993).

Table 2.3: Penn Treebank tag set

The Penn Treebank POS tagset.			
1. CC	Coordinating Conjunction	25. TO	to
2. CD	Cardinal Number	26. UH	Interjection
3. DT	Determiner	27. VB	Verb, base form
4. EX	Existential there	28. VBD	Verb, past sense
5. FW	Foreign Word	29. VBG	Verb, gerund/present participle
6. IN	Preposition/subordinating conjunction	30. VBN	Verb, past participle
7. JJ	Adjective	31. VBP	Verb, non-3rd ps. sing. present
8. JJR	Adjective, comparative	32. VBZ	wh-determiner
9. JJS	Adjective, superlative	33. WDT	wh-pronoun
10. LS	List item marker	34. WP	Possessive wh-pronoun
11. MD	Modal	35. WP$	wh-adverb
12. NN	Noun, singular or mass	36. WRB	wh-pronoun
13. NNS	Noun, plural	37. #	Pound sign
14. NNP	Proper noun, singular	38. $	Dollar sign
15. NNPS	Proper noun, plural	39. .	Sentence-final punctuation
16. PDT	Predeterminer	40. ,	Comma
17. POS	Possessive ending	41. :	Colon, semi-colon
18. PRP	Personal pronoun	42. (	Left bracket character
19. PP$	Possessive pronoun	43.)	Right bracket character
20. RB	Adverb	44. “	Straight double quote
21. RBR	Adverb, comparative	45. ‘	Left open single quote
22. RBS	Adverb, superlative	46. ‘’	Left open double quote
23. RP	Particle	47. ´	Right close single quote
24. SYM	Symbol (mathematical or scientific)	48. ´´	Right close double quote

In the next chapters, we will introduce two different POS tagging models. The first one is the Hidden Markov Model (Thede & Harper, 1999) while the other one is the Maximum Entropy Model (Ratnaparkhi, 1996). In this thesis, we preferred the algorithm that uses MEMM to match POS tags with words. This algorithm can match POS tags with words at the human level, with approximately %97 tag accuracy.

2.6.1 Hidden Markov Model for Part of Speech Tagging

HMM takes a linguistic input (word, lemma, sentence, etc.) and assigns POS to these inputs in accordance with a probability distribution for possible assignments. HMM uses Markov chain to do these assignments. Markov chain is an automata-based formal model. In this model, all states are dependent on some other states with a probability and the sum of those probabilities is equal to one. As a result, the

probability of any sequence that is generated from these states can be calculated.

However, this is not enough for assigning POS tags to sequence of linguistic inputs. Because POS tags in a sequence cannot be observable directly since they are hidden. Thus, we need to add some additional attributes to Markov chain. Therefore, sequence of observations and observation likelihood is appended to the model. Observation likelihood is also called emission probability and it is expressing the probability of observation given an observable state.

Also, there are two assumptions in this model. First one is Markov assumption, which claims the probability of any state in a sequence depends on one unit before the current one. More formally,

$P(s_i|s_1, s_2 \ldots .s_{i-1}) = P(s_i|s_{i-1})$

Second assumption is about emission probability. In HMM, an emission probability o_i is only dependent on the state s_i. Formally,

$P(o_i|s_1, s_2 \ldots .s_i, o_1, o_2 \ldots .o_{i-1}) = P(o_i|s_i)$

Finding the most probable sequence of states $(s_1, s_2 \ldots .s_i)$ given sequence of observations $(o_1, o_2 \ldots .o_i)$ is called decoding and it is handled by a dynamic programming approach called Viterbi algorithm (Viterbi, 1967).

This algorithm takes emission probabilities as input and gives the most probable sequence of states as output.

2.6.2 Maximum Entropy Markov Model for Part of Speech Tagging

MEMM is a logistic regression model for assigning POS tags to the linguistic inputs. Given the sequence of linguistic inputs L = l_1^n and the sequence of tags T = t_1^n, P(T|L) is computed for different tag sequences and the best T is chosen as the output sequence of tags in result.

Best Tag = $argmax\ P(T|L)$

Because of MEMM is a logistic regression model, many features might be used to improve the model accuracy. String features like neighboring words, previous tags, prefixes, suffixes; and boolean features like "given linguistic input (l_i) contain a number", "l_i contains an uppercase letter" or "l_i is all uppercase letter" and different combinations of them might be used in the training stage.

For decoding in MEMM, we will use a model that implements a greedy decoding algorithm. In the algorithm, we take l_i, its neighbors within m words l_{i-m}^{i+m} and previous k tags t_{i-k}^{i-1} as inputs and finding the best tag sequence T as an output. Algorithm implementation taken from Jurafsky and Martin (2014) is shown in below:

function GREEDY SEQUENCE DECODING (WORDS L, MODEL P)
for $i \leftarrow 1$ **to** *length(L)*
    $t_i \leftarrow \underset{t' \in T}{\text{argmax}}\ P(t' \mid l_{i-m}^{i+m}, t_{i-k}^{i-1})$
return tag sequence T

Figure 2.8: Greedy decoding algorithm for MEMM

CHAPTER 3

DATA

In this thesis, we have used two datasets. The first dataset is a development dataset that we have developed our MWP solver. The second dataset is a testing dataset to evaluate our solver. In this chapter, we will present the details of these datasets.

3.1 Development Dataset

As we explained in Chapter 2, new attempts for MWP solvers are based on machine learning and deep learning methods. In general, these studies use the same dataset for training and testing their systems.

The most favored approach for training is k fold cross-validation (Mosteller & Tukey, 1968) in these studies. In k fold cross-validation, MWP dataset is divided into k sub datasets. One dataset is used to test the model and the other k-1 datasets are used to train the model. This process is repeated k times with k different test datasets. The average of the results is reported as the result of the study.

In this thesis, we do not develop a machine learning model. We develop a rule-based model by hand-engineering. Thus, we decided to use two discrete datasets for developing and testing our model.

Since there are plenty of datasets in the MWP literature, we do not prepare development dataset from scratch. We merged different datasets and cleaned them to be sure that they are discrete from our test dataset significantly. Datasets that we merged and cleaned to create our development dataset is shown in Table 3.1:

Table 3.1: List of datasets to compile the development dataset.

Dataset Name	**Number of Questions**	**Average Number of Sentences in a Question**
(Roy et al., 2015)	562	2.79
(Kushman et al., 2014)	514	3.20
(Roy & Roth, 2016)	1492	2.77
(Koncel-Kedziorski et al., 2015)	508	2.69

In these datasets, there are 3076 MWPs in total. However, some of these MWPs are multiplication or division problems which are out the scope of this study. Thus, we cut them off from the merged dataset. Also, we want to make development dataset and test dataset completely discrete. It means that there should not be the same or similar questions in these two datasets. We used the edit distance algorithm (Levenshtein, 1966) to detect similar questions in the merged data and the test data. The algorithm is shown in Figure 3.1:

```
function CREATE RAW DEVELOPMENT DATASET (datasetslist, test_data, threshold)
raw_development dataset ← empty list
for each dataset from datasetslist:
    for each MWP from dataset:
        push MWP to raw development dataset
        for each test MWP from test_data:
            if edit_distance(test MWP,MWP) < threshold:
                pop MWP from raw development dataset
                    break
return raw development dataset
```

Figure 3.1: Algorithm for creating raw development dataset from source datasets.

After the cleaning, we obtained a dataset with 899 MWPs. We call this dataset as raw development dataset and we will use this to map verbs to their categories. However, this dataset is still complicated for developing a rule-based problem solver. there are some MWPs that we never achieve to solve in this dataset because they ask for commonsense knowledge. A sample question which asks for commonsense knowledge is shown in Table 3.2 below.

Table 3.2: Sample question which asks for commonsense knowledge.

"A textbook costs a bookstore 44 dollars , and the store sells it for 55 dollars. Find the amount of profit based on the selling price."

In the example above, our solver has to know that profit is equal to the difference in sell price and cost price. However, adding all commonsense knowledge to the system is not straightforward. Thus, we decided to exclude them while developing our solver. After we cut them off, we got a cleaned development dataset with 537 MWPs. This is the dataset that we have used to develop our model.

Table 3.3: Summarized Statistics for cleaned development dataset.

Name of the Dataset	**Number of Questions**	**Average Number of Sentences in a Question**	**Total Number of Verbs in Dataset**
Cleaned Development Dataset	537	3.01	155

Table 3.4: Sample problem from cleaned development dataset.

"Clarence has 5 oranges. He gets 3 more from Joyce. Later, Clarence buys 9 skittles at the store. How many oranges does Clarence have in all?"

3.2 Test Dataset

In this thesis, we have used the AddSub dataset compiled by Hosseini et al. (2014) as test dataset. Since this was the first wide coverage dataset for addition and subtraction problems, this is the most used dataset in the literature. This dataset is used in various previous studies. Therefore, using the AddSub as test data allowed us to compare our model with the other ones.

AddSub is a merged dataset of three different sub-datasets. The first part of the dataset includes 134 MWPs. These problems are easier than other MWPs and there is no information gap or irrelevant information in these questions. In the second part, there are 140 MWPs. These MWPs have more irrelevant information than others.

The third sub-dataset has 121 MWPs. These problems have more information gaps compared to the other sub-datasets.

Table 3.5: Summarized Statistics for sub-datasets of Test Dataset.

Name of Sub Dataset	**Number of MWPs**	**Average Number of Sentences in a Question***	**Total Number of Verbs in Dataset**
Simple Questions Dataset	134	2.88	50
Irrelevant Information Dataset	140	3.09	110
Information Gap Dataset	121	2.74	63
Total	395	-	140

Table 3.6: Sample problems for different sub dataset of the training dataset.

Name of Sub Dataset	**MWP**
Simple Questions Dataset	“Jason had Pokemon cards . He gave 9 to his friends . He now has 4 Pokemon cards . How many Pokemon cards did he have to start with ?”
Information Gap Dataset	“Irwin ’s family went on a camping trip in the mountains . On the first day , they hiked from their car to the campsite . First, they hiked 0.2 mile from the car to a stream , and 0.4 mile from the stream to a meadow . Then they hiked 0.1 mile from the meadow to the campsite . How many miles did Irwin ’s family hike in all ?”
Irrelevant Information Dataset	“There are 31 short trees and 32 tall trees currently in the park . Park workers will plant short trees today . When the workers are finished there will be 95 short trees in the park . How many short trees did the worker plant today ?”

3.3 Contents of Datasets

Both development dataset and test dataset have four common subparts. These are indexing numbers of the MWP, equation of the MWP, solution of the MWP and the raw text of the MWP. We will use solution to measure our model’s accuracy in both datasets. The raw text of MWPs will be inputs for our solver. Equations for the MWPs will be used to map verb categories to the verbs in development dataset.

Table 3.7: A sample structure for an MWP in our dataset.

{‘iIndex’:	231,
‘lEquations’:	[’X = 0.25 + 0.4166666666666667 + 0.25’],
‘lSolutions’:	[0.9167],
	’During a school play , Jonah staffed the snack bar .
	He served 0.25 pitcher of lemonade during the first intermission ,
‘sQuestion’:	0.4166666666666667 pitcher during the second ,
	and 0.25 pitcher during the third .
	How many pitchers of lemonade did Jonah pour in all ? ’}

CHAPTER 4

IMPLEMENTATION

4.1 Conceptual Framework for Mathematical Word Problem Solver

In this chapter, we will implement our Mathematical Word Problem (MWP) solver. This will be done in four stages:

1. Introducing Intermediate Representation (IR) for problem understanding.
2. Implementing algorithm to generate IR from natural language text.
3. Mapping domain-specific verb categories to the verbs.
4. Implementing algorithm to generate mathematical equations from states that are filled by IR through verb categories.

The general framework is shown in Figure 4.1 with a flow chart. Also, an example question is solved by our system explicitly shown in Figure 4.2.

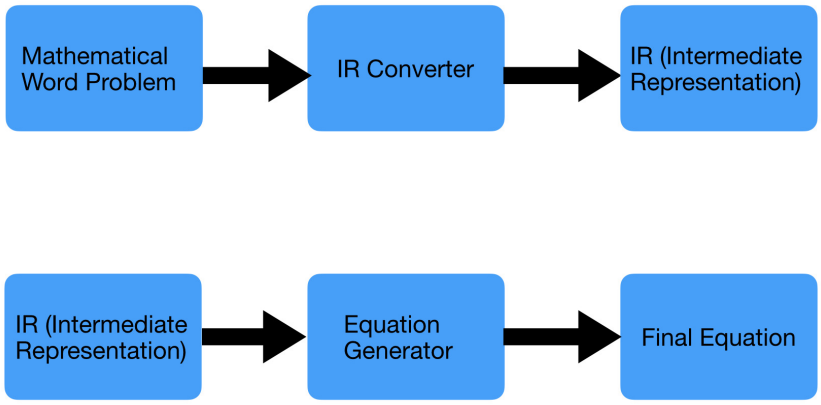

Figure 4.1: The Scheme for MWP Solver.

In Figure 4.2, We showed MWP at the top. This MWP has been divided into three components. We will explain how to extract these components in the next stages. We painted every component with a different color.

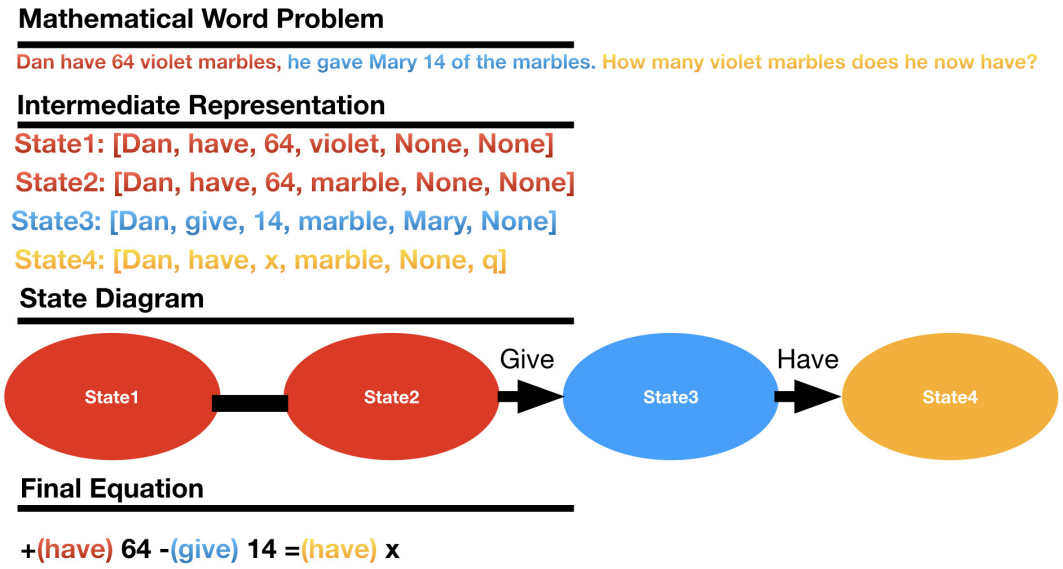

Figure 4.2: An example MWP and solution generation step by step.

After component analysis, we generate IR for every component. 6-tuple lists describe our IRs and we assign them to states to apply semantic actions in the next steps. In the state diagram, these explicit states have shown abstractly. Every verb in states apply semantic action to the previous state. Since state-1 and state-2 come from the same component, we do not have any transition between them. Finally, we have an equation at the bottom of the diagram. After all expressions, we showed the associated verb that appends this expression to the equation.

4.2 Intermediate Representation for Problem Understanding

We constructed a 6-tuple representation system to construct the states. We call this as Intermediate Representation (IR). These IRs construct automata together. Since our automata show MWP as a state progression, we can establish the final equation compositionally.

In the example in Figure 4.2, marble is the common object between components. Thus, we should use objects as unknowns to construct our equation. We keep these objects in IR[3] in our representation.

Also, the number of marbles changes state by state in accordance with the entity. So, we should use numbers and entities to construct an accurate equation. Therefore, entities are placed in IR[0] and numbers are placed in IR[2] in our representation.

We should also add verbs to IR. Verbs determine the sign of numbers in equations. For instance, “lose” maps to negative sign while “add” maps to positive sign, and “have” maps to equality sign under some conditions. Verbs are placed in IR[1] in our representation.

For the some ditransitive verbs like “give” or “take”, we need a second entity too.

One entity for the subject should be extracted, while one entity for the complement is extracted. We keep second entities in IR[4].

Finally, we use some lexical rules to capture the development dataset better. Thus, we append some letter pointers to the last slot of IR (IR[5]) when some specific keywords appear in problems. Our letter pointers and lexical rules are:

"q" pointer: We used this pointer to point the question states. Our problem solver is looking for question words like "what" and "how" and generate a state with lemmas that depend on these question words.

Sample Question:*'Sandy had 26 pet fish. She bought 6 more fish. How many pet fish does Sandy have now?'*

In the sample MWP above, the sentence starts with "How Many" should be treated as a question sentence.

Thus, our system appends a "q" pointer to the states that are generated from this sentence. Expected states are listed below:

- $[' Sandy', ' have', ' x', ' pet', None, ' q']$
- $[' Sandy', ' have', ' x', ' fish', None, ' q']$

"n" pointer: This pointer is used for redundant states. We append "n" pointer to IR[5] while a word has a "PART" part of speech and "neg" dependency label. This state is accepted as redundant state. States with "n" pointers are passed while generating the final equation.

Sample Question:*"Keith returned a CD player for $ 136.01 , spent $ 139.38 on speakers , and $ 112.46 on new tires . He wanted 3.0 CD 's for $ 6.16 , but did n't buy them . In total , how much did he spend"*

As can be seen in the sample MWP above, CD's that costs 6.16 dollars appear in the MWP. However, these CD's are not bought by Keith. Thus, we should pass this redundant information while generating states.

"m" pointer: "m" pointer is used in comparison problems. We append "m" pointer if there is a comparison keyword in question state.

Sample Question:*'Diane is a beekeeper. Last year, she harvested 2479.0 pounds of honey. This year, she bought some new hives and harvested 6085.0 pounds of honey. How many more pounds of honey did Diane harvest this year than last year?'*

In the MWP above, there is a comparison meaning in the question sentence. Thus, our system should append "m" pointer to the states that are generated from question sentence.

"s" pointer: "s" pointer is used in summation problems. We append "s" pointer if there is a summation keyword in question state like "altogether". Also, we use this pointer when a keyword that maps to set union operation appears. "either" is an example for these words.

Sample Question:*'A cake recipe requires 0.6 cup of sugar for the frosting and 0.2*

cup of sugar for the cake. How much sugar is that altogether?'

"c" pointer: Some keywords have one vs. all meaning. When these keywords appear, we append "c" pointer to the IR[5]. "rest" is an example for these words.

Sample Question:*'There were 3409 pieces of candy in a jar. If 145 pieces were red and the rest were blue, how many were blue?'*

"b" pointer: Some keywords reverse the meaning of last state. When this keywords appear, we should apply the exact reverse operation of the last operation to the number in the current state. "but" is an example for these keywords.

Sample Question:*'A car company produced 3884.0 cars in North America but 2871.0 cars in Europe were recalled. How many cars were successfully produced?'*

In the example above, "but" keyword reverse the meaning of the verb "produced" that appears in the first sentence. We expect to see these two states here:

- $[' company', ' produce', ' 3884.0', ' car', None, '']$
- $[' company', ' produce', ' 2871.0', ' car', None, ' b']$

"p" pointer: "p" pointer is adjusting the timeline of the problem. Some keywords give a starting point meaning to their states. These states have to be processed in the equation generation phase before all states. "begin" is an example for these keywords.

Sample Question:*'Michelle began her pizza delivery route with 0.5 tank of gas in her car . When she made it back to the pizzeria , 0.16666666666666666 tank of gas was left. How much gas did Michelle use ?'*

"r" pointer: This pointer is similar to "b" pointer. We should apply exact reverse operation of the last operation here too. However, we apply this to the previous state's number instead of the current one. "off" is an example keyword for "r" pointer.

Sample Question:*'There were 10 students riding on the school bus. At the first stop, 3 students got off of the bus. How many students are left on the bus?'*

To sum up, the formal definition for our automata is shown in Table 4.1. We used a similar formulation with Hopcroft et al. (2001). The algorithm for extracting the components from raw MWP is shown in Figure 4.3:

Table 4.1: Formal definition for our Automata.

Automata: MWP = <*IR, S, F, Q, V, T*>
MWP: MWP is Mathematical Word Problem to be solved automatically. MWP should be divided into its components before extracting *IR*. A component is the smallest informative part of a problem. If we formally showed this, MWP = $c_1c_2 \ldots c_i$; c = component; i = number of components.
Intermediate Representation: *IR* is a representation of MWP. 6-tuple list to keep syntactic and semantic knowledge for problems. *IR* = [Entity, Verb, Number, Object, Entity2, Lexical Attributes]
Set of States: *S* is a finite set of states to describe MWP. Number of elements for this set is determined by the number of components in MWP. Components that are represented by *IR* is called state.
First State: *F* is the first state in *S*. We discriminate this state because it is treated differently from other states in the equation generation stage. *IR* for the first component is labeled as the first state.
Question State: *Q* is the question state in *S*. This state is treated as an accepted state in our system. The final equation is returned after reading this state. *IR* for the component with question word is labeled as question state.
Set of Verbs: *V* is a finite set of (verb, verb category) duplicates for MWPs. There are five alternatives for verb category slot here. These are Observation, Positive, Negative, Positive Transfer, and Negative Transfer verbs. These categories determine the actions in state progression.
Semantic Function: *T* is a 3-tuple for semantic actions. For every verb category defined in *V*, there is a semantic action to contribute to the final equation. These actions are addition, subtraction, and equality.

```
function CREATE MWP COMPONENTS (MWP)
Input: MWP
Output: Components of MWP
verbflag ← False, numberflag ← False, temp ← first_token, index ← 0, components ← empty list
for each token from MWP:
    if token is VERB and verbflag is True:
        components[index] ← MWP[temp_ind:token_ind], index ← index + 1, temp ← MWP[token_ind+1]
    else if token is VERB and verbflag is False: verbflag ← True
    else if token is NUMBER and numfberflag is True:
        components[index] ← MWP[temp_ind:token_ind], index ← index + 1, temp ← MWP[token_ind+1]
    else if token is NUMBER and numberflag is False: numberflag ← True
return components
```

Figure 4.3: Algorithm for dividing MWP to its components.

As shown in Figure 4.3, we create a component when a verb or a numeric value appears. Rationale behind this approach is straightforward. Numbers for the final equation can be found in numeric values that appear in the problem. Operations for the final equation can be found in verbs that appear in the problem.

Thus, we are searching for verbs and numeric values in MWP, when any of them apears, we generate a new component.

4.3 Implementing Algorithm to Generate Intermediate Representation from Natural Language Text

After finding the components, we implement an algorithm to generate IR for any MWP. We have used part of speech and dependency parsing structures in our converter algorithm.

Firstly, we create components through the algorithm explained in Figure 4.3. Then, we use these components as input in our converter algorithm. The algorithm for converting MWP components to IR is shown in Figure 4.4:

```
function GENERATE IR (components of MWP, rulelist ← list of lexical rules for inference)
IR ← [None]*6, statelist ← empty list
for each component from components of MWP:
    copied_IR ← None
    for each token from component:
        if lemma_token in rulelist: map(rulelist[lemma_token], IR[5])

        if lemma_token is PRONOUN and dep_token is nsubj: IR[0] ← entity from previous state
        else if head_token.pos is VERB and dep_token is nsubj and pos_token is not DET:
            IR[0] ← lemma_token
        else if lemma_token+1 is head_token and dep_token is poss and pos_token-1 is not prep:
            IR[0] ← head_token

        if pos_token is VERB and pos_token+1 is NUM: IR[1] ← lemma_token
        else if lemma_token is head_token or head_token.pos is VERB:  IR[1] ← lemma_token
        else if pos_token is NUM and head_token.pos is VERB: IR[1] ← head_token
        else if pos_token+1 is PART and head_token+1 is lemma_token+2 and pos_token+2 is VERB:
            IR[1] ← lemma_token+2

        if pos_token is NUM: IR[2] ← lemma_token
        else if lemma_token is "some": IR[2] ← "x"

        if pos_token is NUM and head_token.pos not in [VERB, PROPN] and head_token-1.pos is ADJ:
            copied_IR ← copy(IR), IR[3] ← head_token-1, copied_IR[3] ← head_token
        else if pos_token is NUM and head_token.pos not in [VERB, PROPN]: IR[3] ← head_token
        else if pos_token is VERB and head_token.pos is NOUN:  IR[3] ← head_token

        if dep_token is dative and head_token.pos is VERB: IR[4] ← lemma_token
        else if head_token is IR[3] and dep_token is poss: IR[4] ← lemma_token
        else if (head_token+1.dep is prep or head_token.head.pos is VERB) and dep_token is poss:
            if lemma_token is not PRONOUN: IR[4] ← head_token
            else: IR[4] ← second entity from previous state

        if component ends and copied_IR is None:  push IR to statelist, keep IR[0], IR[3], IR[4]
        else: push IR to statelist, push copied_IR to statelist , keep IR[0], IR[3], IR[4]

return IR
```

Figure 4.4: Algorithm for generating IR from components.

In this algorithm, we create a 6-tuple list to keep temporary IR for every component in MWP. Algorithm applies some procedures to generate IR slots.

IR[0]: IR[0] is entity slot. We have three conditions here. If the dependency label is a nominal subject, the head of the current token is verb and part of speech is not determiner, we assign lemma for the token to the IR[0]. On the other hand, if the current word is a pronoun, we assign the previous entity instead.

If the first condition is not satisfied, we are looking for a possession modifier for the dependency label when the part of speech for the previous token is not a prepositional modifier and the next token is the head of current token. In that case, we assign the head of the current token to the IR[0]. In all other cases, we are using the last entity that is found in the system. This is a circumscription assumption from McCarthy (1980).

IR[1]: IR[1] is verb slot. We have three procedures here. If part of speech is a verb and the token is not dependent on any other token, if it is a head, or next token's pos is a number, we assign lemma for the token to IR[1].

The second procedure is for detecting to infinitives. In to infinitives, second verbs include the mathematical meaning in MWP context. Thus, we assign second verb to IR[1]. In the third statement, we are checking the part of speech for the current token. If it is number and its head is a verb, we assign the head to IR[1].

IR[2]: IR[2] is number slot. If part of speech is a number, we assign this number to IR[2]. On the other hand, if lemma for the current token is some, we assign "x" to IR[2] as unknown.

IR[3]: IR[3] is object slot. In our algorithm, we are interested in objects that depend on numeric values. Thus, we check the head of tokens, which have number part of speech. If it is not a verb or proper noun, we assign the head of number to IR[3].

Also, if there is an adjective that modifies the object, we should create two states for this component. Hence, we are creating a copy of IR and change IR[3] to the adjective that modifies the object in first IR. Sample MWP from development dataset is below:

(4.1) "There are 95.0 short trees and 32.0 tall trees currently in the park. Park workers will plant 31.0 short trees today. When the workers are finished, how many short trees will be in the park?"

In this problem, we need to separate short and tall trees to find the answer. Without adding our adjective rule, the system would accept all kinds of trees as similar type and we could end up with a wrong answer.

Also, we add nouns to IR[3] when they are head of sentence. If any object has not been found at the end of component, we assign the last object to IR[3] as IR[4] using circumscription assumption.

IR[4]: IR[4] is second entity slot. If head of the word is a verb and the word is a dative, we assign lemma of this word to IR[4]. If head of the current token is equal to object and dependency label for the current word is possession modifier, we assign lemma of this token to IR[4].

On the other hand, if the head of the next token is a prepositional modifier or head of head for the current word is a verb and current token is a possessional modifier, we

assign the head of current word to IR[4] when the current word is not a pronoun. For all other situations, we used the second entity from the previous state as IR[4] using circumscription assumption.

IR[5]: We introduce a lexical rule list for the last slot of IR. These rules are domain-specific ad-hoc rules and they contribute to the algorithm with letter pointers to fill the last slot.

When the component ends, the algorithm appends this IR as a state to statelist. For the next components, all IR slots are assigned to None except to slots that use circumscription assumption. If there is a copied_IR (this is valid if there is an adjective that modifies the object in IR), the algorithm appends this copied_IR to statelist together with IR.

4.4 Mapping Verbs to Their Domain-Specific Verb Categories

As we mentioned earlier, we will generate mathematical equations using an automata-based system. The essential tool we will use in doing this will be verb categories. Since verbs are elements that represent actions in any natural language, it is quite an effective method to determine verbs as the main element providing dynamism in natural language modeling.

Table 4.2: Verb Categories and MWP examples.

Observation Verbs: These verbs include verbs that provide pieces of information about the current situation. They contribute the final equation with either addition operation or equality operation. **Sample Verbs**: be, have, notice. **Example**: Joan found 70 seashells on the beach. She gave Sam some of her seashells. She **has** 27 seashells. How many seashells did she give to Sam?
Positive Verbs: These verbs include verbs that have a positive meaning. They contribute to the final equation with addition operation. **Sample Verbs**: add, pick, collect **Example**: There are 22 walnut trees currently in the park. Park workers will **plant** walnut trees today. When the workers are finished, there will be 55 walnut trees in the park. How many walnut trees did the workers plant today?
Negative Verbs: These verbs include verbs that have a negative meaning. They contribute to the final equation with the subtraction operation. **Sample Verbs**: spend, lose, return **Example**: Joan has 8 orange balloons but **lost** 2 of them. How many orange balloons does Joan have now?
Positive Transfer Verbs: This category is for ditransitive verbs. These verbs contribute the final equation with addition operation if the question sentence entity is in entity position for the given component. If question sentence entity is in second entity position, they contribute as subtraction operation. **Sample Verbs**: give, lend **Example**: Sam had 9 dimes in his bank. His dad **gave** him 7 dimes. How many dimes does Sam have now?
Negative Transfer Verbs: This category is for ditransitive verbs. These verbs contribute the final equation with subtraction operation if the question sentence entity is in entity position for the given component. If question sentence entity is in second entity position, they contribute as addition operation. **Sample Verbs**: take, get **Example**: There are 46 rulers in the drawer. Tim **took** 25 rulers from the drawer. How many rulers are now in the drawer?

In our model, we need to use a domain-specific verb categorization method to match our verbs with mathematical operations. Because we want to reach mathematical expressions from an intermediate representation. In this context, the actions are equivalent to mathematical operations. In this thesis, five of the seven categories used in ARIS will be sufficient for our solver.

Since the number of verbs in observe and transfer categories are narrow, we assign the verbs that belong to these categories by hand. After the assignment of these seventeen verbs in these three categories, we use a straightforward algorithm to place the remaining verbs into the positive or negative verb categories. In the first stage, we produce intermediate representations for the problems in our development dataset using the algorithm given in Figure 4.4.

In the second stage, we match the numerical values in the annotated equation from development dataset and the states with the same numerical value in the IR that we produced in the first stage. After this matching, we match the mathematical operation in the equation with the verbs in given states. If there is addition in the equation, we increase the value of the verbs by one in a hash table. If there is subtraction in the equation, we decrease the value of the verbs by one in the hash table.

After examining all the problems in the development dataset with this method, we evaluate the hash table's values for each verb. If the value is less than zero, we place the verb in the negative category. Otherwise, we place it to the positive category. In this process, we use the raw development dataset, which we describe how we derive in the chapter 3, as it contains more problems and verbs.

However, we found seven verbs unmatched in the test dataset after applying the process that we explained above. For these verbs, we calculated their semantic similarity with matched verbs in the dataset. Word2vec model (Mikolov et al., 2013) is used for this calculation. As a result of this calculation, we assign these verbs to the closest verb category. The whole algorithm is shown in Figure 4.5:

```
function MAP VERB CATEGORIES (raw development dataset, verb categories, verb list for test data)
assign observation, positive transfer, negative transfer verbs to their verb categories by hand
hash table ← empty hash table
for each MWP from raw development dataset:
   IR ← GENERATE_IR(MWP), equation ← MWP[equation]
   for each number from equation:
      hash table[find state for number and extract verb from this state] += find operation for number
for each verb from hash table:
   if hash table[verb] < 0: push verb to negative verb category
   else: push verb to positive verb category
for each verb from verb list for test data:
   if verb not in hash table: push verb to closest verb category found by word2vec similarity
return verb categories
```

Figure 4.5: Algorithm for assigning verbs to their verb categories.

4.5 Generating Mathematical Equations from Intermediate Representation

At this stage, we already have a set of states which are filled with IR and set of verb categories that are filled by verbs. Now we can generate equations through using verbs as transition function. Our algorithm to extract the final equation from states is shown in Figure 4.6:

```
function GENERATE EQUATION (states)
stack_hash ← empty stack, constant_entity ← question_state[entity]
for each state from states:
    if state[obj] not in stack_hash: stack_hash[state[obj]] ← None, temporary_stack ← empty stack
    else: temporary_stack ← stack_hash[state[obj]]
    addition_temp ← None
    if state is first_state:
        if state[verb] is (observationverb or positiveverb): addition_temp ← True
        else if state[verb] is negativeverb: addition_temp ← False
        else if state[verb] is positivetransferverb and constant_entity is state[ent]: addition_temp ← False
        else if state[verb] is positivetransferverb and constant_entity is state[ent2]: addition_temp ← True
        else if state[verb] is negativetransferverb and constant_entity is state[ent]: addition_temp ← True
        else if state[verb] is negativetransferverb and constant_entity is state[ent2]: addition_temp ← False
        if (addition_temp is True and "r" not in IR[5]) or (addition_temp is False and "r" in IR[5]):
            push "+state[num]" to temporary_stack
        else if (addition_temp is True and "r" not in IR[5]) or (addition_temp is False and "r" in IR[5]):
            push "-state[num]" to temporary_stack

    else if state is not question_state and state is not first_state:
        if state[verb] is positiveverb: addition_temp ←True
        else if ALL verbs from first_state to current_state are observationverb: addition_temp ←True
        else if state[verb] is observationverb and SOME previous verbs are not observationverb:
            push "=state[num]" to temporary_stack, addition_temp ← None
        else if state[verb] is negativeverb: addition_temp ← False
        else if state[verb] is positivetransferverb and constant_entity is state[ent]: addition_temp ← False
        else if state[verb] is positivetransferverb and constant_entity is state[ent2]: addition_temp ← True
        else if state[verb] is negativetransferverb and constant_entity is state[ent]: addition_temp ← True
        else if state[verb] is negativetransferverb and constant_entity is state[ent2]: addition_temp ← False
        if (addition_temp is True and "r" not in IR[5]) or (addition_temp is False and "r" in IR[5]):
            push "+state[num]" to temporary_stack
        else if (addition_temp is True and "r" not in IR[5]) or (addition_temp is False and "r" in IR[5]):
            push "-state[num]" to temporary_stack

    else if state is question_state:
        if "m" in IR[5]: return(max(stack_hash[state[obj]]) - min(stack_hash[state[obj]]) = x)
        else if "s" in IR[5]: return(+numeric for each numeric from stack_hash[state[obj]] = x)
        else if "=" not in stack_hash[state[obj]]: return(stack_hash[state[obj]] = x)
        else if state[verb] is observationverb or positiveverb: return(+x stack_hash[state[obj]])
        else if state[verb] is negativeverb: return(-x stack_hash[state[obj]])
    stack_hash[state[obj]] ← temporary_stack
```

Figure 4.6: Algorithm for extracting equation from States.

First of all, we define required data structures. "stack_hash" is a hash table that keeps state objects as key, numbers and expressions as value. We also have a "constant_entity". This entity equal to the entity of question state. This is important for tracing the MWP.

After introducing data structures, the algorithm traces every state starting from the

first state. Here we have three different situations for the current state. It might be first state, question state, or any other state.

In the first state situation, we have six sub situations. In the first one, verb is in positive or observation category. Either way, we add the number of the state with a positive sign to the stack. In the second situation, verb is in negative category. We add number of the state with a negative sign to the stack.

The third and fourth situation is for positive transfer verb categories. If verb is in positive transfer category and constant_entity is equal to the entity of the current state, we add number of the state with a negative sign to the stack. Otherwise, we add number of the state with a positive sign to the stack. Last two situation is for negative verb categories. If verb is in negative transfer category and constant_entity is equal to the entity of the current state, we add number of the state with a positive sign to the stack. Otherwise, we add number of the state with a negative sign to the stack.

In addition to these procedures, we also check the IR[5] for an “r” pointer. “r” pointer was appended to the IR[5] while “out” or “off” keyword is found in the current component. In that case, we are reversing the sign that we added lastly.

In the second state situation, the current state is neither first state nor question state. Here we have seven sub situations. First of all, if verb is in positive verb category or observation verb category and for all previous states, their verbs are in observation verb category; we add numeric value of the state with a positive sign to the stack.

Secondly, if verb is in observation verb category and for at least one previous state, its verb is not in observation verb category; we add number of the state with an equality sign (=) to the stack. This means that some positive or negative actions are applied before this observation and we assume that this first observation verb constructs the equation.

In third, verb is in negative category. We add number of the state with a negative sign to the stack. For the last four sub situations, algorithm is using the same rules with the last four sub situation in the first state situation. Lastly, we check the IR[5] for an “r” pointer and following the same procedure that we explained in the first state situation.

Finally, in question state situation, we have five sub situations. As a first step, we are checking for "m" pointer in the last slot. This is a minus pointer that comes from the lexical rule list. If question sentence is a comparison sentence, that means subtraction in MWP context. Thus, we add the maximum number from the values to our final equation with a positive sign for the given object in our hash table. Because we need subtraction operation, we are adding the minimum number of the values to our final equation with a negative sign for the given object in our hash table. After assigning an unknown x with equality sign, we are returning the result equation.

In second sub situation, we are checking for s pointer in the last slot. This is a summation pointer that comes from the lexical rule list. If there are some keywords in question sentence, that means summation in MWP context. Thus, we are adding all the numbers to our final equation with positive sign for given object in our hash table. After assigning an unknown x with equality sign, we are returning the result equation.

Thirdly, if equality sign is not found for given object, we are pushing an unknown x with equality sign to the hash table value and return this as a final equation. In the fourth and fifth sub situations, we know that equality sign is found for given object. Thus, we are checking the verb category in both situations. If verb category is observation verb or positive verb, we are pushing an unknown x with positive sign to the front of hash table value and return this as a final equation. If verb category is negative verb, we are pushing an unknown x with negative sign to the front of hash table value and return this as a final equation. After every state evaluation, we append the result of the evaluation to the given object's hash value.

CHAPTER 5

RESULTS

In this chapter, we describe our evaluation results on the test data. The output of our solver is the answer for a given MWP. Thus, we are going to use only one criterion here. This is the accuracy rate of the model in terms of correct answers. After the evaluation, we will make an error analysis at the end of this chapter.

5.1 Evaluation

As we explained in chapter 3, our test data is the AddSub dataset that includes 395 MWPs compiled by three sub-datasets. These sub-datasets are simple questions dataset with 134 straightforward MWPs, irrelevant information dataset with 140 MWPs that includes redundant information and information gap dataset with 121 MWPs that lack information.

We compared our model with eight previous models from the literature. All of them were tested in the AddSub dataset that we explained above. We also add a baseline model that uses an edit distance algorithm for answer generation.

In this baseline model, we find the most similar MWP for every test MWP in the development data. After finding the closest problem, we change the numbers in the equation with the numbers from test data problems. Then, we solve the generated equation to find an answer. Results for the evaluations are shown in Table 5.1, the algorithm for the baseline model is shown in Figure 5.1:

```
function BASELINE MODEL (raw development dataset, test dataset, answer ← empty hash table
for each MWP test from test dataset:
    for each MWP dev from raw development dataset:
        if edit_distance(MWP_test, MWP_dev) is minimum:
            answer[MWP_test] ← map(numbers(MWP_test),equation(MWP_dev))
return answer
```

Figure 5.1: Algorithm for the baseline model.

Table 5.1: Evaluation results on AddSub dataset for different models.

Model	**Accuracy**
(Hosseini et al., 2014)	77.70
(Kushman et al., 2014)	64.00
(Koncel-Kedziorski et al., 2015)	77.00
(Roy & Roth, 2016)	78.00
(Zhou et al., 2015)	53.14
(Mitra & Baral, 2016)	86.07
(Roy & Roth, 2017)	60.99
(L. Wang et al., 2018)	78.50
Baseline Model	18.73
Our Model	**76.46**

As a result, our model is somewhere in the middle compared to the previous studies in the literature. On the other hand, when we evaluate our model's accuracy in sub-datasets, results change a bit. Evaluation results for sub-datasets that are reported so far are shown in Table 5.2 below:

Table 5.2: Evaluation results for sub-datasets.

Model	**Accuracy of Simple Questions**	**Accuracy of Information Gap Questions**	**Accuracy of Irrelevant Information Questions**	**Average Accuracy**
(Hosseini et al., 2014)	83.6	75.0	74.4	77.7
(Kushman et al., 2014)	89.6	51.1	51.2	64.0
(Mitra & Baral, 2016)	**96.27**	82.14	**79.33**	**86.08**
Our Model	82.84	**88.43**	60.0	76.46

According to sub-datasets evaluation results, our model has the best accuracy in the information gap sub-dataset. 107 MWPs out of 121 are solved correctly by our system. This is %88.43 of the sub-dataset and beats the Mitra and Baral (2016) by a margin of %6.29.

Although our model gives average results in other datasets, we want to note that we are using completely different datasets to develop and test our models. We explained how to do this in chapter 3. Since we are making development and test datasets significantly different from each other, an edit distance algorithm could be failed while finding similar questions. Therefore, our baseline model has extremely low accuracy.

However, other studies that we listed in Table 5.1 does not have the same sensitivity while prepare training and test datasets. They are using k-fold cross-validation for

evaluation. Thus, all questions in the AddSub were used in the training phase for k-1 times and the testing phase for one time.

In this case, Overfitting might arises because of the high similarity between the elements of datasets. In MWP solvers, some parts of the linguistic data are not used at all. Thus, high similarity can be accepted as data overlapping in MWP solvers.

In the AddSub, we detected 70 problem duplicates with very high similarity. 86 of the 395 MWPs are appears in these duplicates at least one time. An example of these duplicates are shown in 5.1 and 5.2:

(5.1) "There are 4 walnut trees currently in the park . Park workers will plant 6 walnut trees today . How many walnut trees will the park have when the workers are finished ?"

(5.2) "There are 33 walnut trees currently in the park . Park workers will plant 44 walnut trees today . How many walnut trees will the park have when the workers are finished ?"

As can be seen in the examples above, verbs, entities, objects, and mathematical operations for solutions are similar in both questions. If one of these problems is in the training dataset and the other one is in the test dataset, the accuracy of the tested model will appear higher than it should be.

For a better understanding of this claim we replicated Kushman et al. (2014) as an example. First, we pruned multiplication and division problems from their dataset. Secondly, we reproduced their model using this raw dataset. Finally, we pruned similar questions from this dataset and reproduced their model using this cleaned dataset. Comparative results are shown below:

Table 5.3: Comparative results for replicated KAZB.

Dataset	**Number of Questions in Dataset**	**Number of Similar Questions in Dataset**	**Accuracy of the Model**
Cleaned Dataset	43	0	77.78
Raw Dataset	52	9	90.91

As can be seen in Table 5.3, the accuracy of the model decreased drastically after cleaning the dataset in accordance with the question similarity.

These results support our claims about the models that have similar questions in the dataset have an overfitting problem. These results also show that statistical models reach answers through linguistic similarity instead of understanding the semantics of questions. On the other hand, our IR does not depend on linguistic similarity. Thus, our model's accuracy is close to the statistical approaches, even though we use separated datasets in the developing and testing phases.

Given this information, we can say that our model is more powerful than the other ones since their models are tended to be overfitting because of dataset choices.

5.2 Error Analysis

Our problem solver answered 302 of 395 MWPs in the test dataset correctly. In this section, we will analyze 93 MWPs that are answered incorrectly. The summarization table for the wrong answers can be seen in Table 5.4 below. In total, we found five error types among these 93 MWPs and three of them exist in all sub datasets.

Table 5.4: Summarization Table for Incorrect Answers.

Name of Sub Dataset	**Wrong Verb Category**	**Redundant Number Assignment**	**Ill-Formed IR**	**Commonsense Knowledge**	**Ill-Formed MWP**
Simple Questions	15	-	6	2	-
Information Gap	5	-	5	4	-
Irrelevant Information	4	27	21	3	1
Total	24	27	32	9	1

First error type is the wrong verb category. Our system generates the final equation using verb categories. Thus, a wrong assignment might cause wrong equations and a wrong equation drives us to the wrong answer. However, this error is the easiest one for recovering. In these questions, intermediate representation is produced correctly.

- "Jessica spent $ 10.22 on a cat toy , and a cage cost her $ 11.73 . What was the total cost of Jessica 's purchases ? "

In the question above, "spent" is used with an addition meaning. However, it is in negative verb category in the verb lexicon. Therefore, we get a wrong answer for this question.

This error type occurs in 24 questions in the test dataset. If we used gold verb categories to solve these questions, the system's total accuracy would increase to %82.53 and accuracy for simple question sub dataset would increase to %94 from %82.84.

The second error category is the redundant number assignment. This error appears with irrelevant information. When there is irrelevant information in MWPs, our system may not discriminate this information from the rest of the question. This happens when the objects are common in the MWP. Because we discriminate irrelevant information with the objects, appearance of the same object in irrelevant information breaks our system.

- 'Tom found 15 seashells and Fred found 43 seashells on the beach . When they cleaned them , they discovered that 29 were cracked . How many seashells did they find together ? '

As can be seen above, all numbers are about seashells in the MWP. However, the question sentence asks for discrimination in accordance with the verb. Thus, our system fails in these kinds of questions.

Thirdly, there are some MWPs with ill-formed intermediate representation. This is the most fundamental error type in our system because this error shows that the system does not understand the problem computationally.

- 'Ella owns 2 dogs . Each day , 1 dog eats 0.125 scoop of dog food and the other dog eats 0.125 scoop . Together , how much dog food do the 2 dogs eat each day ? '

In the MWP above, our system generates a completely meaningless IR as we have shown below:

'state1': ['Ella', 'own', '2', 'dog', None, '']
'state2': ['dog', 'eat', '1', 'dog', None, '']
'state3': ['dog', 'eat', '0.125', 'scoop', None, '']
'state4': ['dog', 'eat', '0.125', 'dog', None, '']
'state5': ['Ella', 'eat', '0.125', 'scoop', None, '']
'state6': ['Ella', 'eat', 'x', 'scoop', None, '']
'state7': ['dog', 'do', 'x', 'food', None, 'q']

Another error category is commonsense knowledge. Commonsense knowledge is facts about daily life. Our system does not cover all of this knowledge. Thus, some MWPs could not be solved by our system.

- 'Joan bought toy cars for $ 14.88 , a skateboard for $ 4.88 , and got toy trucks for $ 5.86 . She spent $ 14.55 on pants . In total , how much did Joan spend on toys ?'

In the question above, we are asked to find the total spend on toys. Our system should know which objects are toys to solve this question. However, this information is not reachable from our system. Therefore, true answer could not be produced by the problem solver.

The last kind of error is ill-formed MWP. There is only one example for this MWP in the test dataset. These kinds of MWPs are not clear enough to be processed by our model successfully.

- 'Pamela bought 9.8 ounces of sugar , and she spilled 5.2 ounces of it on the floor . How much is ? '

In the example question above, the question sentence is not complete. Thus, our system can not decide what to do in the equation generation phase and the system generates a wrong answer.

CHAPTER 6

DISCUSSION

We expected the MWP solver developed in this thesis to contribute to the literature at three points:

1. Contributing to the automatic MWP solver literature in the cognitive science axis. In this context, introducing an intermediate representation to model the problem solving strategies used by children.
2. Achieving a close problem coverage of ARIS using five of seven categories they introduced.
3. Demonstrating that a rule-based model can achieve the accuracy close to statistical models in a dataset that only used for testing statistical models to date. Separating the development and test dataset while achieving such accuracy.

In the first stage, we build an intermediate representation that mimics children's problem solving strategies through the rules using linguistic attributes like POS tags and dependency grammar labels.

Secondly, we developed a simpler model than the machine learning based complex model used in ARIS to place the verbs into their verb categories. This model expresses each verb's category in the test dataset from the development dataset with a simple binary scoring system. For a small number of verbs that are not shared in the two datasets, verb categories were found with the help of word2vec (Mikolov et al., 2013).

In addition to this, in our study, developing a flexible method in which the actions fired by the verb categories can change according to the relative positions they are in the problem. In this way, we have shown that transitions can be handled with five verb categories instead of seven.

(6.1) "Joan found 70 seashells on the beach . she gave Sam some of her seashells . She has 27 seashells . How many seashells did she give to Sam ?"

(6.2) "Sam had 9 dimes in his bank . His dad gave him 7 dimes . How many dimes does Sam have now ?"

In both example questions above, "have" verb appears with numeric values. However, there is equality meaning in the first problem while there is positive meaning in the second problem. Thus, separating these two types of appearance is crucial for accuracy. The algorithm that we implemented for extracting the equation can handle these kinds of bottlenecks.

At the last stage, we claimed that we could reach the level of statistical models with our rule-based model. In the development dataset that we implemented our model, our system has %80,07 accuracy. On the other hand, our system has %76,46 accuracy in the test dataset. This accuracy is similar to the accuracy of statistical models.

CHAPTER 7

CONCLUSION AND FURTHER STUDIES

7.1 Conclusion

In this thesis, we have proposed a problem solver to solve MWPs automatically. In this problem solver, MWPs were modeled with an intermediate representation for further analysis. This representation mechanism parallels the children's problem solving strategy according to the interviews reported by Carpenter et al. (1983).

In this study, simple MWPs were asked to a group of first-grade students and they were asked to explain their answers. From the answers given, it is understood that the children formed an IR similar to ours for problem solving.

Interviewer: Mark won four prizes at the fair. His sister Connie won thirteen prizes. How many more prizes did Connie win than Mark?
Kara: (Makes a row of thirteen red cubes. Next to this row, she makes a row from white cubes so that the white cubes are next to the last four red cubes. She counts the unmatched cubes). Nine (Carpenter et al., 1983)

The representation made by the child here coincides with the representation mechanism we will use in our model. Rows that she used matches with the states in IR while colors that she used matches with entities. Also, cubes have the same meaning with object slot while number of cubes matches with number slot in IR. The child creates two rows (states) for this problem. These rows consist of two sentences except the question statement. Using the comparison meaning in the question sentence, she gives the number of unmatched cubes (objects) in these two rows as answer. Last part is handled by "m" pointer in our system, as we described in chapter 4.

As a final step, the equation describing the MWPs was produced with the help of IR through verb categories.

In equation generation step, every slot in IR was used for different purposes as can be seen below:

1. First slot is used for entities to detect redundant information in the text.
2. Second slot is used for the action word (verbs) in the problem.
3. Third slot keeps the numeric value.
4. Fourth slot is used for objects to determine the unknown variable.
5. Fifth slot is reserved for second entities when directionality information is needed to solve the problem.
6. The Last slot keeps different pointers to capture the specific information.

This modularity in our system parallel to the modularity of mind idea introduced by Fodor (1983).

During the development of this model, a mixed development dataset consisting of 537 problems described in chapter 3 was used. After our model was developed as described in chapter 4, it was tested in the AddSub dataset with 395 questions. In chapter 5, our model's results were compared with the state-of-the-art studies and the one baseline model.

Our problem solver answered 302 of these 395 questions correctly. The accuracy of our model is %76.46. However, the AddSub dataset is divided into the three different parts. Our model reached %88.43 accuracy on one of this three sub-datasets of the AddSub. This sub-dataset has 121 problems and it is separated from other sub-datasets with the MWPs that usually lack information. As far as we know, this is the highest reported accuracy in the literature so far. Mitra and Baral (2016) has %82.14 accuracy, while the Hosseini et al. (2014) has %75.0 accuracy in information gap questions. Also, according to Hosseini et al. (2014), Kushman et al. (2014) has %51.1 accuracy in this sub-dataset.

In addition, all previous studies in the literature evaluate their models with k-fold cross-validation. This method may cause us to reach wrong conclusions about the power of the model in case of the existence of very similar problems in the dataset. In the Addsub dataset with 395 MWPs, there are 70 significantly similar MWP duplicates.

AddSub dataset contains 86 MWPs that are significantly similar to at least one other MWP in the dataset. This number is equal to %21.77 of the entire dataset. Such similarities in MWPs may lead to overfitting problem.

In order to prevent this conflict, we have paid attention to the fact that none of the problems used in development and testing phase have similarities above a threshold. However, our model's accuracy is close to the models using the k-fold cross-validation method. These results show that our model is more powerful than its statistical predecessors.

7.2 Further Studies

In the MWP solver literature, there are varied approaches for representing and solving the MWPs. We briefly explained these approaches in chapter 2. The main difference in our approach from others is its compositional nature. This compositional nature may be a source for further studies in computer-based math tutoring.

Our system is dividing MWPs to their components and reaching to the answer step by step. Since the equation for any MWP is generated by tracing the states, a rationale that explains the solution process can be generated simultaneously. We can show this with the help of an example MWP from our study:

(7.3) "Joan found 70 seashells on the beach . she gave Sam some of her seashells . She has 27 seashells . How many seashells did she give to Sam ?" In this example, our system generates an IR, as shown in 7.4 below:

(7.4) 'state1': ['Joan', 'find', '70', 'seashell', None, ''], 'state2': ['Joan', 'give', 'x', 'seashell', 'Sam', ''], 'state3': ['Joan', 'have', '27', 'seashell', 'Sam', ''], 'state4': ['Joan', 'give', 'x', 'seashell', None, 'q']

After producing this IR, our system applies an equation generator algorithm to this IR to find the equation representing our input MWP. Since this generation is handled step by step, we can print an explanation for every step respectively. In total, these explanations might be helpful to children in understanding the problem solution. An example for a rationale can be shown in four steps:

- Joan have 70 seashells after find them on the beach. [+70] (State-1)
- After finding seashells, Joan gave Sam unknown number of seashells. [+70 - x] (State-2)
- After giving seashells, Joan left 27 seashells. [+70 - x = 27] (State-3)
- Since we are asked that "How many seashells did she give to Sam?", we should subtract 27 from 70. The answer is 43.

Also, since MWPs are constructed by several sentences, they can be thought of as domain-specific discourses. Thus, our approaches might be inspiring for further studies in discourse parsing.

Bibliography

Bakman, Y. (2007). Robust understanding of word problems with extraneous information. *arXiv preprint math/0701393*.

Blum, W., Galbraith, P., Henn, H., & Niss, M. (2006). *Applications and modelling in mathematics education.* New York: Springer.

Bobrow, D. G. (1964). Natural language input for a computer problem solving system.

Bozşahin, C. (2018). Computers aren't syntax all the way down or content all the way up. *Minds and machines*, *28*(3), 543–567.

Bundy, A., Byrd, L., Luger, G., Mellish, C., & Palmer, M. (1979). Solving mechanics problems using meta-level inference.

Carpenter, T. P., Hiebert, J., & Moser, J. M. (1983). The effect of instruction on children's solutions of addition and subtraction word problems. *Educational Studies in Mathematics*, *14*(1), 55–72.

Charniak, E. (1968). Carps: a program which solves calculus word problems.

Chen, D., & Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 740–750).

Finkel, J. R., Grenager, T., & Manning, C. D. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting of the association for computational linguistics (acl'05)* (pp. 363–370).

Fletcher, C. R. (1985). Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods, Instruments, & Computers*, *17*(5), 565–571.

Fodor, J. A. (1983). *The modularity of mind*. MIT press.

Ganesalingam, M. (2013). The language of mathematics. In *The language of mathematics* (pp. 17–38). Springer.

Honnibal, M., Goldberg, Y., & Johnson, M. (2013). A non-monotonic arc-eager transition system for dependency parsing. In *Proceedings of the seventeenth conference on computational natural language learning* (pp. 163–172).

Honnibal, M., & Johnson, M. (2015). An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 1373–1378).

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2001). Introduction to automata theory, languages, and computation. *Acm Sigact News*, *32*(1), 60–65.

Hosseini, M. J., Hajishirzi, H., Etzioni, O., & Kushman, N. (2014). Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 523–533).

Jurafsky, D., & Martin, J. H. (2014). *Speech and language processing. vol. 3.* Pearson London London.

Klein, D., & Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics* (pp. 423–430).

Koncel-Kedziorski, R., Hajishirzi, H., Sabharwal, A., Etzioni, O., & Ang, S. D. (2015). Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, *3*, 585–597.

Kushman, N., Artzi, Y., Zettlemoyer, L., & Barzilay, R. (2014). Learning to automatically solve algebra word problems. In *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 271–281).

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (Vol. 10, pp. 707–710).

Lin, D., et al. (1998). An information-theoretic definition of similarity. In *Icml* (Vol. 98, pp. 296–304).

Ling, W., Yogatama, D., Dyer, C., & Blunsom, P. (2017). Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.

Marcus, M., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank.

Marr, D. (1982). Vision: A computational investigation into the human representation and processing of visual information, henry holt and co. *Inc., New York, NY*, *2*(4.2).

McCarthy, J. (1980). Circumscription—a form of non-monotonic reasoning. *Artificial intelligence*, *13*(1-2), 27–39.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111–3119).

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, *38*(11), 39–41.

Mitra, A., & Baral, C. (2016). Learning to use formulas to solve simple arithmetic problems. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 2144–2153).

Mosteller, F., & Tukey, J. W. (1968). Data analysis, including statistics. *Handbook of social psychology*, *2*, 80–203.

Murnane, R. J., Willett, J. B., Braatz, M. J., & Duhaldeborde, Y. (2001). Do different dimensions of male high school students' skills predict labor market success a decade later? evidence from the nlsy. *Economics of Education Review*, *20*(4), 311–320.

Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In *Proceedings of the eighth international conference on parsing technologies* (pp. 149–160).

Nivre, J. (2004). Incrementality in deterministic dependency parsing. In *Proceedings of the workshop on incremental parsing: Bringing engineering and cognition together* (pp. 50–57).

Nivre, J., & Fernández-González, D. (2014). Arc-eager parsing with the tree constraint. *Computational linguistics*, *40*(2), 259–267.

Novak Jr, G. S. (1976). Computer understanding of physics problems stated in natural language. *American Journal of Computational Linguistics*.

Novak Jr, G. S., & Bulko, W. C. (1993). Diagrams and text as computer input. *Journal of Visual Languages & Computing*, *4*(2), 161–175.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *Conference on empirical methods in natural language processing.*

Recasens, M., de Marneffe, M.-C., & Potts, C. (2013). The life and death of discourse entities: Identifying singleton mentions. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies* (pp. 627–633).

Roy, S., & Roth, D. (2016). Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Roy, S., & Roth, D. (2017). Unit dependency graph and its application to arithmetic word problem solving. In *Thirty-first aaai conference on artificial intelligence.*

Roy, S., Vieira, T., & Roth, D. (2015). Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics*, *3*, 1–13.

Shi, S., Wang, Y., Lin, C.-Y., Liu, X., & Rui, Y. (2015). Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 1132–1142).

Tesnière, L. (1959). Éléments de syntaxe structurale.

Thede, S. M., & Harper, M. (1999). A second-order hidden markov model for part-of-speech tagging. In *Proceedings of the 37th annual meeting of the association for computational linguistics* (pp. 175–182).

Verschaffel, L., Greer, B., & De Corte, E. (2000). *Making sense of word problems*. Swets & Zeitlinger Lisse.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, *13*(2), 260–269.

Wang, L., Zhang, D., Gao, L., Song, J., Guo, L., & Shen, H. T. (2018). Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Thirty-second aaai conference on artificial intelligence.*

Wang, Y., Liu, X., & Shi, S. (2017). Deep neural solver for math word problems. In *Proceedings of the 2017 conference on empirical methods in natural language processing* (pp. 845–854).

Zhou, L., Dai, S., & Chen, L. (2015). Learn to solve algebra word problems using quadratic programming. In *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 817–822).