Conversational Mathematics

Independent Study Thesis

Presented in Partial Fulfillment of the Requirements for
the Degree Bachelor of Arts in the
Mathematics and Computer Science at The College of
Wooster

by
Dylan Orris
The College of Wooster
2019

Advised by:

Dr. Fox (Mathematics and Computer Science)

THE COLLEGE OF
WOOSTER

Abstract

Include a short summary of your thesis, including any pertinent results. This section is *not* optional for the Mathematics and Computer Science or Physics Department ISs, and the reader should be able to learn the meat of your thesis by reading this (short) section.

This work is dedicated to the future generations of Wooster students.

Acknowledgments

I would like to acknowledge Prof. Lowell Boone in the Physics Department for his suggestions and code.

Contents

Abstract v

Dedication vii

Acknowledgments ix

Contents xi

List of Figures xiii

List of Tables xv

List of Listings xvii

CHAPTER PAGE

1 Introduction 1
- 1.1 Problem Statement 2
  - 1.1.1 Project Aims 2
    - 1.1.1.1 Natural Language Processor 3
    - 1.1.1.2 Automated Theorem Prover 3

2 Background on Natural Language Processing 5
- 2.1 Early Years 5
  - 2.1.1 SHRDLU 5
  - 2.1.2 ELIZA 8
- 2.2 Advancments of the Twentieth Century 9
  - 2.2.1 Context Free Grammars 9
  - 2.2.2 Advancements of the 1970s and 1980s 11
- 2.3 Syntax and Semantics 12

References 17

List of Figures

Figure		Page
2.1	Sample SHRDLU starting world	6
2.2	Sample SHRDLU dialogue	6
2.3	Flowchart for 2.2	7
2.4	SHRDLU world after 2.2	7
2.5	Sample ELIZA dialogue	8
2.6	Flowchart for dialogue in 2.5	9
2.7	Two production rules, one of which produces a variable and one of which produces a terminal	10
2.8	A production rule which produces a terminal	10
2.9	Sample Context Free Grammar	11
2.10	Sample Semantic Analysis of a Sentence	13

List of Tables

Table Page

List of Listings

Listing Page

CHAPTER 1

Introduction

Over the past decade, natural language processors have become more and more commonplace. In late 2011, Apple introduced their digital assistant, Siri, to iOS. Siri was the successor to voice control, an iOS feature that allowed the user to interact with their phone using their voice, though in both an extremely limited fashion and rather unreliably. Siri was powered by a much better voice recognition model. Not only could it process many commands into actions, but it could even ask for clarification based on context. Since then, Google introduced the Google Assistant to its Android and ChromeOS platforms, Amazon released smart home products powered by Alexa, and Siri has been ported to MacOS and WatchOS. All of these digital assistants rely on understanding the commands of users, and the only way this is possible is through natural language processing.

From Q2 2017 to Q2 2018, smart speakers, the quintessential aspect of the smart home to many, had shipments grow 187% [**?**]. This growth is expected to continue in the coming years, with smart speakers and other smart home technologies becoming an enormous industry. As this industry expands, there will be greater and greater access to these systems among the general population. Smart devices offer more than simply convenience, they could become an amazing source of educational opportunities which are not otherwise available for many. For example, what if it was possible for a student to ask their smart screen why a mathematical claim was true, and that system could respond by displaying the proof?

While smart devices do not appear in this project, we have created a front end interface for students to more easily investigate mathematical questions, without the barrier of technical language.

1.1 Problem Statement

Automated theorem proving systems are a powerful tool which can allow a student to check their work or a researcher to prove a claim which could be extremely time consuming if done by hand. Unfortunately, these systems are not trivial to use, with the use of any such system requiring the user to learn the proper method for input to that system. As a result, these systems are quite intimidating to learn, and once one learns one such system, it is easy to be stuck using what is already known, even if better options may be available. A wonderful solution to this problem is leveraging computational power to convert a human language request, e.g. "Prove that the length of the third side of a triangle is determined by the two other sides and the angle between them" into a language the proving system understands.

1.1.1 Project Aims

This project has two separate but interconnected goals; the production of a natural language processor, and the creation of a basic automated prover. The processor:

- Serves as a front end for any automated prover.
- Is usable without any understanding of symbolic notation or programming.
- Is modular enough to become the front-end to any automated proving system.
- Accurately translates standard English to symbolic notation.

The prover had far more modest aims:

- Understand basic questions in a limited scope, such as geometry or arithmetic.
- Take symbolic input, and return the truth value of said statement.
- Display the steps of the proof to the end user.

As the two pieces are separate, the processor is not limited by the prover, and may instead be joined with a more robust system by specifying what said system is. This is necessary to convert the request properly, so that the processor knows which symbolic language to translate to.

1.1.1.1 Natural Language Processor

The processor will takes a user's input in English, and determines the goal of the user based on the content of that input. The user is not required to know anything, other than that the system is intended for mathematical queries and will be able to do nothing else. Once the input is analyzed, the processor converts it to the proper symbolic language for the automated prover.

1.1.1.2 Automated Theorem Prover

The prover takes a mathematical claim, in a symbolic notation, and applies axioms, theorems, and other knowledge as necessary to prove or disprove the claim. If the prover is able to use theorems in its proofs, it is barred from using them in such a way as to give the appearance of circular logic. For example, when a user asks for a proof of the Pythagorean Theorem, the prover may not use the Pythagorean Theorem as its justification for the truth of the theorem. Should the user ask for a proof of an axiom, it is permissible to simply point to the axiom. The prover is not expected to explain an axiom to a user.

CHAPTER 2

Background on Natural Language Processing

2.1 Early Years

Natural language processing has been an area of interest in computer science for the past 60 years. Initially, the subject was limited to machine translation, with the Georgetown experiment in 1954 being an early foray into the field. In this experiment over sixty Russian sentences were translated to English entirely automatically. The results were promising enough that the researchers anticipated automatic translation being solved within the next five years. [**?**] Progress was, obviously, much slower than this.

Funding was cut when the program had run twice as long as promised, and progress in the field slowed. Two successful systems were created in the 1960s, SHRDLU and ELIZA.

2.1.1 SHRDLU

SHRDLU was a querying system which created a small "Blocks World" which was populated by cones, spheres, cubes, and other geometric shapes of various sizes and colors. [**?**] The user was able to instruct SHRDLU on how to move these objects around simply by specifying the shape in addition to its color or size. SHRDLU also had a simple memory system, allowing for reference to objects recently interacted with. This memory also allowed for SHRDLU to be queried on what she had previously done. SHRDLU's world contained basic physics, allowing for the program to describe what was possible in the world and what was not. The final major feature was to remember the name a user gave an object or collection of objects. From this, it was possible to more easily instruct SHRDLU.

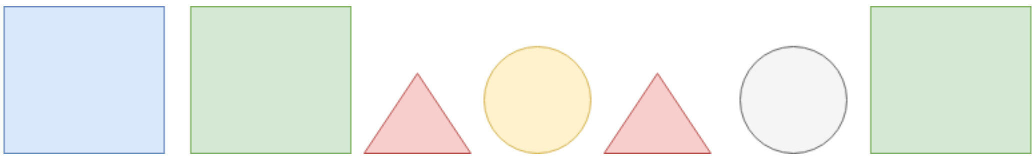

Figure 2.1: Sample SHRDLU starting world

> Hello, I am SHRDLU
* Are there any boxes in the world?
> There are three boxes. Two are green and one is blue.
* Put a green box on the red one.
> There is no red box.
.
.
.
*Put a red cone on top of the blue box.
> Okay.
* Can you put the other cone on top of the cone?
>I assume by other cone you mean
the cone on the ground. I don't know.
*Try it.
>I cannot put one cone on top of another.

Figure 2.2: Sample SHRDLU dialogue

SHRDLU would remember changes in the environment until reset after use. From this simple technique, it became much easier to consider SHRDLU to inhabit a real world, as locations were consistent and basic physics were the same as reality. While SHRDLU's domain was small, effective code and good design helped making

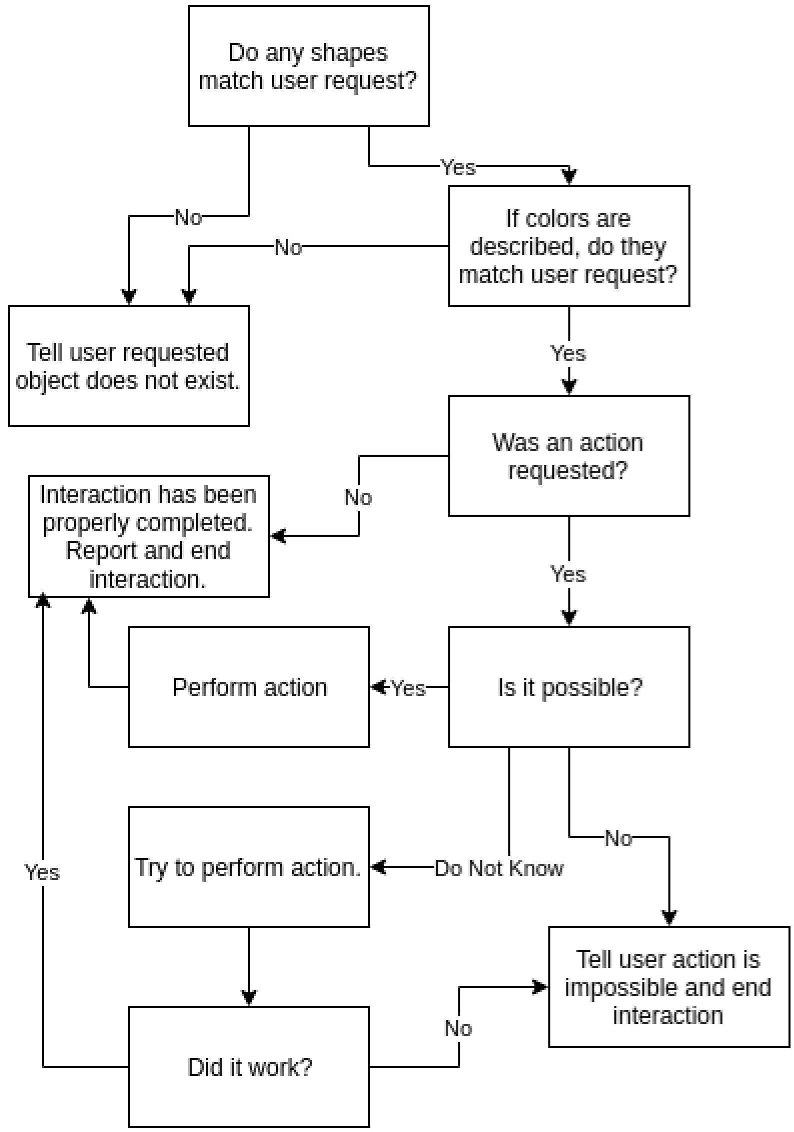

Figure 2.3: Flowchart for 2.2

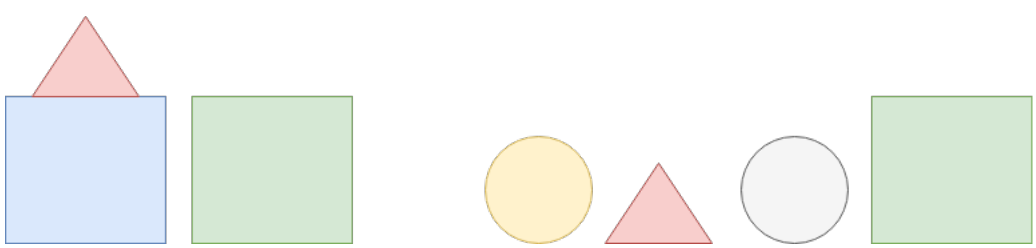

Figure 2.4: SHRDLU world after 2.2

for a convincing approximation of both a world and an intelligent entity which allowed interaction with that world.

2.1.2 ELIZA

ELIZA is an early language based program which took user input and responded much like a Rogerian psychologist [**?**]. She was created by Joseph Weizenbaum around 1965 at MIT, using pattern matching and substitution to give an illusion of understanding. While ELIZA has only a very small vocabulary, she convinced many users that she was truly intelligent through the use of these techniques, despite being completely unable to go into detail on almost all subjects.

ELIZA uses NLP in a very different way from SHRDLU – rather than attempting to understand what the user is inputting, she instead uses the context surrounding phrases, such as “I feel” or “I am” to insert the phrase properly into previously constructed sentence types.

```
>Hello, I am ELIZA
*Hello, I am Tom
.
.
.
*I feel trapped by my parents
>Do you often feel trapped by your parents?
*Yes
>Tell me more
```

Figure 2.5: Sample ELIZA dialogue

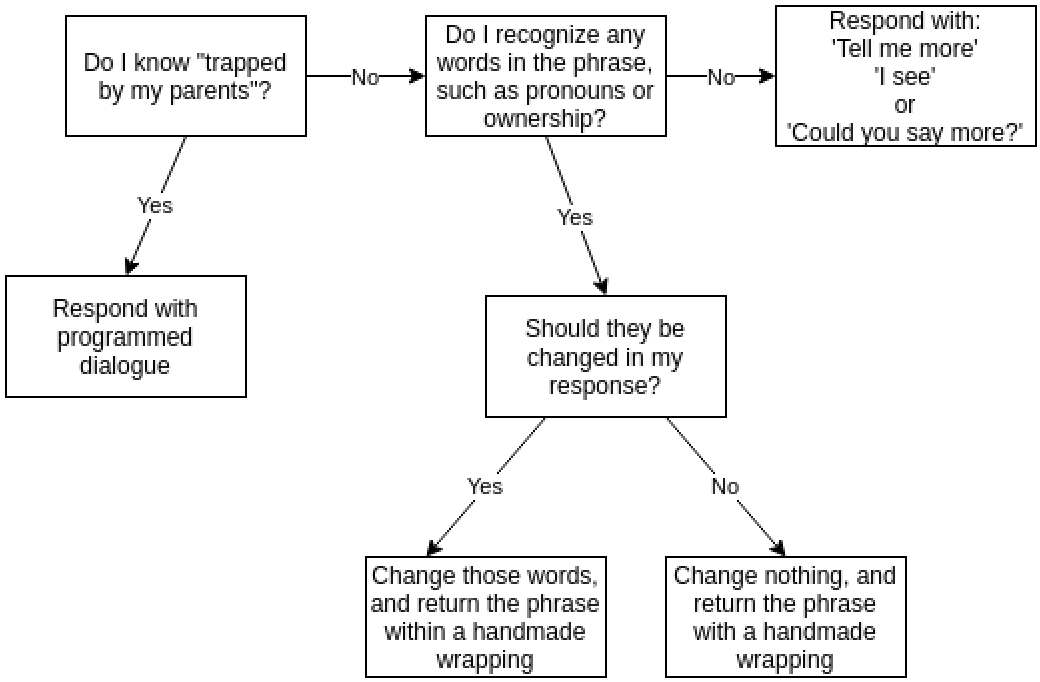

Figure 2.6: Flowchart for dialogue in 2.5

2.2 Advancments of the Twentieth Century

2.2.1 Context Free Grammars

To understand the functioning of NLP systems, context free grammars (CFGs) should first be described. CFGs are a set of rules, known as *production rules* which describe all strings which can possibly be produced within a grammar. Here, string simply means a sequence of words, such as, "The man walked down the road" and grammar refers to the possible structure of strings, and to the words which may be included in these strings. There are two basic elements of a CFG – variables and terminals. A variable is a part of the grammar which will be modified by continuing to follow the production rules which are described by the grammar. We denote variables by enclosing them in angle brackets ('<' and '>'). A terminal is a word, such as "the", which is in its most basic form and undergoes no further changes.

This is most easily seen through an example. Consider the following two figures:

$$< S > \rightarrow < B >$$

$$< B > \rightarrow \text{Dog} \mid \text{Pig}$$

Figure 2.7: Two production rules, one of which produces a variable and one of which produces a terminal

$$< S > \rightarrow \text{Cat}$$

Figure 2.8: A production rule which produces a terminal

In each of the cases, we have the variable $< S >$ on the left of the arrow. To the right of the arrow is what will replace $< S >$, which is another variable $< B >$ in 2.7, and a terminal ("Cat") in **??**. Notice that in 2.7, the second variable has two terminals on the right of the arrow with '|' between them. This indicates that $< B >$ may be replaced by "Dog" or "Pig". Variables may follow rules which are:

- one-to-one, where the variable is replaced by a single other variable or terminal.
- one-to-many, where the variable has several variables or terminals which may replace it.
- one-to-none, where the variable is replaced by nothing (blank space).

The first variable used for a CFG is $< S >$, which stands for "start". In language processing, $< S >$ will typically produce a noun-phrase ($< NP >$) and verb-phrase ($< VP >$), which then produce many possible sentences by defining the location of parts of speech. Each variable serves as a part-of-speech tag, such as noun, verb, or adjective. For this reason, CFGs are an effective way of capturing the inherent structure of language. A valid sentence always contains a noun and a verb, for example, and the CFG will be unable to produce a sentence without a noun or a verb, assuming it is constructed properly. Thus an incorrect sentence such as "Jumped." will not be produced by our grammar. This allows for a somewhat simple checking of proper English sentences - if we know the parts of speech of each word, we can either work our way up from the sentence to see if $< S >$ could have produced it, or work our way down and see if the sentence structure appears.

2.2.2 Advancements of the 1970s and 1980s

Come the 1970s, William Woods introduced augmented transition networks (ATNs) as a method to represent language input, rather than phrase structure rules [?]. Phrase structure rules are typically known as CFGs. Let us consider a CFG structured as follows:

- $<S>\rightarrow<NP>\quad<VP>$
- $<NP>\rightarrow<DET>\quad<N>$
- $<VP>\rightarrow(<ADVB>)\ \ <VB>$
- $<N>\rightarrow$ Dog
- $<DET>\rightarrow$ The | A
- $<VB>\rightarrow$ Runs | Jumps
- $<ADVB>\rightarrow$ Excitedly

Figure 2.9: Sample Context Free Grammar

Here, we have a simple grammar which breaks a sentence into noun and verb phrases, which then break into a determiner and a noun, and a possible adverb and verb, respectively. Once these variables are reached, they lead to terminals, which here are the English words which are represented. In this grammar, the sentence "The dog excitedly runs" is valid, while "The dog happily jumps" is not. The second sentence is not incorrect due to any issues with the English grammar, but rather due to the given grammar not accounting for the terminal 'happily'.

ATNs use finite state machines in order to parse sentences. Woods claimed that, by adding a recursive mechanism to finite state models, it was possible to parse much more efficiently. The system builds a set of finite state automata, which have transition states between them. Should a sentence reach a final state, the sentence is valid. These systems have advantages, including the delaying of ambiguity. Rather than simply guessing a path as some systems will, the ambiguity may be delayed until more of the sentence has been parsed, allowing for greater information to be used in resolving said ambiguity. Additionally, they effectively capture the structure of languages, allowing for ease of processing [**?**].

The 1980s, with the introduction of machine learning algorithms, led to immense changes. No longer were parsers built based on complex rules formed by the programmer, instead, algorithms like decision trees began to make the classification rules. Eventually, this change led to the use of modern statistical models, which assign probabilities to words for part-of-sentence identification, rather than rigid if-then rule sets [**?**].

2.3 Syntax and Semantics

When examining natural language, meaning is found through an analysis of both syntax and semantics. Syntax consists of the rules which determine the structure of sentences in a language, for example, English requires a verb and subject for a sentence to be grammatically correct. Semantics refers to the meaning of words within the language, such as the word 'dog' referring to a four-legged, furry animal which descends from wolves and was domesticated by humans. Early implementations of natural language processors typically focused on one of these aspects of language to the detriment of the other [].

Certain camps believed that, by simply knowing the definition of each word in a sentence, their relationship to one another would be determinable. Others saw syntax as the defining feature to study, so broke sentences down based purely on anticipated structures of sentences, fitting the sentences into some hypothetical structure which could be determined by a CFG.

Let us first examine the issues with a purely semantic analysis of a sentence. Consider 'The dog ran to the man who owned him.' Each word can easily be defined, but the meaning of the sentence becomes extremely unclear. In this situation, it is clear that the final three words 'who owned him' refer to a relationship between the man and dog. However, we may not examine the structure of the sentence, as it was regarded as unnecessary. The 'him' in this context has an unclear referent without examining previously discussed entities.

- The → Indexical
- Dog → Canine
- Ran → Moved Quickly
- To → In The Direction Of
- The → Indexical
- Man → Adult Male Human
- Who → Identity Request or Explanation
- Owned → Posessed
- Him → Male Pronoun

Figure 2.10: Sample Semantic Analysis of a Sentence

Even disregarding this issue, a hypothetical system using this methodology would be completely powerless to even attempt to understand a sentence containing a word it did not already have the definition of. The system, assuming parsing completed, would return a description of each word other than the unknown. Without understanding the meaning of this region, the meaning of the sentence becomes unintelligible. This system can also never attempt to determine potential meaning, as relationships between words are completely ignored.

With some issues of purely semantic analysis discussed, let us now move on to the issues facing a purely syntactic analysis. First, consider the possible shapes of every sentence in English. While they look different from one another in more than simply word count when we understand that a noun follows an article, a linking verb comes between two nouns, and so on. By seeing how each component comes together, we can easily match sentences to their derivation from a CFG. However, by purely examining syntax, we cannot see the part-of-speech of the component words. What we are left doing is taking an n-word sentence, and applying to it the structure of every possible n-word archetype.

The introduction of syntactic analysis brings ambiguity into our parser as well. When the meaning of a word is known to the parser, ambiguity can often be resolved. As natural language is full of ambiguity, this is not always true, but understanding the meaning can resolve situations such as "The man walked down the road wearing a hat".

Insert Example Parse Trees Here

When we don't understand what a 'road' and 'hat' are, we find ourselves unsure which noun is wearing the hat. The sentence could be parsed either way, and would be valid. The best way to resolve these issues is to consider syntax and semantics simultaneously.

A semantic consideration need not be of the definition of the word, but can instead consist of part-of-speech tagging, or a way of indicating what types of actions it can take. In the **Whatever the Man in Hat Example Figure is** example, simply knowing that roads, hats, and men are nouns will not help to resolve the ambiguity.

However, if we know that men have a property such as *can wear apparel* and the hat is part of a class *apparel*, we know that the man wearing a hat is valid. Inspecting the road, we will see it has no *can wear apparel* property, and thus, the sentence in which the road wears the hat is invalid. Recognition of word position and relations between words allows semantic analysis to provide a far better description of the content of each word.

References