Translating Natural Language Propositions to First Order Logic

A thesis submitted
in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology

by

Naman Bansal

to the
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
May, 2015

CERTIFICATE

It is certified that the work contained in the thesis titled **Translating Natural Language Propositions to First Order Logic**, by **Naman Bansal**, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

Amey Karkare 8/5/2015

Dr. Amey Karkare
Department of Computer Science & Engineering
IIT Kanpur

Sumit

Dr. Sumit Gulwani
Microsoft Research
Redmond

May, 2015

ABSTRACT

Name of student: **Naman Bansal** Roll no: **13111035**

Degree for which submitted: **Master of Technology**

Department: **Computer Science & Engineering**

Thesis title: **Translating Natural Language Propositions to First Order Logic**

Name of Thesis Supervisor: **Dr. Amey Karkare**

Month and year of thesis submission: **May, 2015**

Translating Natural Language Propositions into First Order Logic (FOL) is a key component of Logic course taught in first year of graduate programme. Students face a lot of difficulty in understanding this translation process. Major reason for this has been characteristics of Natural Language and the ambiguity in natural language sentences. Building a tutoring system for this problem requires us to have a system which could automatically translate Natural Language sentences to First Order Logic.

In this thesis we make two contributions. First, we have built a benchmark suite for the problem of translating natural language sentences to FOL. We collected more than 350 natural language sentences and annotated them with correct First Order Logic formula and list of predicates. Secondly, we propose an algorithm to solve the problem of translation for our domain and build a system which, given a natural language sentence and list of predicates that satisfy the sentence, outputs first order logic translation for it. A clustering of the benchmarks based on the First Order Logic structure shows that we require 248 different structures to cover our benchmark suite.

Dedicated to my

Grand-Mother: Kaushal Bansal

Grand-Father: Shyam Sunder Lal Bansal

Mother: Pawan Rekha Bansal

Father: Neeraj Bansal

Brother: Nayan Bansal

family, teachers and friends

Acknowledgements

This project would have never been possible without the guidance and support of my advisors **Dr. Amey Karkare** and **Dr. Sumit Gulwani**. I am extremely thankful and express my gratitude to them for accepting me as their student. They always encouraged my ideas and helped me in my thought process while brainstorming for this project. I also did a summer project on ESC-101 under them and was TA under Amey sir for two semesters. It has been a great journey. I am truly indebted to them for all the opportunities that were given to me.

I would also like to thank **Umair Z Ahmed**. Implementing this project would not have been possible without his expertise. We had numerous meetings during the course of this project and I will definitely miss our discussions.

I would also like to thank **Ayush Sekhari** and **Sanil Jain** who worked on the project during summers. I would also thank my friends **Rajdeep Das, Puneet Gupta and Saurabh Srivastava** who supported me during my stay at IIT Kanpur. I also thanks all the teachers at IIT Kanpur and the Department of Computer Science for providing me world class environment to learn and grow as a student.

I would also like to thank my parents and family who have always supported me. I would express my gratitude to my Mother and Grand-Mother who always showed interest in my project and encouraged me to do my best.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Education is the single most important aspect of ones life that shapes his future. In fact, it is our education system that keeps accumulating human knowledge and makes sure that next generation learns on the building blocks of the previous ones and we, "humans", keep progressing as a race. We have realized that and currently have a lot of focus on education. Success of Massive Open Online Courses $\left(\text{MOOCs}\right)$ [1, 2] is a very big example for that. MOOCs have helped us to increase the reach of our education system, with MOOC now we can aim to provide education in remote places. Technology has also helped a lot in this aspect, in near future we could expect to have MOOC free for all, infrastructure investment is not much when compared to the reach. MOOCs offer quality education but they still have one problem that it is not possible to clear the doubts of all the student or help them in the learning process. Let us look at the traditional education model, we have an instructor who provides us with the knowledge about the subject and the most important aspect of this setting is that we get personalized feedback. In a MOOC setting it is not feasible for the instructor to provide the personalized feedback, so they mostly resort to the discussion forums, which are effective to a certain measure. To tackle this problem we have seen a lot of research in Intelligent Tutoring Systems.

Intelligent Tutoring System $\left(\text{ITS}\right)$ targets a particular domain and act as a virtual tutor which could learn where the student has problem and help him accordingly, based on the weak areas that it has identified for the student, just like an instructor in traditional education model. ITS aim to provide personalized feedback in a sense that helps student

learn the concept better and teaching them basic concepts underlying a problem. ITS have three main features - solution generation, feedback generation and problem generation. Each one targets a different aspect of teaching. We describe these features in detail.

1. **Solution Generation:** It aims at generating solution to a given problem. It is the first step at building a Intelligent Tutoring System, it gives us the intuition about the features of the problem that we are solving. Once we understand how the problem is being is solved and how our system understands the problem, it is easy to aim for the feedback generation. Solution generation for the problems in Propositional Logic [3] has been done.

2. **Feedback Generation:** It aims at generating feedback for a particular problem if a student is not able to solve it. One way to look at it is, say, we have a path containing multiple steps which leads to the solution. We already know all these paths and steps using the solution generation algorithm. Now, if a student takes a different path which doesn't leads to the solution, then we can give feedback to the student based on the path which is uncommon between the correct path and the path taken by the student to approach the problem. AutoProf [4] generates feedback for Introductory programming course in Python.

3. **Problem Generation:** It aims at generating multiple problems of same difficulty for a given problem. Difficulty is a major aspect as it helps us classify and differentiate among the problems. One way of problem generation is to have templates for the problems. Problem generation is particularly significant since we can have unique problems for each student and the instructor would have to care less about the cheating and plagiarism. Automatic problem generation for algebra problems [5] has been done.

Intelligent Tutoring System have been developed for various problems. A lot of research has been going on in this field, ITS for Geometry [6], Algebra [5], Automata-DFA and NFA construction [7], Propositional Logic [3] and Introductory Programming in Python [4] have been developed.

1.1 Problem Statement

In this thesis we solve the problem of translating a given Natural Language sentence in English to its corresponding First Order Logic Translation.This enables us to do Solution Generation. We have designed the rules such that they motivate all the components of ITS. We also give an algorithm for the problem of translating given First Order Logic into its corresponding translation in English. Sample space for us is the problems in the text book of the graduate level course, where introductory translation of NL to FOL and vice versa is taught.

Some example sentences that we have collected from the graduate level book *The Essence of Logic* [8]:

1. **NL:** If Barbara practices she will win

   Predicates: Practices(x):x practices; Win(x):x will win

   FOL: Practices(Barbara) $\rightarrow$Win(Barbara)

2. **NL:** All grass is green

   Predicates: Grass(x):x is grass;Green(x):x is green

   FOL: $\forall$(y)(Grass(y) $\rightarrow$Green(y))

3. **NL:** There is a winning combination

   Predicates: Combination(x):x is a combination;Winning(x):x wins

   FOL: $\exists$(x) (Combination(x) $\wedge$Winning(x))

Solving this problem enables us to target the ultimate goal which would be to build a ITS for graduate level course in Logic where students are taught to translate Natural language sentence to corresponding First Order Logic formula.

We give an approach to solve this problem and build a system for the same. We also build a framework for testing out various algorithms. Our approach is rule based, as in, we use rules based on the structure of the sentence to solve the current problem of translation.

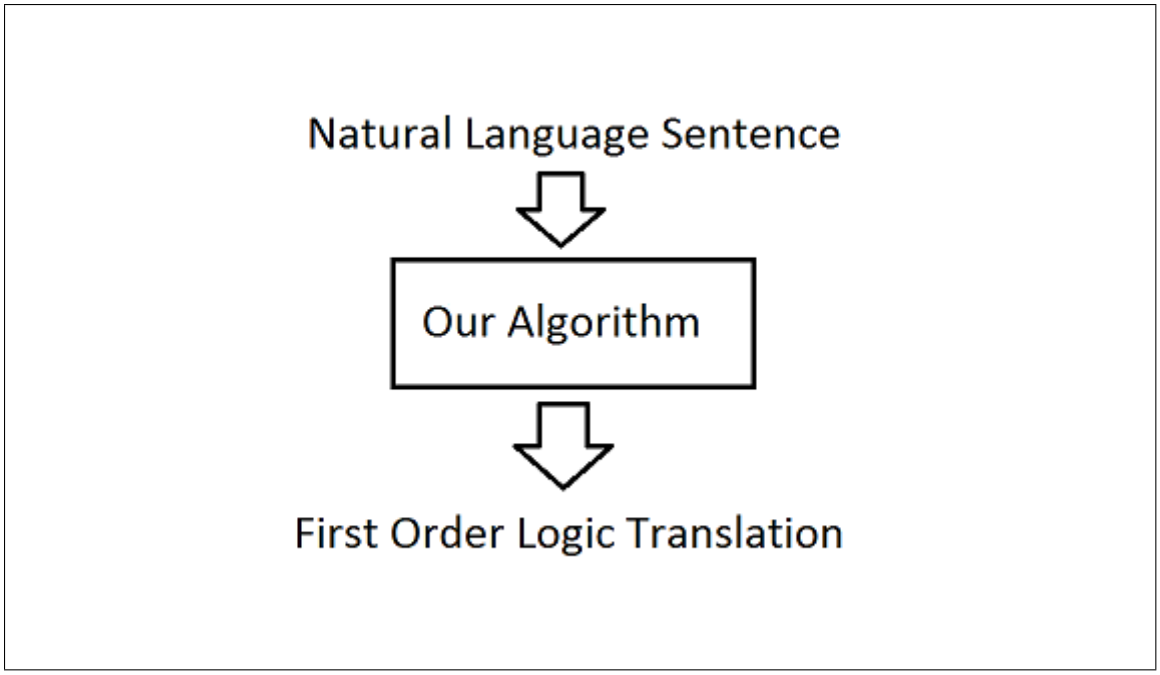

Figure 1.1: High Level Description of our Problem

1.2 Motivation

Translating Natural Language sentence into First Order Logic is a key component of Logic course taught in first year of graduate programme. Students face a lot of difficulty in understanding this translation process [9]. Major reason for this has been the characteristics of the Natural Language and the ambiguity in it.

Building a tutoring system for the graduate level First Order Logic course taught in the first year to the students requires us to have a system which could automatically translate Natural Language sentences to First Order Logic.

Given an English sentence and a set of Predicates which satisfy the sentences, students are required to translate it into First Order Logic formula. This translation is often not that intuitive and often students give translations which are incorrect. Interesting point to note here is that often it is the case that the students think that their translation is correct but it is not.

If we can show students what their translation actually means we would be helping them understand that their First Order Logic translation is incorrect and actually means something else.

This is main motivation of the project is to help the student understand the First Order Logic translation more effectively and help them clear their intuition about the subject [9]. In MOOCs it is impractical for the teacher to give feedback to all the students about what their translation actually mean. Giving students feedback about their mistake helps them to understand better. There are a lot of problems in the course and a lot of students as well. Both the reason demand for an intelligent system to tutor the students.

First Order Logic translation captures the semantics of the given English sentence. Capturing semantics is a topic of current research in Natural Language Processing Community. If we are able to get effective translation then we will also be able to help the Natural Language Processing community by pushing the envelope, but that is a very ambitious goal. Our main focus in this thesis would be to build a system which could effectively translate Natural Language sentences to First Order Logic. If we are able to build a system which is effective enough in the Intelligent Tutoring Domain, for Solution Generation, then we will be able to target a system which could aim at Problem Generation and Feedback Generation, which would be the ultimate goal for building a complete Tutoring System for teaching students translation between Natural Language to First Order Logic and vice versa.

1.3 Our Contribution

We make following contributions in this Thesis:

- Build a benchmark for the current problem statement
- Design an algorithm to solve the current problem of translation from NL to FOL and vice versa.
- Build a system to implement the algorithm.
- Build a test suite to test the algorithm developed against the benchmark that we have created.
- Build a framework where different algorithm can be implemented for this problem.

Chapter 2

Related Work

Intelligent Tutoring System has been a hot topic of research recently. One example for that is Automata Tutor [7], this tool [10] does automatic grading of the DFA and NFA submitted by the students, it also provides feedback in case the solution submitted by the student is incorrect. It has been used by a lot of professor across the world [11] in their course, to teach the students DFA and NFA construction.

AutoProf (Automated Program Feedback) [4] provides automatic feedback to the students for assignments in Introductory Programming, tool has been integrated with EdX course "Introduction to Computer Science and Programming (6.00x)".

There has been an extensive study done by a research group, as to why students find translating Natural Language sentence into First Order Logic hard [9]. This research group has collected over 4.5 million translations provided by around 55,000 students from 50 countries worldwide over a period of 10 years [12]. They have named this corpus as Grade Grinder. In their paper [9] they have classified the problems that students face in converting Natural Language sentences to First Order Logic into four categories, first, Hard to Get Wrong, Easy to Resolve, second, Easy to Get Wrong, Easy to Resolve, third, Hard to Get Wrong, Hard to Resolve, fourth, Easy to Get Wrong, Hard to Resolve.

Solution generation for Natural Deduction has been done by Umair et al. [3]. Authors in this paper construct a Universal Proof Graph over all possible formulations generated by applying inference rules. Then they do a forward search in this graph to reach the solution. Interestingly, they can also do problem generation using this technique itself. First, they

obtain an abstract proof of the given problem by doing a forward search and then using this abstract proof they perform a backward search to find similar instances of the abstract proof obtained from the original problem. This enables them to do problem generation.

Successful attempts have been made to translate Natural Language to some query logic where the domain of the Natural Language is fixed [13]. These method mostly leverage the fixed Domain by designing a Domain Specific Language (DSL) which is expressive enough to capture the operation required for the domain. NLyze: Interactive Programming by Natural Language for Spreadsheet Data Analysis and Manipulation [13] shows how Domain Specific Language can enable us to do seemingly complex task of converting "Users intent", in context of the spreadsheets' temporal and spatial specification, into an underlying program representing users intent. Main ideas mentioned by the authors are keyword programming and semantic parsing, helping them to achieve high precision and high recall.

Attempts have been made to show how restricted Natural Language can replace First Order Logic [14]. In this paper the authors show that Attempto Controlled English (ACE) can replace First Order Logic for certain tasks. Attempto Controlled English (ACE) is a subset of English Language which can be translated to First Order Logic unambiguously.

Machine Learning has also been applied to understand the semantics of the Natural Language. In [15] authors show that natural language can be mapped to lambda calculus to capture its meaning. They do this for task of learning natural language interfaces to databases. Authors do this by, first, annotating the natural language benchmarks with the corresponding lambda calculus translation and then, learn a combinatory categorial grammar (CCG) using the benchmarks. CCG learns all the different interpretations of the words in the benchmarks. Interpretations are then combined to get the translation. Techniques using CCG work well when the domain is restricted, this is so because the CCG is learnt over the benchmarks and is restricted by it. When we use CCG for specialized task like mapping instructions to actions [16] the possibility of capturing interpretations which satisfy the domain are increased by using a lexicon or benchmark. This work could not be applied to our problem as, though our domain is restricted to graduate level problems but

the sentences that we target are general and not limited by some specialized underlying task.

Chapter 3

Background

3.1 First Order Logic

We Describe First Order Logic $\left(\text{FOL}\right)$ used to describe the logical form of the Natural Language Sentence. First Order Logic is very similar to Object Oriented Programming $\left(\text{OOP}\right)$ concept in terms that it also relates to Objects like people, sun, earth, color, etc. Main concept of FOL are:

- **Objects:** people, sun, earth, color, etc.

  They are the constants in FOL.

- **Relation:** brotherOf, sonOf, smallerThan, etc.

  They are predicates in FOL and return True or False.

- **Functions:** fatherOf, motherOf, grandfatherOf, etc.

  Functions are predefined and return a object.

In this section we describe the syntax of First Order Logic:

1. **Term:**

   - **Constants:** These are the fixed properties/individuals like Sun, John, Mary, etc.

- **Functions:** It is a mapping that maps N terms to a single term.

$$f\big(t_1, t_2, \ldots, t_n\big) = t_x$$

e.g. colorOf(Sky)= Blue

2. **Predicates:** It is a mapping that maps N terms to True or False.

$$f\big(t_1, t_2, \ldots, t_n\big) = True$$

$$or$$

$$f\big(t_1, t_2, \ldots, t_n\big) = False$$

In other words, Predicate is a function that returns True or False.

e.g. smaller(5,3)

Here, smaller is a predicate, defined as,

smaller(x,y): x is smaller than y

$$smaller(x, y) = True \quad if \quad x < y$$

$$smaller(x, y) = False \quad if \quad x \geq y$$

3. **Logical Connectives:** There are five types of connectives that we have in First Order Logic. These connectives define the value of expression when applied on Predicates combined with quantifiers.

   - **And:** Represented by symbol $\wedge$. And is true only if all the predicates are true. Truth table for AND is shown in the following table.

A	B	A∧B
0	0	0
0	1	0
1	0	0
1	1	1

Table 3.1: Truth Table for AND

- **Or:** Represented by symbol ∨. OR is true if one or more input is true. Truth Table for OR is shown in the following table.

A	B	A∨B
0	0	0
0	1	1
1	0	1
1	1	1

Table 3.2: Truth Table for OR

- **Not:** Represented by symbol ¬. NOT is a Unary operator and inverts the value of the input. Truth Table for NOT is shown in the following table.

A	¬A
0	1
1	0

Table 3.3: Truth Table for NOT

- **Implies:** Represented by symbol →. IMPLIES produces false as output only when first input is true and second input is false, for all the other combinations it gives true as output. Truth Table for IMPLIES is shown in the following table.

A	B	A→B
0	0	1
0	1	1
1	0	0
1	1	1

Table 3.4: Truth Table for IMPLIES

- **Equivalent or Bi-implication:** Represented by symbol $\leftrightarrow$. BI-IMPLIES gives output as true when both the inputs are same and gives output as false for rest of the cases. Truth Table for BI-IMPLIES is shown in the following table.

A	B	A↔B
0	0	1
0	1	0
1	0	0
1	1	1

Table 3.5: Truth Table for BI-IMPLIES

4. **Quantifiers:** There are two types of Quantifiers, Existential, represented by symbol $\exists$ and Universal, represented by symbol $\rightarrow$.

   - **Existential:** $\exists$ quantifier says that there is at least one instance in the domain for the variable such that the predicate or FOL logic is true.

     e.g. Sentence "Some coins are round" would be translated in FOL as:

     $\exists(x)\ \text{Coin}(x)\ \wedge\text{Round}(x)$

     It means that there is at least one coin that is round.

   - **Universal:** $\forall$ quantifier says that for all the instance of the variable in the domain the predicate or FOL is true.

     e.g. Sentence "All men are mortal", would be translated in FOL as:

     $\forall(x)\ \text{Man}(x)\ \rightarrow \text{Mortal}(x)$

     It means that that all men are mortal.

5. **First Order Logic Expression:** They are formed by combination of the fundamental building blocks that we defined above.

   e.g. let Gardener(x): x is a gardener and Likes(x, y): x likes y

   Every gardener likes the sun.

   $\forall(x)$ Gardener$(x) \rightarrow$ Likes$(x$,sun$)$

   e.g. let Grass(x): x is grass and Green(x): x is green

   All grass is green.

   $\forall(x)$ Grass$(x) \rightarrow$ Green(x)

3.2 Natural Language Processing Concepts

In this section we describe some terms related to Natural Language Processing. These terms have been used in this Thesis to describe the work done.

1. **Part of Speech Tagging:** This process tags a given sentence with Part of Speech, namely, Noun, Adjective, Verb, Preposition, etc. The tags given by a POS Tagger are standard, which are given by Penn Tree Bank.

   Some of the tags are:

   - NN: Noun, singular
   - NNS: Noun, plural
   - NNP: Proper Noun, singular
   - VB: Verb, base form
   - VBZ: Verb, 3rd person singular present
   - JJ: Adjective
   - IN: Preposition
   - PRP: Personal Pronoun

   An example, when we give a sentence to Stanford parser for Part of Speech Tagging.

   Input: Man is Mortal

Output: Man/**NNP** is/**VBZ** mortal/**JJ**

2. **Lemmatization:** This process groups the different inflected forms of a word. According to Wikipedia, "In grammar, inflection or inflexion is the modification of a word to express different grammatical categories such as tense, mood, voice, aspect, person, number, gender and case."
   e.g. move, moved, moving will have "move" as their lemma form.

3. **Tokenization:** It is a process of breaking up a stream of character or text into meaningful words called "tokens".

4. **Co-referencing:** It tells you which "object" are certain words, generally pronouns, in the sentence are pointing to.
   e.g. In sentence "If Dogers win against Pennant then they win the series", co-referencing will tell us that they refers to "Dogers".

3.3 Tools

In this section we describe the various tools that we have used in our technique.

1. **MSR SPLAT:** MSR SPLAT [17] is a language analysis toolkit. It is a web service that MSR has provided. It provides a lot of Natural Language Processing functionalities like:

   - Part of Speech Tagging
   - Dependency tree
   - Constituency tree
   - Lemmatization
   - Tokenization

- Chunking
- Parse tree
- Sentiment Analysis, etc.

We use MSR Splat to obtain the Part of Speech Tag of the words.

2. **StanfordCoreNLP:** It is a NLP engine developed at Stanford [18] which provides state of the art tools. Lemmatization, Deterministic Co-referencing, POS Tagging, etc. are some of the features provided by this NLP suite. We use StanfordCoreNLP for Lemmatization of words and Co-referencing. Lemmatization helps us get the base forms of the words so that the matching is easy and co-referencing helps us resolve the prepositions.

3. **Humanizer:** Humanizer is a Natural Processing Tool which gives singular and plural forms of a word. You can specify whether you want singular form or a plural form of a word and the tool returns the same. It has a decent accuracy but we found some cases during our usage where the tool gave wrong results.

3.4 Implementation Backbone

In this section we describe some fundamental details about our system, before we delve in the main algorithm. Functionalities in this section work as a backbone for our system and support the main algorithm.

1. **Representing a First Order Logic Term:** We used Context Free Grammar$\big($CFG$\big)$ to formalise for computer to understand FOL terms. This was the first challenge that we faced, we needed to design a representation for the First Order Logic that could be understood by our system.
   For this we designed a simple Context Free Grammar$\big($CFG$\big)$ which could represent the First Order Logic formulas in our system.

CFG used for representation: We use the following grammar within our implementation to represent the First Order Logic Term. It is powerful and expressive enough to capture the semantics that we need currently in our system.

BinaryOp	→	OR
	→	AND
	→	IMPLIES
	→	BIMPLIES
UnaryOp	→	NEG
Quantifier	→	ALL
	→	EXISTS
Predicate	→	Name Args
Term	→	Predicate
	→	BinaryOp Term Term
	→	UnaryOp Term
	→	Quantifier Term

Table 3.6: Grammar for Representing First Order Logic

Now we define the terms used in the CFG,

Definition 3.4.1. *Predicate* It is the basic unit in FOL, representing True or False.
e.g. $son(x, y)$: x is son of y
$color(x)$: color is x
$bright(x)$: x is bright

Definition 3.4.2. *BinaryOp* It stands for the Binary Operator in the FOL system. These operators require two Terms to operate on. Binary Operators are OR, AND, IMPLIES, BI-IMPLIES.

e.g. $color(Red) \vee color(BLUE)$: color is Red or color is Blue

e.g. $color(Red) \wedge bright(RED)$: color is Red and Red is bright

Definition 3.4.3. *UnaryOp* It stands for Unary Operator in the FOL system. These operators require only one Terms to operate on. Unary Operator in our system is only NEG, representation for Negation.

e.g. $\neg color(Red)$: color is not Red

Definition 3.4.4. *Term* It has a recursive definition.

Term can either be

(a) a Predicate

e.g. $color(Red)$: Red is a color

(b) a Quantifier applied to Term

e.g. $\exists\big(x\big)color(x)$: there exists a color

(c) a combination of Terms using the Binary or Unary Operators.

e.g. $\exists\big(x\big)color(x) \wedge bright(x)$: there is a color which is bright

2. **Data Structures:** We define the data structures defined in our system here. These data structures represent a class in our system and have data members and functions that operate on them. We describe them in detail below.

   (a) **Problem:** It is read from the benchmark and it has following data member in our system.

   - Problem.Id: Every problem has an Id. This field represents the identification number given to it.
   - Problem.English: This field has the Natural Language sentence of the problem.
   - Problem.Predicates: This field contains the predicates for the given problem.
   - Problem.Logic: This field has the annotated First Order Logic translation for the corresponding Problem.English.

(b) **Rule:** Rules is our system have four components:

- Rule.English: This field has the Natural Language template that should match with the given problem.English. It is combination of word fragments and POS Tag fragments.
- Rule.Predicates: This field contains the predicates for the given Rule. I
- Rule.Logic: This field has the template for the First Order Logic translation for the corresponding Rule.English.
- Rule.Constraint: This field has the constraint which should hold for the rule to be applied on the given problem.

(c) **Predicate:**

- Predicate.symbol: This field has the name of the predicate and the argument for it.
- Predicate.Definition: This field has the definition for the predicate.

3. **Utility Functions:**

- Regex.Match: This function is used to match the two given strings using the Regular Expressions. We have used Regular Expression in our system as it gives us quite a few advantages for expressibility in the Rules. Some of the features that we use in the rules are:
  - **OR (|) :**This gives us the ability to match multiple choices for a single options in a rule. In order to give more perspective, we show an example, Rule.English: **(a | an | the)** (NN) is (JJ)

    The first word fragment in this rule has three options which could lead to a possible match i.e. the sentence could start with a or an or the and still would match with our rule. This is the power that regular expression gives us in writing rules.
  - **Optional (?) :**This feature lets us keep some of the word fragments in the rule as optional. We could say in a rule that a particular word in the rule could be present or not. To give more perspective on that, we show an

example,

Rule.English: (a|an|the) (NN) is **(the)?** (JJ)

The fourth fragment in this rule is marked as optional. Regular Expression give us this feature that we can mark fragments as optional using "**?**".

- Regex.Matches: It is a standard function defined in the Regular Expression library of F#. It searches an input string for all occurrences of a regular expression and returns all the matches.
- searchPredicates: We have built this function to match the given predicates in the sentence. This matching problem is not simple because we could have singular, plural variations. We could also have some words in predicate.Definition which are not in the problem.English. We built following algorithm to search for predicates in the sentence.

```
Data: problem
Result: List Pred_out
sentence ←problem.English
for pred in problem.Predicates do
    defn ←pred.Definition
    if Regex.Match(sentence, defn) then
        Pred_out ←pred
    end
    else
        lemmaSentence ←getLemma(sentence)
        lemmaDefn ←getLemma(defn)
        if Regex.Match(lemmaSentence, lemmaDefn) then
            Pred_out ←pred
        end
        else
            singularSentence ←getSingular(sentence)
            singularDefn ←getSingular(defn)
            if Regex.Match(singularSentence, singularDefn) then
                Pred_out ←pred
            end
            else
                pluralSentence ←getPlural(sentence)
                pluralDefn ←getPlural(defn)
                if Regex.Match(pluralSentence, pluralDefn) then
                    Pred_out ←pred
                end
            end
        end
    end
end
```

Algorithm 1: Search Predicates in Sentence

4. isConstraintSatisfied: This function checks for the validity of the constraint of the given predicate. Input to the function is the list of words captured from the sentence(captureW) and the captured list of word from the predicate(captureD).
   Constraint in our system is defined as Constraint(1, EXACT, 1,1, PLURAL)
   First argument in this tells the index of the word in captureW.
   Second argument tells the form in which the word has to be matched.
   Third argument is the index of the predicate that we would be matching.
   Fourth argument is the index of the word in the predicate specified in third argument.
   Fifth argument tells the form in which the word has to be matched.
   e.g. captureW=men ; captureD=man
   Then we match EXACT(men) with PLURAL(man).

5. **Splat Integration:** Splat is available as a web service. We integrated it with our system to get the Part of Speech Tags for the words.

   Data: String S_{INP} for which POS Tags are required
   Result: List P_{out}, each entry has POS Tag corresponding to the word in S_{INP}
   1 msrSplat ← new SplatServiceClient() ;
   `/* call the Analyze function of Splat Service Client */`
   2 P_{out} ← msrSplat.Analzye(S_{INP},"POS_Tags")

   Algorithm 2: Getting POS Tags for a sentence from MSR Splat

6. **Regression/Test Suite:** We have built a regression functionality in our framework. Regression is necessary for such a system as when the count of the rules goes up it becomes very difficult to keep track of:

   - How many problems have been solved by the new rule?
   - Was there any problem that gets unsolved due to current changes?

   We maintain the details in an excel sheet. Excel sheet has following columns:

   - **Natural Language Sentence:** This is the Id for the Regression suite. We either generate the FOL for this sentence or we take the FOL of this sentence from the benchmark and generate the English for it.

- **First Order Logic Translation:** This column holds the result that we got by applying our algorithm for Natural Language to First Order Logic.
- **Solved Ever E2L:** If the natural language sentence was ever translated to correct logic then this column is set as 1.
- **Current Run Solved E2L:** If the natural language sentence was correctly translated to first order logic in the current run of the algorithm then this column is 1.
- **Natural Language Translation:** This column holds the result we get by our algorithm for First Order Logic to Natural Language.
- **Solved Ever L2E:** If the logical form was ever translated to correct natural language sentence then this column is set as 1.
- **Current Run Solved L2E:** If the logical form was correctly translated to natural language sentence in the current run of the algorithm then this column is 1.

Chapter 4

Technical Details

4.1 Benchmark

One of the contribution of this Thesis has been the development of benchmarks for this problem. We have collected around 350 sentences which we have annotated with correct First Order Logic (to the best of our knowledge). We have also listed the predicates used in the sentences.
There are around 350 sentences in the benchmarks and for each sentence we have following information:

1. Source for where it was obtained
2. Natural Language Sentence
3. Predicates used in the FOL translation
4. Translation in First Order Logic

4.2 Top Down Approach

In this section we describe the technique that we implemented to solve the given problem of translating Natural Language sentence to its corresponding First Order Logic and translating given First Order Logic formula to its corresponding Natural Language sentence.

In our technique we targeted the full structure of the sentence. We tried to achieve English to Logic translation and Logic to English translation with the same set of rules.
In order to help the student in learning FOL Translation, it is very essential to help them if their translation is wrong. To solve this problem we had an idea that if we are able to show students what their "wrong translation" actually means, then it would be very helpful for them to understand where they are going wrong. This was one of the "aha"! moment for us. Though this didn't worked out as we would have wanted. Initially we thought that it would be easier to get back the Natural Language meanings of the FOL formulas, if we tied the NL sentence and the FOL in the rule itself. We had a hypothesis that since we have bound our domain to the graduate level problems, we might have multiple problems of the same structure. So we based our rules on the entire structure of the sentence. Rules had a template of the Natural Language sentence that would match the structure of the sentence and map it to its corresponding FOL formula. So there were two things that we focused on:

1. Build a framework where we can easily Add/Edit Rules in a specification file and the algorithm implementation remains unchanged.

2. Having same set of Rules which helps us in translating NL to FOL and vice versa.

4.2.1 Overall Idea

Our main idea was to have a set of Rules, which help us in translating both ways. We had a Matcher and a Applier for each case. Utility functions used have been defined in the Utility section.

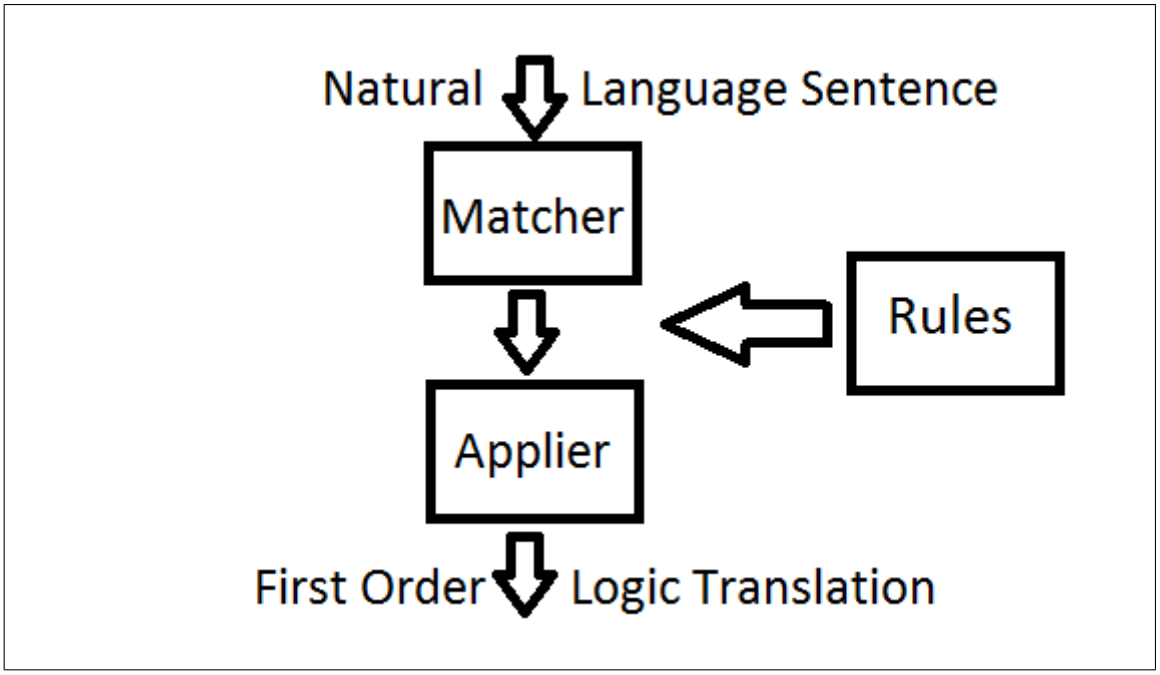

Figure 4.1: Overall Idea

We use the same set of Rule for both way translation. In figure 4.1 we can have input to matcher as NL sentence and we will get output as FOL Translation. If we give input as FOL then we have output as NL using the same set of Rules. Point to note, Matcher and Applier are different in both cases.

Now let us define the role of each component: We will talk about the roles of each component in context of Natural Language to FOL Translation.

1. **Rules:** We have a set of rules in our system. These rules have a bi-directional mapping between the template of Natural Language, template of Predicates and the corresponding template FOL. Rules are used by the matcher and applier to generate the FOL for a given NL or to generate the NL sentence for a given FOL.

   Rules have four components, an example of such a rule would be:

   - **Rule.English:** all (NN) are (NN|JJ)
   - **Rule.Predicates:** p(x):(NN) ; q(x):(NN|JJ)
   - **Rule.Logic:** $\forall(x)P1(x) \rightarrow P2(x)$
   - **Rule.Constraint:** Constraint(1, EXACT, 1,1, PLURAL); Constraint(2, EXACT,2,1, EXACT)

2. **Matcher:** Role of Matcher is to match the template English of the Rule with the given Natural Language sentence. If the template from the rule matches then we extract certain information using utility function Capture which captures the required information and passes it to the Applier.

3. **Applier:** Role of Applier is to use the information received from the Matcher and place it accordingly in the FOL Term as defined in the FOL template for the corresponding Rule. The template of FOL defined in the Rule have placeholders in them. Applier has to walk-through the entire template and replace the placeholders with the values returned by the Matcher.

Let us see few examples to demonstrate, we use the rule defined above:
Input.English: All monkey are mammals
Input.Predicates: monkey(x): x is a monkey; mammal(x): x is a mammal
Matcher:

1. Convert the input and rule to internal representation using 1
   Input.English: all/DT monkey/NN are/VB mammals/NN
   Rule.English: all/DT \w+/NN are/VB \w+/NN|JJ

2. If Input.English and Rule.English match then we proceed to convert the predicates,
   Input.Predicates: monkey(x):monkey/NN; mammal(x):mammal/NN
   Rule.Predicates: p(x):\w+/NN ; q(x):\w+/(NN|JJ)

3. Match the predicates,
   To match the predicates we match for two things in Predicates:

   (a) **Arguments:** Number of arguments in the Rule.Predicate and Input.Predicate should be same.

   (b) **Definition:** Regex Match of the Definitions should be the same.
   p(x) matches with monkey and q(x) matches with mammal(x)

   Applier: If Matcher returns true for a combination of Input sentence and Rule, then Applier:

(a) Applies the Rule i.e. it outputs the FOL for the corresponding Rule. FOL in the Rules have placeholders for the holes in the Term. Applier fills these holes corresponding to the match we get from the Matcher.

(b) Compares the generated FOL with the gold standard FOL in the Benchmarks. Since the internal representation of FOL is in a form of a Tree, we expand both the Terms and compare the nodes.

Output: $\forall(x)\text{monkey}(x) \rightarrow\text{mammal}(x)$

Steps mentioned above illustrate with the help of an example how we do NL to FOL translation. Example above gives us a high level picture about our technique. We would now go in detail and explain each component. We would be as comprehensive as we can be in our attempt and explain in details about each algorithm that we have in our framework.

4.2.2 Algorithms and Implementation

In this section we describe the algorithms that we use to implement the overall idea in our system. We have taken great care to implement a system which acts as a framework, where we can implement different algorithms for the given problem of translation. The implemented system also has a Regression framework, where we can test the effectiveness of our translation algorithm against the Benchmark that we have created.
Now we describe in detail about each component in our system:

(a) **Rules:** Rules in our system help us to tie together the Natural Template to the First Order Logic Template.

Rules have four components:

- **Rule.English:** It contains the Natural Language Template.
  e.g. some (NN) are (NN|JJ)
- Rule.Predicate: It has the description about the Predicates that are defined over the Rule.English.
  $P(x) : (NN); Q(x) : (NN)$

- **Rule.Constraint:** It ties the sentence fragments of the Rule.English with the Rule.Predicate. It also introduces the concept of Capture. Word fragments that have brackets around them are meant to be captured by the system. It has four component in total.
  - First component tells about the index of the captured word from the Problem.English.
  - Second component tells in what form we need to compare the captured word. It can be either the EXACT form in which the word was captured or it can be PLURAL form in which case we convert the captured word to its plural form.
  - Third component tells about the index of the captured word from the Predicate. It has two sub components, first one tells the index of the Predicate and second one tell the index of the word fragment in the Predicate itself.
    e.g."1,EXACT,1,1,PLURAL ; 2,EXACT,2,1,EXACT"
    This example has two constraints, first one is to compare the exact form of the first captured word from the sentence with the plural form of the first captured word fragment from the first predicate.
    Second constraint is to compare the exact form of the second captured word from the sentence with the exact form of the first captured word fragment from the second predicate.

- **Rule.Logic:** It has the rule template which has holes in it, to be filled by the applier before generating the FOL.
  e.g. $\forall(x)P(x) \rightarrow Q(x)$

We have two types of rules in our system:

- **Recursive:** These rules have sentence fragments as holes and replace the FOL Term of the sentence fragment in the FOL output. An example of such rule in our system would be, rule for “if then” structure,

  - Rule.English: if S1 then S2
  - Rule.Logic: FOL(S1) $\rightarrow$ FOL(S2), this is the representational form of the Logic.

    In the system it is defined as,

    BinaryOp([], IMPLIES, predicateTerm([], "/s1", []), predicateTerm([], "/s2", []))

    S1 and S2 are the sentence fragments. We pass these sentence fragments to the algorithm again to get it solved and get the FOL represented by placeholders "/s1" and "/s2", hence the recursion. FOL(S1) and FOL(S2) represent the First Order Logic Terms corresponding to the S1 and S2.

- **Non Recursive:** These do not call the Rule engine again and directly output the FOL Term based on the Rule. An example of such a Rule would be,
  - Rule.English: (NN) VB (NN)
  - Rule.Predicates: p(x):(NN); q(x,y):VB
  - Rule.Constraint: "1, EXACT, 1,1, PLURAL"
  - Rule.Logic: $\forall(x)P1(x) \rightarrow P2(x, y)$ Representation in the system is done as: BinaryOp(All("x"), IMPLIES, predicateTerm([], "/p1", ["x"]), predicateTerm([], "/p2", ["x";"/w2"])),

    here, "/p1" and "/p2" are placeholders for the predicates in the order defined by the constraints and "/w2" refers to the capture of second word in the sentence fragment. "/w2" will refer to the second Noun is the sentence as there are two captures in the sentence.

(b) **Matcher:** In this section we would go through the details of how the Matcher works. There are two major functionalities that Matcher has to perform:

- Match the correct Rule with the given statement.
- Capture the details mentioned in the Rule and pass them to Applier.

Data: rule, problem

Result: $Capture_P$, $Capture_W$, $Capture_D$, $Capture_S$

```
1  probRegex ←getMatchRegex(problem.English)
2  ruleRegex ←getMatchRegex(rule.English)
3  if Regex.Match(probRegex, ruleRegex) then
4      if rule.recFlag then
5          matchRecRule(rule, problem, probRegex, ruleRegex)
6      end
7      else
8          matches ←Regex.Matches( probRegex, ruleRegex )
9          captureW ←getCaptureW(matches)
10         capturePOS ←getCapturePOS(matches)
11         captureP, captureD ←matchPredicates(problem, rule, captureW,
           capturePOS)
12         return captureW captureP captureD
13     end
14 end
```

Algorithm 3: Matcher Algorithm

Now we define in detail and see how each line manipulates the input in our system. We take the example of a sentence to show the working of our Algorithm. Let that example be,

- Problem.English: If men love apples then women play cricket.
- Problem.Predicate: man(x):x is a man; love(x,y):x love y; woman(x):x is a woman; play(x,y):x plays y

We will go line by line for the Algorithm defined above:

1. getMatchRegex(problem.English)

   Here we pass the English statement for which we need the translation and get the POS tags in the in-house representation(inspired from Stanford parser representa-

tion) that lets us do Regex matching. e.g.,

If/IN men/NNS love/VBP apples/NNS then/RB women/NNS play/VBP cards/NNS all/\w+ \w+/NN are/\w+ \w+/NN

2. getMatchRegex(rule.English)

   Here we pass the rule.english which we are trying to match and get the POS tags in the in-house representation.e.g.,

   îf/\w+ (.+?) then/\w+ (.+?)$

3. Regex.Match(probRegex, ruleRegex)

   We use it to match the natural language statement with the english part of the rule. This function has been defined in the utility function section 3.

4. rule.recFlag

   It is a data member of Rule class. It is set to true if the rule is recursive. Here we check if the rule is recursive or not.

5. call matchRecRule()

6. end of if

7. else, the rule is non recursive

8. matchRecRule(rule, problem, probRegex, ruleRegex)

   Call the function which handles the recursive sentences.

9. matches ←Regex.Matches(probRegex, ruleRegex)

   This function has been defined in the utility function section 3.If input is men love apples then matches would be men/NN love/VB apples/NN.

10. captureW ←getCaptureW(matches)

    Here we get the words that are captured from the sentence. For our example captureW would be men, love and apples.

11. capturePOS ←getCapturePOS(matches)

    Here we get the POS tags of the words that are being captured. For our example capturePOS would be NN, VB and NN.

12. captureP, captureD ←matchPredicates(problem, rule, captureW, capturePOS)
    Now we pass the information captured till now to matchPredicates, defined in algorithm 4, which would now match the predicates.
13. end of else

4.2.3 Match Predicates

Another major aspect of Matcher is to match the predicates of the given problem with the predicates of the rule in our system. We call this algorithm “matchPredicates”. In this algorithm we first check if the predicates in the sentence and rule match and then

verify the constraint specified.

```
Data: problem, rule, captureW, capturePOS
Result: Capture_P, Capture_D
for predProb in problem.Predicates do
    while predRule in rule.Predicates do
        if comparePredicate(predProb, predRule ) then
            probRegex ←getMatchRegex (predProb.Definition)
            ruleRegex ←getMatchRegex (predRule.Definition)
            matches ←Regex.Matches( probRegex, ruleRegex )
            captureDef ←getCaptureW(matches)
            capturePOS ←getCapturePOS(matches)
            if isConstraintSatisfied(rule.Constraint) then
                Capture_D ←captureDef
                Capture_P ←predProb
                rule.Predicates.remove(predRule)
                break
            end
        end
    end
end
return Capture_P   Capture_D
```

Algorithm 4: Match Predicates

We will now run this algorithm on our example.

Input:

problem = Men love apples

rule = (NN) VB (NN)

captureW = men, apples

capturePOS = NN, VB

1. **for** predProb in problem.Predicates

We try to find the match for all the predicates in the given problem.

problem.Predicates: man(x):x is a man; love(x,y):x loves y

2. **while** predRule in rule.Predicates

   We try to match any possible match of the rule predicates with the problem predicates.

   rule.Predicates: p(x):(NN); q(x,y):VB

3. **if** comparePredicate(predProb, predRule)

   This function checks if both the predicates have same structure. Structure in the case of predicates is the number of arguments.

   e.g. man(x) will match with p(x), since they have same number of arguments.

   Similarly q(x,y) will match with love(x,y).

4. probRegex ←getMatchRegex (predProb.Definition)

   Here we pass the definition of the problem predicate which we are trying to match and get the POS tags in the in-house representation.e.g.,

   man/NN , love/VB

5. ruleRegex ←getMatchRegex (predRule.Definition)

   Here we pass the definition of the rule predicate which we are trying to match and get the POS tags in the in-house representation.e.g.,

   \w+/NN , \w+/VB

6. matches ←Regex.Matches(probRegex, ruleRegex)

   This function has been defined in the utility function section 3.Matches for above defined probRegex and ruleRegex would be man/NN , loves/VB

7. captureDef ←getCaptureW(matches)

   Here we capture the words from the matches which represent the definition of the rule.

   In example that would be man and loves

8. capturePOS ←getCapturePOS(matches)

   Here we capture the POS tags of the word matched in the matches.

9. **if** isConstraintSatisfied(rule.Constraint, captureW, captureDef)

   This utility function is defined in section 3. If the constraint is satisfied then we have matched the predicate.

10. $Capture_D$ ←captureDef

    We add the captured definition to the final list of capture definition.

11. $Capture_P$ ←predProb

    We add the predicate to the list of captured predicates.

12. rule.Predicates.remove(predRule)

    Here we remove the matched predicate from the list of rule predicates since we would not want it to match with some other predicate.

13. break

    Here we break as we have matched the predicate and would not like to match the predProb with any other predRule.

4.2.4 Match Recursive Rules

We need to handle the matching of recursive rules differently. First we need to match the sentence fragments in place of the word fragments. Then we need to call Matcher Algorithm on the sentence fragments captured and combine the Terms returned according to the recursive rule and pass it to the applier. Now we define the

exact algorithm.

Data: rule, problem, probRegex, ruleRegex

Result: $Capture_P$, $Capture_W$, $Capture_D$, $Capture_S$

```
matches ←Regex.Matches( probRegex, ruleRegex )
captureW ←getCaptureW(matches)
for capture in captureW do
    wordL, posL ←extractEngPOS capture
    phrase ←String.concat " " wordL
    newProblem ←problem.copy(phrase)
    for rule in rules do
        Capture_P, Capture_W, Capture_D ←matcher(rule, newProblem)
        if Capture_P then
            term ←apply(rule.FOL, Capture_P, Capture_W, Capture_D)
            Capture_S ←term
            break
        end
    end
end
return Capture_P Capture_W Capture_D Capture_S
```

Algorithm 5: Match Recursive Rules

We will now run this algorithm on our example.

Input:

Problem = if men love apples then women play cards

Rule = if S1 then S2

probRegex = If/IN men/NNS love/VBP apples/NNS then/RB women/NNS play/VBP cards/NNS

ruleRegex = îf/\w+ (.+?) then/\w+ (.+?)$

1. matches ←Regex.Matches(probRegex, ruleRegex)

   This function has been defined in the utility function section 3.Matches for above

defined probRegex and ruleRegex would be men/NNS love/VBP apples/NNS , women/NNS play/VBP cards/NNS

2. captureW ←getCaptureW(matches)

   Here we extract the matches into captureW

3. **for** capture in captureW

   We iterate over the captured sentence fragments

4. wordL, posL ←extractEngPOS capture

   Here we extract the words and POS into two different lists.

   wordL = {men, love, apples}

   posL = {NNS, VBP, NNS}

5. phrase ←String.concat " " wordL

   We construct the new statement by concatenating all the words from the wordL.

   phrase = men love apples

6. newProblem ←problem.copy(phrase)

   Once we have the new statement we replace it in the given problem. So, the predicate list remains the same and we make a copy of the given problem by replacing the problem.English with phrase.

7. **for** rule in rules

   Now we try to solve the new problem by matching all the rules with it.

8. $Capture_P$, $Capture_W$, $Capture_D$ ←matcher(rule, newProblem)

   Here we call the matcher algorithm, defined in algorithm 3 with the required arguments.

9. if $Capture_P$

   If the rule matching was successful then we will have some predicates captured.

10. term ←apply(rule.FOL, $Capture_P$, $Capture_W$, $Capture_D$)

    Call the apply function to get the First Order Logic of the new problem that we extracted from the original problem.

11. $Captures_S$ ←term

    Add the term to the list of captured FOL of the sentences.

12. break

    Once we have got the FOL of the new term we do not apply more rules and go out of the loop.

(c) **Applier:** Role of applier is to fill in the holes in the given FOL term in the rule. Holes in the rules can be of following form:

- **/p**: It is placeholder for a **predicate** from the problem.
- **/w**: It is a placeholder for **word** from the problem.English.
- **/s**: It is a placeholder for **FOL** of a partial sentence in case of recursive rules.

The FOL Term in our rule is written as follows:
BinaryOp(All("x"), IMPLIES, predicateTerm(NONE, "**/p1**", ["x"]), predicateTerm(NONE, "**/p2**", ["x";"**/w2**"]))
/p1 and /p2 are the placeholder for the predicates in captureP.
/w2 is the placeholder for the word in captureW
BinaryOp(NONE, IMPLIES, predicateTerm(NONE, "**/s1**", []), predicateTerm(NONE, "**/s2**", []))
/s1 and /s2 are the placeholder for the FOL Term in captureS.
Applier recursively opens the FOL Term defined in the rule and then replaces the placeholders with the captured froagments obtained from Matcher.

4.2.5 Main

This is the main function for Natural Language to First Order Logic Translation. It calls the Matcher function and passes the data received from it to Applier. Applier returns the First Order Logic for the respective problem that we attempted to solve.

```
Data: Problems
Result: FOL
for problem in Problems do
    for rule in Rules do
        FOL ←Applier(Matcher(rule, problem) )
        if FOL.IsSome then
            print FOL
        end
    end
end
```

Algorithm 6: Matcher Algorithm

4.2.6 First Order Logic To Natural Language

In order to go from First Order Logic to Natural Language the idea was to compare the skeleton First Order Logic that we have in our Rules with the FOL that we have to convert to Natural Language. Match is only considered when the structure of the predicates also match. Once we have the match we fill in the holes that are there in the natural language skeleton of the rule.

This was the main idea for going from First Order Logic to Natural Language. We could not invest much in this technique due to lack of time.

Chapter 5

Results and Observations

5.1 Experimental Setup

Implementation of the algorithm defined in the previous chapter is done in F#. We used Visual Studio 2013 as the Integrated Development Environment (IDE). We used MSR Splat web service and Stanford Online parser. Benchmarks are stored in an Excel sheet and we use Csv Reader for reading the file. Results are also stored in an Excel Sheet.

5.2 Results

In this section we will describe the results that we are getting from our system. We will also talk about the observations that we have made in the course of solving this problem. We currently have around 195 rules in our system and we get translation of around 215 sentences using those rules. Translation achieved by our system are given below.

Table 5.1: Results

Natural Language	First Order Logic Translation
All men are smart mortals	$\forall$(x) ((man(x)) $\rightarrow$ (mortal(x)))
Every gardener likes the sun	$\forall$(x) ((gardener(x)) $\rightarrow$ (likes(x, sun)))
No purple mushroom is poisonous	($\neg$ $\exists$(x) ((mushroom(x)) $\wedge$ ((purple(x)) $\wedge$ (poisonous(x)))))
Pluto loves its master	$\exists$(x) ((master(Pluto, x)) $\wedge$ (loves(x, Pluto)))
Continued on next page	

Table 5.1 – continued from previous page

Natural Language	First Order Logic Translation
All humans eat some food	∀(x) ((human(x)) → ∃(y) ((eat(y)) ∧ (food(x, y))))
Pen is mightier than sword	(mightier(Pen, sword))
Julie feeds all her dogs	∀(x) ((feeds(x, Julie)) → (dog(Julie, x)))
All men are smart mortals	∀(x) ((manu(x)) → (mortalu(x)))
Every gardener likes the sun	∀(x) ((gardener(x)) → (likes(x, sun)))
Pluto loves its master	∃(x) ((master(Pluto, x)) ∧ (loves(x, Pluto)))
All humans eat some food	∀(x) ((human(x)) → ∃(y) ((eat(y)) ∧ (food(x, y))))
Pen is mightier than sword	(mightier(Pen, sword))
Julie feeds all her dogs	∀(x) ((feeds(x, Julie)) → (dog(Julie, x)))
John looks for a unicorn	∃(x) ((unicorn(x)) ∧ (looks(John, x)))
There is a barber who shaves every man	∃(x) ((barber(x)) ∧ ∀(y) ((man(y)) → (shaves(x, y))))
Not every person who plays football knows cricket	(¬ ∀(x) (((person(x)) ∧ (plays(x, football))) → (knows(x, cricket))))
Some students are both intelligent and hard workers	∃(x) ((student(x)) ∧ ((intelligent(x)) ∧ (hardWorker(x))))
Not every famous actor is talented	(¬ ∀(x) (((actor(x)) ∧ (famous(x))) → (talented(x))))
Every executive has a secretary	∀(x) ((executive(x)) → (secretary(x)))
A dead man tells no tales	∃(x) (((dead(x)) ∧ (man(x))) ∧ (¬ ∃(y) ((tale(y)) ∧ (tell(x, y)))))
Every flower has a fragrance	∀(x) ((flower(x)) → (has(x, fragrance)))
The head which wears the crown lies uneasy	∀(x) (((head(x)) ∧ ∃(y) ((crown(y)) ∧ (wears(x, y)))) → (uneasy(x)))
Monkeys are mammals	∀(x) ((monkey(x)) → (mammal(x)))
No person donates to every charity	(¬ ∃(x) ((person(x)) ∧ ∀(y) ((charity(y)) → (donates(x, y)))))
All flowers are not fragrant	(¬ ∀(x) ((flower(x)) → (fragrant(x))))
If something is damaged then the curator will be blamed	(∃(x) (damaged(x)) → (blamed(curator)))
Nelson can drive every car in the lot	∀(x) (((car(x)) ∧ (inLot(x))) → (canDrive(Nelson, x)))
James is a friend of Ellen or Connie	((friend(James, Ellen)) ∨ (friend(James, Connie)))
Every person can sell some apples	∀(x) ((person(x)) → ∃(y) ((apple(y)) ∧ (sell(x, y))))
Nelson teaches a few morons	∃(x) ((moron(x)) ∧ (teaches(Nelson, x)))
No person can sell every product	(¬ ∃(x) ((person(x)) ∧ ∀(y) ((product(y)) → (sell(x, y)))))
Some people can sell any product	∃(x) ((people(x)) ∧ ∀(y) ((product(y)) → (sell(x, y))))
Bromine is extractable from seawater	(ExtractableFrom(Bromine, seawater))

Continued on next page

Table 5.1 – continued from previous page

Natural Language	First Order Logic Translation
All purple mushrooms are poisonous	∀(x) (((mushroom(x)) ∧ (purple(x))) → (poisonous(x)))
No liberal candidate will be appointed or elected	(¬ ∃(x) (((liberal(x)) ∧ (candidate(x))) ∧ ((appointed(x)) ∨ (elected(x)))))
The early bird catches the worm	∀(x) (((early(x)) ∧ (bird(x))) → (catches(x, worm)))
Steve took the bus or the train	((took(Steve, the)) ∨ (took(Steve, bus)))
Doctors and lawyers are graduates	(∀(x) ((doctor(x)) → (graduate(x))) ∧ ∀(y) ((lawyer(y)) → (graduate(y))))
If Argentina joins alliance then Brazil or Chile boycotts the alliance	((joins(Argentina, alliance)) → ((boycotts(Brazil, alliance)) ∨ (boycotts(Chile, alliance))))
If Amherst wins the first game then Colgate or Dartmouth wins the first game	((winsFirstGame(Amherst)) → ((winsFirstGame(Colgate)) ∨ (winsFirstGame(Dartmouth))))
If Aristedes is corruptible then everyone is corruptible	((corruptible(Aristedes)) → ∀(x) (corruptible(x)))
Either taxes are increased or if expenditure rises then the debt rises	((increased(taxes)) ∨ ((rises(expenditure)) → (rises(debt))))
Charlie is neat and Charlie is sweet	((neat(Charlie)) ∧ (sweet(Charlie)))
All fruits and vegetables provide nourishment	∀(x) (((fruit(x)) ∨ (vegetable(x))) → (provnourishment(x)))
Chicago is smaller than New York	(smaller(Chicago, New York))
Saul and Jonathan were lovely and pleasant	(((lovely(Saul)) ∧ (pleasant(Saul))) ∧ ((lovely(Jonathan)) ∧ (pleasant(Jonathan))))
If I am president then I am famous	((president(I)) → (famous(I)))
All men who survived were honoured	∀(x) (((man(x)) ∧ (survived(x))) → (honoured(x)))
All men who play cricket know football	∀(x) (((man(x)) ∧ (plays(x, cricket))) → (knows(x, football)))
All men and boys play cricket and know football	∀(x) (((man(x)) ∨ (boy(x))) → ((plays(x, cricket)) ∧ (knows(x, football))))
If I read then I make good grades	((read(I)) → ∀(x) (((good(x)) ∧ (make(I, x))) → (grade(x))))
A student who takes geometry also takes calculus	∀(x) (((student(x)) ∧ (takes(x, geometry))) → (takes(x, calculus)))
All friends of Ali are friends of Bill	∀(x) ((friend(x, Ali)) → (friend(x, Bill)))
Any person who supports Ickes will vote for Jones	∀(x) ((supports(x, Ickes)) → (votes(x, Jones)))
Every person who draws circles draws figures	∀(x) (((person(x)) ∧ (draws(x, circles))) → (draws(x, figures)))
There is a man who has no son	∃(x) ((man(x)) ∧ (¬ ∃(y) (son(x, y))))

Continued on next page

Table 5.1 – continued from previous page

Natural Language	First Order Logic Translation
If Ed wins the first_prize then if Fred wins the second_prize then George is disappointed	((wins(Ed, first_prize)) → ((wins(Fred, second_prize)) → (disapointed(George))))
No man who is a candidate will be defeated if he is a campaigner	∀(x) ((((man(x)) ∧ (candidate(x))) ∧ (campaigner(x))) → (¬ (defeated(x))))
Any positive number has a square root	∀(x) ((positive(x)) → (squareroot(x)))
There is a country that borders Iraq and Pakistan	∃(x) ((country(x)) ∧ ((borders(x, Iraq)) ∧ (borders(x, Pakistan))))
Every person loves his mother	∀(x) ((person(x)) → ∃(y) ((mother(y, x)) ∧ (loves(x, y))))
Joe is an actor but he holds a job	((actor(Joe)) ∧ ∃(x) ((holds(x)) ∧ (job(Joe, x))))
Emily has a dog which loves her	∃(x) (((dog(x)) ∧ (has(Emily, x))) ∧ (loves(x, Emily)))
All car buyers are gullible	∀(x) ((carbuyer(x)) → (gullible(x)))
A number can be negative or positive or zero	∀(x) ((number(x)) → ((negative(x)) ∨ ((positive(x)) ∨ (zero(x)))))
If tap dancing is foolish then I want to be a fool	((foolish(tap dancing)) → ∃(x) ((fool(x)) ∧ (wannabe(I, x))))
If you do not love yourself then you can not love everybody	((¬ (love(you, yourself))) → (¬ ∀(x) (love(you, x))))
There is a dog which does not love all humans	∃(x) ((dog(x)) ∧ (¬ ∀(y) ((human(y)) → (loves(x, y)))))
All good events end well	∀(x) (((event(x)) ∧ (good(x))) → (endwell(x)))
Everyone who loves an animal has a friend	∀(x) (∃(y) ((animal(y)) ∧ (loves(x, y))) → ∃(z) (friend(x, z)))
Every Indian city is overpopulated	∀(x) (((city(x)) ∧ (Indian(x))) → (overpopulated(x)))
You can fool some men all times	∃(x) ((canfool(x)) ∧ ∀(z) (man(You, x, z)))
The empty set has no elements adjoined to it	∀(x) ((emptyset(x)) → (¬ ∃(y) ((element(y)) ∧ (adjoin(y, x)))))
John and Robert can fool Harry	((fool(John, Harry)) ∧ (fool(Robert, Harry)))
The book was signed by every guest	∀(x) ((guest(x)) → (signed(book, x)))
Wolfhounds and terriers are hunting dogs	(∀(x) ((wolfhound(x)) → (huntingDog(x))) ∧ ∀(y) ((terrier(y)) → (huntingDog(y))))
The fastest person is a Scandinavian	∃(x) ((fastest(x)) ∧ ((person(x)) ∧ (Scandinavian(x))))
Every person who is not a scandinavian can be outrun by someone	∀(x) (((person(x)) ∧ (¬ (scandinavian(x)))) → ∃(y) (outrun(y, x)))
Any fish can swim faster than any smaller fish	∀(x) ((fish(x)) → ∀(y) (((fish(y)) ∧ (smaller(y, x))) → (swimfaster(x, y))))

Continued on next page

Table 5.1 – continued from previous page

Natural Language	First Order Logic Translation
Pearson is taller than Jim brother	∃(x) ((brother(x, Jim)) ∧ (taller(Pearson, x)))
For every integer there is an integer which is greater than it	∀(x) ((integer(x)) → ∃(y) ((integer(y)) ∧ (greater(y, x))))
Not all real numbers are rational numbers	(¬ ∀(x) ((realNumber(x)) → (rationalNumber(x))))
Every natural number is either odd or divisible by two	∀(x) ((natNum(x)) → ((odd(x)) ∨ (div2(x))))
Some fierce creatures do not consume coffee	∃(x) ((fierce(x)) ∧ ((creature(x)) ∧ (¬ (consume(x, coffee)))))
There is a man who has never seen a computer	∃(x) ((man(x)) ∧ (¬ ∃(y) ((computer(y)) ∧ (seen(x, y)))))
All watches which are sold by Omega are made in Switzerland	∀(x) (((watch(x)) ∧ (sold(x, Omega))) → (made(x, Switzerland)))
Any person who commits a burglary is unfortunate	∀(x) (((person(x)) ∧ ∃(y) ((burglary(y)) ∧ (commits(x, y)))) → (unfortunate(x)))
If you enroll in the course and prepare hard then you will pass the course	(((enroll(you, course)) ∧ (preparehard(you))) → (pass(you, course)))
If he enters the primary then if he canvasses vigorously then he wins the nomination	((enters(he, primary)) → ((canvasses(he)) → (wins(he, nomination))))
If Argentina or Peru joins the alliance then Chile or Brazil foregoes the alliance	(((join(Argentina, alliance)) ∨ (join(Peru, alliance))) → ((forego(Chile, alliance)) ∨ (forego(Brazil, alliance))))
If we go to Europe then we tour Scandinavia	((go(we, Europe)) → (tour(we, Scandinavia)))
Not every object that glitters is gold	(¬ ∀(x) (((object(x)) ∧ (glitters(x))) → (gold(x))))
No automobile that is very old will be repaired unless it is severely damaged	∀(x) ((((automibile(x)) ∧ (VeryOld(x))) ∧ (WillBeRepaired(x))) → (SeverelyDamaged(x)))
Not every person who talks a great deal has a great deal to say	(¬ ∀(x) (((person(x)) ∧ (talksGreatDeal(x))) → (GreatDealtoSay(x))))
There are no uniforms that are not washable	(¬ ∃(x) ((washable(x)) ∧ (¬ (uniform(x)))))
A communist is a fool or a knave	∀(x) ((communist(x)) → ((fool(x)) ∨ (knave(x))))
All butlers and all valets are well behaved	∀(x) (((butler(x)) ∨ (valet(x))) → (wellBehaved(x)))
All houses which are built of brick are warm and cozy	∀(x) (((house(x)) ∧ (builtOfbrick(x))) → ((warm(x)) ∧ (cozy(x))))
If a bee is angry or frightened then it stings	∀(x) (((bee(x)) ∧ ((angry(x)) ∨ (frightened(x)))) → (stings(x)))
Some authors are successful but not well read	∃(x) (((author(x)) ∧ (successful(x))) ∧ (¬ (wellread(x))))

Continued on next page

Table 5.1 – continued from previous page

Natural Language	First Order Logic Translation
All dogs are domesticated animals	∀(x) ((dog(x)) → (domesticatedanimal(x)))
Domestic animals are gentle and useful	∀(x) (((domestic(x)) ∧ (animal(x))) → ((gentle(x)) ∧ (useful(x))))
Any man who runs for office is a candidate	∀(x) (((man(x)) ∧ (runsFor(x, office))) → (candidate(x)))
Every man who is elected is a good campaigner	∀(x) (((man(x)) ∧ (elected(x))) → ((good(x)) ∧ (campaigner(x))))
If there are some geniuses then every great composer is a genius	(∃(x) (genius(x)) → ∀(x) ((greatComposer(x)) → (genius(x))))
No honest person would conceal his real feelings	(¬ ∃(x) ((honest(x)) ∧ ((person(x)) ∧ (feel(x)))))
Every businessman who is a poet must be a wealthy man	∀(x) (((businessman(x)) ∧ (poet(x))) → (wealthyMan(x)))
No uranium isotope which is radioactive has a very short life	(¬ ∃(x) ((uraniumIsotope(x)) ∧ ((radioactive(x)) ∧ (hasShortLife(x)))))
Everybody respects a person who respects himself	∀(x) ∀(y) (((person(y)) ∧ (respects(y, y))) → (respects(x, y)))
The professor of Greek at Stanford is very learned	∃(x) ((professor(x, Greek)) ∧ ((veryLearned(x)) ∧ (atStanford(x))))
All states in New England are primarily industrial	∀(x) (((state(x)) ∧ (inNewEngland(x))) → (primarilyIndustrial(x)))
The youngest member in our team has climbed Mount Blanc	∃(x) ((youngestMember(x)) ∧ ((inOurTeam(x)) ∧ (hasClimbedMtBlanc(x))))
Every song which lasts an hour is tedious	∀(x) (((song(x)) ∧ (lastsAnHour(x))) → (tedious(x)))
Some things which are meant to give light produce very little light	∃(x) ((thing(x)) ∧ ((meantToGiveLight(x)) ∧ (producesLittleLight(x))))
Railways has never been ill-managed	(¬ (illManaged(Railways)))
No fossil can be crossed in love	(¬ ∃(x) ((fossil(x)) ∧ (canBeCrossedIn(x, love))))
Every logician who eats porkchops for dinner will probably lose money	∀(x) (((logician(x)) ∧ (probablyLose(x, porkchops))) → (eatsForDinner(x, money)))
Umbrellas are useful on a journey	∀(x) ((umbrella(x)) → (usefulOn(x, journey)))
Nothing in the house escaped destruction	(¬ ∃(x) ((inHouse(x)) ∧ (escapedDestruction(x))))
No coat is waterproof unless it has been specially treated	∀(x) ((coat(x)) → ((waterproof(x)) → (treated(x))))
Policemen and firemen are both indispensable and underpaid	∀(x) (((policeman(x)) ∨ (fireman(x))) → ((indispensable(x)) ∧ (underpaid(x))))
Any candidate who is not defeated will be elected	∀(x) (((candidate(x)) ∧ (¬ (defeated(x)))) → (elected(x)))

Continued on next page

Table 5.1 – continued from previous page

Natural Language	First Order Logic Translation
Everyone who would do well in logic ought to study logic	∀(x) ((oughtToStudyLogic(x)) → (doWellInLogic(x)))
Everyone who can not think logically ought to study logic	∀(x) ((¬ (thinkLogically(x))) → (oughtToStudyLogic(x)))
The temperature and air-pressure remained constant	((remainedConstant(temperature)) ∧ (remainedConstant(air-pressure)))
If the prices go up then the housing will be both comfortable and expensive	((goup(prices)) → ((comfortable(housing)) ∧ (expensive(housing))))
If the car runs out of gas then the car stops	((runsOutOfGas(car)) → (stops(car)))
If someone suffers then God is not both benevolent and gracious	(∃(x) (suffers(x)) → (¬ ((benevolent(God)) ∧ (gracious(God)))))
If the housing is not expensive then it will still be plentiful	∀(x) (((housing(x)) ∧ (¬ (expensive(x)))) → (plentiful(x)))
Jones can drive any car in the lot	∀(x) (((car(x)) ∧ (inlot(x))) → (drive(Jones, x)))
The company advertises everything which it produces	∀(x) ((advertisesCompany(x)) → (producesCompany(x)))
There is a man whom everybody does not love	∃(x) ((man(x)) ∧ (¬ ∀(y) (love(y, x))))
All candidates are wealthy liberals	∀(x) ((candidate(x)) → ((wealthy(x)) ∧ (liberal(x))))
If all bananas are yellow then some bananas are ripe	(∀(x) ((banana(x)) → (yellow(x))) → ∃(x) ((banana(x)) ∧ (ripe(x))))
Everyone who is convicted will hang	∀(x) ((convicted(x)) → (hang(x)))
If there is some liberal then every philosopher is a liberal	(∃(x) (liberal(x)) → ∀(x) ((philosopher(x)) → (liberal(x))))
If some liberal is humanitarian then all philosophers are humanitarians	(∃(x) ((liberal(x)) ∧ (humanitarian(x))) → ∀(x) ((philosopher(x)) → (humanitarian(x))))
Anyone who accomplishes something will be envied by everyone	∀(x) (∃(y) (accomplish(x, y)) → ∀(z) (envied(x, z)))
If Harrison is a friend of Kelly then Anderson will not support the Ickes	((friend(Harrison, Kelly)) → (¬ (support(Anderson, Ickes))))
Whoever donates to the UN gives his donations to the UN	∀(x) ((donates(x, UN)) → ∀(y) ((donation(y, x)) → (gives(y, UN))))
Anything which has a tariff paid on it costs the purchaser extra	∀(x) (∃(y) ((tariff(y)) ∧ (paidon(x, y))) → (costsextra(x, purchaser)))
Every thing on the desk is a masterpiece	∀(x) (((thing(x)) ∧ (ondesk(x))) → (masterpiece(x)))
If the directors wanted Smith at the meeting then Smith was invited to the meeting	(∀(x) ((director(x)) → (wantedatmeeting(x, Smith))) → (invitedmeeting(Smith)))
If the investigation continues then new evidence is brought to light	((continues(investigation)) → (broughtto(new evidence, light)))

Continued on next page

Table 5.1 – continued from previous page

Natural Language	First Order Logic Translation
If John has measles then John has a fever	((has(John, measles)) → ∃(x) ((fever(x)) ∧ (has(John, x))))
A greedy fish will chase every shiner	∀(x) (((greedy(x)) ∧ (fish(x))) → ∀(y) ((shiner(y)) → (willchase(x, y))))
Every man who landed on Mars did not return	∀(x) (((man(x)) ∧ (landed(x, Mars))) → (¬ (return(x))))
The reader who is interested in deducing some theorems will find the methods easy	∀(x) (((reader(x)) ∧ ∃(y) ((theorem(y)) ∧ (interestdeduce(x, y)))) → (easymethod(x)))
Any relation which is transitive and irreflexive is asymmetric	∀(x) ((((relation(x)) ∧ (transitive(x))) ∧ (irreflexive(x))) → (asymmetric(x)))
All diligent students are successful	∀(x) (((student(x)) ∧ (diligent(x))) → (successful(x)))
Unpleasant experiences are not eagerly desirable	∀(x) (((unpleasant(x)) ∧ (experience(x))) → (¬ (eagerlydesirable(x))))
Everything which is lawful can be done without scruple	∀(x) ((lawful(x)) → (donewithout(x, without)))
Nobody who is dreaded is asked to prolong his visit	(¬ ∃(x) ((dreaded(x)) ∧ (askprolongvisit(x))))
Iron is less dense than mercury	(lessdense(Iron, mercury))
A constitutional amendment which bans flag-burning should not be adopted	∀(x) (((constAmend(x)) ∧ (bans(x, flagburning))) → (¬ (adopted(x))))
Either music originated from Greece or music originated from Egypt	(((originated(music, Greece)) ∨ (originated(music, Egypt))) ∧ (¬ ((originated(music, Greece)) ∧ (originated(music, Egypt)))))
Man is a social animal	∀(x) ((man(x)) → (socialanimal(x)))
Alfred sings alto while Paul sings bass	((sings(Alfred, alto)) ∧ (sings(Paul, bass)))
Moira was changing plugs and listening to radio	∀(z) ((plug(z)) → ((changing(Moira, z)) ∧ (listening(Moira, radio))))
Every dog that is barking is gullible	∀(x) (((dog(x)) ∧ (barking(x))) → (gullible(x)))
If Barbara practices she will win	((Practices(Barbara)) → (Win(Barbara)))
All grass is green	∀(x) ((Grass(x)) → (Green(x)))
There is a winning combination	∃(x) ((Combination(x)) ∧ (Winning(x)))
Every dog has his day	∀(x) ((Dog(x)) → ∃(y) ((Day(y)) ∧ (Has(x, y))))
Some cat did this	∃(x) ((Cat(x)) ∧ (DidThis(x)))
Everybody loves Paris	∀(x) (Loves(x, Paris))
Every even number is divisible by 2	∀(x) ((Even(x)) → (Div(x, 2)))
Paul knows everything	∀(x) (Knows(Paul, x))
There is no prime number between 23 and 29	(¬ ∃(x) ((Prime(x)) ∧ (Between(x, 23, 29))))
Steve is a computer scientist	(Compscientist(Steve))

Continued on next page

Table 5.1 – continued from previous page

Natural Language	First Order Logic Translation
No integer is both even and odd	(¬ ∃(x) ((Integer(x)) ∧ ((Even(x)) ∧ (Odd(x)))))
John likes anybody who does not like himself	∀(x) ((¬ (Likes(x, x))) → (Likes(John, x)))
There is no person who is not happy	(¬ ∃(x) ((Happy(x)) ∧ (¬ (Person(x)))))
All students are smart	∀(x) ((Student(x)) → (Smart(x)))
There exists a student	∃(x) (Student(x))
There exists a smart student	∃(x) ((Student(x)) ∧ (Smart(x)))
Bill is a student	(Student(Bill))
Every student loves some student	∀(x) ((Student(x)) → ∃(y) ((Student(y)) ∧ (Loves(x, y))))
Bill takes Analysis or Geometry	((Takes(Bill, Analysis)) ∨ (Takes(Bill, Geometry)))
Bill takes Analysis and Geometry	((Takes(Bill, Analysis)) ∧ (Takes(Bill, Geometry)))
Bill does not take Analysis	(¬ (Takes(Bill, Analysis)))
Bill has at least one sister	∃(x) (sister(x, Bill))
Bill has no sister	(¬ ∃(x) (sister(x, Bill)))
Bill has at most one sister	∀(x) ∀(y) (((sister(x, Bill)) ∧ (sister(y, Bill))) → (Equal(x, y)))
Bill has exactly one sister	∃(x) ((sister(x, Bill)) ∧ ∀(y) ((sister(y, Bill)) → (Equal(x, y))))
Bill has at least two sisters	∃(x) ∃(y) (((sister(y, Bill)) ∧ (sister(x, Bill))) ∧ (¬ (Equal(x, y))))
Every student takes at least one course	∀(x) ((student(x)) → ∃(y) ((course(y)) ∧ (takes(x, y))))
Every student who takes Analysis also takes Geometry	∀(x) (((student(x)) ∧ (takes(x, Analysis))) → (takes(x, Geometry)))
No student can fool all the other students	(¬ ∃(x) ((student(x)) ∧ ∀(y) ((student(y)) → (fool(x, y)))))
Neither Claire nor Jenny is in love with Max	((¬ (Love(Claire, Max))) ∧ (¬ (Love(Jenny, Max))))
Jenny will not marry Max unless he is intelligent and in love with her	((¬ ((Love(Max, Jenny)) ∧ (intelligent(Max)))) → (¬ (Marry(Jenny, Max))))
Max is not both intelligent and in love with Jenny	(¬ ((intelligent(Max)) ∧ (Love(Max, Jenny))))
Jenny is Nancy youngest daughter and Claire is her oldest daughter	((youngestdaughter(Jenny, Nancy)) ∧ (oldestdaughter(Claire, Nancy)))
All small cubes are at back of a	∀(x) (((small(x)) ∧ (cube(x))) → (backOf(x, a)))
You can fool some of the people all the time	∃(x) ((people(x)) ∧ ∀(y) ((time(y)) → (fool(x, y))))
You can fool all of the people some of the time	∀(x) ((fool(x)) → ∃(y) ((time(y)) ∧ (people(x, y))))

Continued on next page

Table 5.1 – continued from previous page

Natural Language	First Order Logic Translation
All purple mushrooms are poisonous	∀(x) (((mushroom(x)) ∧ (purple(x))) → (poisonous(x)))
Deb is not tall	(¬ (tall(Deb)))
All bunnies are cute	∀(x) ((Bunny(x)) → (Cute(x)))
Every student who is taking AI is cool	∀(x) (((Student(x)) ∧ (TakingAI(x))) → (Cool(x)))
There is atleast one student who does not hate any of the AI homework	∃(x) ((student(x)) ∧ ∀(y) ((AIhomework(y)) → (¬ (Hates(x, y)))))
Cats rule and dogs drool	(∀(x) ((Cat(x)) → (Rule(x))) ∧ ∀(y) ((Dog(y)) → (Drool(y))))
There is a mushroom that is purple and poisonous	∃(x) ((Mushroom(x)) ∧ ((Purple(x)) ∧ (Poisonous(x))))
There are no mushrooms that are purple and poisonous	∀(x) ((Mushroom(x)) → (¬ ((Purple(x)) ∧ (Poisonous(x)))))
Elder Gods do not like Hello Kitty	∀(x) ((ElderGod(x)) → (¬ (Likes(x, Hello Kitty))))
There is a bunny who is cute	∃(x) ((bunny(x)) ∧ (cute(x)))
There is only one Elvis	∃(x) ((isElvis(x)) ∧ ∀(y) ((isElvis(y)) → (Equal(x, y))))
Every child who owns a Pokemon Card is cool	∀(x) (((child(x)) ∧ ∃(y) ((pokemoncard(y)) ∧ (owns(x, y)))) → (cool(x)))
Someone at Stanford is smart	∃(x) ((at(x, Stanford)) ∧ (smart(x)))
Pluto is a dog	(dog(Pluto))
There is atleast one theif	∃(x) (thief(x))
Some cats are black	∃(x) ((cat(x)) ∧ (black(x)))
Every apple is delicious	∀(x) ((apple(x)) → (delicious(x)))
Peaches are edible unless they are rotten	∀(x) ((peach(x)) → ((¬ (edible(x))) → (rotten(x))))
The Dodgers win the pennant	(win(Dodgers, pennant))
they will win the series	(win(they, series))
some officers are present	∃(x) ((officer(x)) ∧ (present(x)))
all officers are captains	∀(x) ((officer(x)) → (captain(x)))
some captains are present	∃(x) ((captain(x)) ∧ (present(x)))
Alexander died from typhoid	(died(Alexander, typhoid))
Lucy is a professor	(isprof(Lucy))
All professors are people	∀(x) ((isprof(x)) → (isperson(x)))
Deans are professors	∀(x) ((isdean(x)) → (isprof(x)))
Lucy criticized John	(criticize(Lucy, John))

5.3 Observations

We made some interesting observations in the process of building the current system. There are some interesting properties about the way Natural Language is being translated to First Order Logic that we understood. Some of them we will list down:

1. **No generic rules completely based on POS Tags**

   We cannot have generic rules completely based on the POS tags. Example sentences which have the POS tag sequence as **DT NN VB DT NN**,are:

   Every executive has a secretary

   $\forall$ (x) executive(x) $\rightarrow\exists$(y)secretary(y,x)

   Some executive has a secretary

   $\exists$ (x) executive(x) $\rightarrow\exists$(y)secretary(y,x)

   Both these sentences have very different FOL structure but the POS tag structure is very similar.

2. **Multiple rules for plural variation**

   In order to get correct english while going from logic to english it becomes necessary to specify in the rule whether the word is its singular form or plural form. e.g.,

   John looks for a **unicorn** unicorn(x): x is a **unicorn**

   $\exists$(x) ((unicorn(x)) $\wedge$(looks(John, x)))

   Nelson teaches a few **morons** moron(x): x is a **moron**

   $\exists$(x) ((moron(x)) $\wedge$(teaches(Nelson, x)))

   all officers are captains

   all students are smart

3. **Multiple rules for same sentence and predicate variation**

   Given that we have rules based on structure of sentence and its predicate, we need to have rules which have same sentence structure but have variation in their predicate definition. e.g.,

   Every flower has a fragrance

flower(x): x is a flower; **has(x,y): x has y**

∀(x) (flower(x) →has(x,fragrance))

Every executive has a secretary

executive(x): x is an executive; **secretary(x): x has a secretary**

∀(x) (executive(x) →secretary(x))

4. **POS tagger highly sensitive**

   We have observed that the stanford POS Tagger is highly sensitive with respect to the kind of tuning that we require. e.g.,

   x/SYM is/VBZ a/DT human/NN

   x/SYM is/VBZ human/JJ

   Lucy/NNP criticized/NNP John/NNP

   criticized/VBN

   We can see in this example how the POS tags of the words change with addition or deletion of words. We would have liked to have same POS tags for the words in these case.

5. **Lemma Dilemma**

   We encountered cases where even lemmatizing the words was of no use. This was the case because we had different version of the word in US English and UK English. One such example is,

Word	**Lemma**
criticize	criticize
criticise	criticise

Table 5.2: Lemma Dilemma

Here criticize is the US usage and criticise is the UK usage.

Chapter 6

Conclusions and Future Works

In this thesis we have given a system which translates Natural Language sentences to First Order Logic with the help of rules. We show that it is feasible to have rule based system which would help us in translating Natural Language propositions. We also formalize the problem by building a benchmark and having a framework where different algorithms could be implemented for this problem.

Major drawback of our system is that we need to have a lot of rules due to high complexity in the structure of Natural Language. Rules in our system target the full structure of the sentence in the non-recursive case, which adds to the count of rules. Optimal number of rules for such a system would be the number of distinct structure for First Order Logic we could have. We did clustering based on the First Order Logic structure on the benchmarks and found that we have 248 different structures for FOL in our benchmarks.

We worked on a approach which targets partial sentence. Main focus was to reduce the number of rules in our system. We built a prototype based on initial ideas and we got some encouraging results. We had around 160 sentence being translated with around 25 rules. Problem that we faced in this approach was that we were having multiple translations as rules were being applied on partial structures so the order of rule application was playing a role. We are still working on it and hope to get better results.

Appendix A

POS Tag Description

We give the complete list of Penn Tree Bank POS Tags. Fist column tell the POS Tag and second column gives the description for the corresponding POS Tag.

Table A.1: POS Tags

POS Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural

Continued on next page

Table A.1 – continued from previous page

POS Tag	Description
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

References

[1] *Coursera*. URL: https://www.coursera.org/.

[2] *Khan Academy*. URL: https://www.khanacademy.org/.

[3] Umair Z. Ahmed, Sumit Gulwani, and Amey Karkare. "Automatically Generating Problems and Solutions for Natural Deduction". In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. IJCAI '13. Beijing, China: AAAI Press, 2013, pp. 1968–1975. ISBN: 978-1-57735-633-2. URL: http://dl.acm.org/citation.cfm?id=2540128.2540411.

[4] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. "Automated feedback generation for introductory programming assignments". In: *Proceedings of the 34th SIGPLAN conference on Programming Language Design and Implementation*. ACM. 2013.

[5] Rohit Singh, Sumit Gulwani, and Sriram K. Rajamani. "Automatically Generating Algebra Problems". In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* 2012. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5133.

[6] Chris Alvin, Sumit Gulwani, Rupak Majumdar, and Supratik Mukhopadhyay. "Synthesis of Geometry Proof Problems". In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.* 2014, pp. 245–252. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8617.

[7] Rajeev Alur, Loris D'Antoni, Sumit Gulwani, Dileep Kini, and Mahesh Viswanathan. "Automated Grading of DFA Constructions". In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6759.

[8] John J. Kelly. *The Essence of Logic*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997. ISBN: 0-13-396375-6.

[9] Dave Barker-Plummer, Richard Cox, and Robert Dale. "Dimensions of Difficulty in Translating Natural Language into First-Order Logic". In: *Educational Data Mining - EDM 2009, Cordoba, Spain, July 1-3, 2009. Proceedings of the 2nd International Conference on Educational Data Mining*. 2009, pp. 220–229. URL: http://www.educationaldatamining.org/EDM2009/uploads/proceedings/barker.pdf.

[10] *Automata Tutor*. URL: http://www.automatatutor.com/about/.

[11] Luca Aceto. *Ode to the Automata Tutor*. URL: http://processalgebra.blogspot.in/2015/03/ode-to-automata-tutor.html.

[12] Dave Barker-Plummer, Richard Cox, and Robert Dale. "Student Translations of Natural Language into Logic: The Grade Grinder Translation Corpus Release 1.0". In: *Proceedings of the 4th International Conference on Educational Data Mining, Eindhoven, The Netherlands, July 6-8, 2011*. 2011, pp. 51–60. URL: http://educationaldatamining.org/EDM2011/wp-content/uploads/proc/edm2011_paper28_full_Barker-Plummer.pdf.

[13] Sumit Gulwani and Mark Marron. "NLyze: interactive programming by natural language for spreadsheet data analysis and manipulation". In: *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*. 2014, pp. 803–814. DOI: 10.1145/2588555.2612177. URL: http://doi.acm.org/10.1145/2588555.2612177.

[14] S. Torge N. E. Fuchs U. Schwertel. "Controlled Natural Language Can Replace First-Order Logic". In: *14th IEEE International Conference on Automated Software Engineering, ASE'99, Cocoa Beach, Florida, October 1999*. Cocoa Beach, Florida: IEEE, Oct. 1999, pp. 295–298. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=802325&tag=1.

[15] Luke S. Zettlemoyer and Michael Collins. "Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars". In: *CoRR* abs/1207.1420 (2012). URL: http://arxiv.org/abs/1207.1420.

[16] Yoav Artzi and Luke Zettlemoyer. "Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions". In: *Transactions of the Association for Computational Linguistics* 1.1 (2013), pp. 49–62.

[17] Chris Quirk, Pallavi Choudhury, Jianfeng Gao, Hisami Suzuki, Kristina Toutanova, Michael Gamon, Wen-tau Yih, Colin Cherry, and Lucy Vanderwende. "MSR SPLAT, a language analysis toolkit". In: *Proceedings of the Demonstration Session at the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Montrèal, Canada: Association for Computational Linguistics, 2012, pp. 21–24. URL: http://aclweb.org/anthology/N12-3006.

[18] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. "The Stanford CoreNLP Natural Language Processing Toolkit". In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 55–60. URL: http://www.aclweb.org/anthology/P/P14/P14-5010.