

# **CHAPTER 1**

## **Introduction**

We use the natural language in our everyday life to communicate with others. To understand the natural languages, human brain has perceptual and intuitional power. The problem of understanding natural language is a perceptual problem. The ability to use language to communicate a wide variety of ideas is perhaps the most important thing that separates humans from the other animals. To understand the sentences, it is required to know the semantic form of those sentences. It is a complex task to make machine understand and generate natural language. Question Answering (QA) is a new growing research area in Natural Language Processing (NLP). To answer the question, it might be needed knowledge about that question. By inferring the knowledge, a system can able to answer the question. Our system has introduced a question answering method in Natural Language Processing for semantic parsing. Semantic parsing is used for inferring the meaning of sentences.

### **1.1 Background**

Though we have come a long way in NLP, but one of the biggest challenges is resolving the many deep and complex semantic ambiguities that are an inherent property of natural language, and prevalent in most standard NLP tasks such as question answering. It is very tough to find semantic meaning from a given text. We need to introduce about parsing methods for semantic parsing the text. Suppose a given text is :

“Rose is a flower. It is red.”

If we ask the question “What is the color of Rose?”, here answer is “red”. But for finding this answer by machine, a relation must be made between the two sentences. Finding the dependency among the text, we can simulate the meaning of text. It is clear that having a proper understanding of text is a prerequisite for answering questions. Understanding the text and finding answer of the given question is the main goal of our thesis work. We have a sequence of input text and system can give answer corresponding to this text.

## **1.2 Statement of this problem**

The goal of this thesis is to find relevant facts from the texts and answer the questions. Two steps are needed here. Firstly, parse the input text and finding relevant facts. Finally, analyzing the question and implementing answering techniques.

## **1.3 Applications**

There are many real-world applications in NLP. The following tasks are common research field in NLP and QA techniques:

### **i) Automatic Summarization**

A readable summary of chunk of text is produced, such as summary of a bulletin in a school.

### **ii) Discourse Analysis**

The rubric includes a number of related tasks. One task is identifying the discourse structure of connected text, i.e. the nature of the discourse relationships between sentences.

### **iii) Machine Translation**

The texts are translated automatically from human language to another. It is very difficult problem required types of knowledge (grammar, semantics, facts etc.).

### **iv) Name Entity Recognition (NER)**

Given a sequence of text, determine which item from the text map to proper names such as person, places or organizations.

### **v) Parts-of-speech tagging**

Determine the part of speech from a given text.

### **vi) Parsing**

Determine the parse tree form a sentence to find logical structure.

### **vii) Sentiment Analysis**

Extract subjective information usually from a set of documents, often using online reviews to determine "polarity" about specific objects. It is especially useful for identifying trends of public opinion in the social media, for the purpose of marketing.

### **viii) Question Answering**

Given a human language question and determine the answer. It would be used in the admission examinations, robotic system for textual and visual question answering. This technique analyses the relationship between human intelligence and artificial intelligence.

### **ix) Some other applications of QA:**

- a. Answer reuse or caching
- b. Question of definition and terminology
- c. Multilingual question
- d. Social media analysis with QA
- e. Biographical questions
- f. Utilization of linguistic resources, such as WordNet, FrameNet
- g. Automated essay scoring

## **1.4 Objectives**

The objectives of this thesis are given below:

- i) Massive knowledge found as natural language
- ii) Text based question answering
- iii) NLP based problem solving system
- iv) Finding semantic meaning of sentences
- v) Ultimate goal is to get closer to strong Artificial Intelligence (AI)

## **1.5 Scope and Limitations**

The area of this thesis is vast. The scopes are given below:

- a. Finding semantic knowledge from text.
- b. Answering the question, the system would like to be an expert system.
- c. Automated question answering may be used in the examination system.

Some limitations might be in every research area. This thesis has some limitations. Those are below:

- a. Low accuracy for parsing text in complex and compound sentences.
- b. For QA, it is needed to input a sequence of sentences which have sequential meaning.

## **1.6 Outline and Contributions**

The outline of this thesis is as follows. It has covered total 7 chapters. In the current chapter (chapter 1), we introduce the background of this thesis and define scope & limitations. In chapter 2, we discuss briefly about the Natural Language Processing in current world. In chapter 3, various types of parsing technique are represented to analyze the text. Chapter 4 covers the question answering technique in NLP. We discuss the implementation procedure of our thesis in chapter 5. Results and analysis of current system are mentioned in chapter 6. We test the system with different types of dataset and compare the results with others. Chapter 7 is the last chapter of this thesis in which we describe the conclusion of the thesis and future work for this thesis.

## **Conclusion**

This thesis focuses on the question answering technique in NLP based system. We also introduce the parsing techniques. An abstract knowledge of our thesis work might be understood in this chapter. We discuss our system in brief in the next chapters.

# Natural Language Processing

Natural Language Processing (NLP) is a field of computer science, artificial intelligence (AI) and computer linguistics which is concerned about interactions between computer and human languages. It is a related area of human-computer interactions. NLP is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring text at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications [1]. Research in natural language processing (NLP) is aimed at making machines understand and generate natural language. There has been a lot of progress in this field. The field of NLP is included semantic structures of sentences, learning by machine, question answering etc. NLP is one of the enormous fields in AI of computer science. Linguistic processing is one of the common parts in modern research area which is related to NLP.

Every day we use natural language to communicate with others. From the ancient period of time every human being would express their thoughts to others by their own languages. There are many different types of languages in the world. Every language has its own grammar, vocabulary. Sometimes to express feelings and thoughts, people may not follow the rules of grammar. We can give example of English language. It has its own vocabulary and grammar.



## 2.2 Processing of Natural Language in Machine

At the core of any NLP task there is the important issue of natural language understanding. The process of building computer programs that understand natural language involves three major problems: the first one relates to the thought process, the second one to the representation and meaning of the linguistic input, and the third one to the world knowledge. Thus, an NLP system may begin at the word level - to determine the morphological structure, nature (such as part-of-speech, meaning) etc. of the word- and then may move on to the sentence level- to determine the word order, grammar, meaning of the entire sentence, etc. A given word or a sentence may have a specific meaning or connotation in a given context or domain, and may be related to many other words and/or sentences in the given context [2].

Will a computer program ever be able to convert a piece of English text into a programmer friendly data structure that describes the meaning of the natural language text? Unfortunately, no consensus has emerged about the form or the existence of such a data structure. Until such fundamental Artificial Intelligence problems are resolved, computer scientists must settle for the reduced objective of extracting simpler representations that describe limited aspects of the textual information [3].

These simpler representations are often motivated by specific applications (for instance, bag-of-words variants for information retrieval), or by our belief that they capture something more general about natural language. They can describe syntactic information (e.g., part-of-speech tagging, chunking, and parsing) or semantic information (e.g., word-sense disambiguation, semantic role labeling, named entity extraction, and anaphora resolution). Text corpora have been manually annotated with such data structures in order to compare the performance of various systems. The availability of standard benchmarks has stimulated research in Natural Language Processing (NLP) and effective systems have been designed for all these tasks. Such systems are often viewed as software components for constructing real-world NLP solutions [3].

The overwhelming majority of these state-of-the-art systems address their single benchmark task by applying linear statistical models to ad-hoc features. In other words, the researchers themselves discover intermediate representations by engineering task-specific features. These features are often derived from the output of preexisting systems, leading to complex runtime

dependencies. This approach is effective because researchers leverage a large body of linguistic knowledge. On the other hand, there is a great temptation to optimize the performance of a system for a specific benchmark. Although such performance improvements can be very useful in practice, they teach us little about the means to progress toward the broader goals of natural language understanding and the elusive goals of Artificial Intelligence.

We discuss here some benchmark task Part-Of-Speech tagging (POS), chunking (CHUNK), Named Entity Recognition (NER) and Semantic Role Labeling (SRL).

### 2.3 Parts-of-Speech Tagging

POS aims at labeling each word with a unique tag that indicates its syntactic role, for example, plural, noun, adverb etc. The best POS classifiers are based on classifiers trained on windows of text, which are then fed to a bidirectional decoding algorithm during inference. The basic idea of POS tagging is shown in below:

- **Input:** A string of word and a specified Tagset.
- **Output:** single best tag for each word.

We see an example of POS tagging:

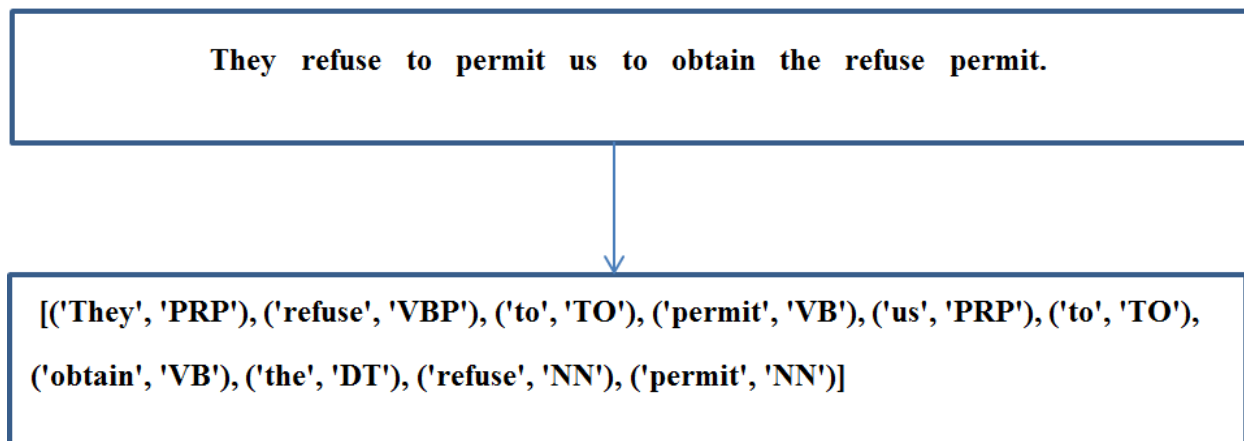


Figure 2.2: POS tagger of an input sentence

We see in the figure 2.2, every word is tagged to its corresponding parts of speech. It used tagset for every word. We introduce the Penn Treebank of POS tags [4].

### Penn TreeBank POS Tags

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition
JJ	Adjective
JJR	Adjective , comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun , plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun

Tag	Description
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-3 <sup>rd</sup> person singular present
VBZ	Verb, 3 <sup>rd</sup> person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP\$	Possessive wh-pronoun
WRB	Wh-adverb

Table 2.1: Penn Treebank POS tags



We see from the previous Penn Treebank in table 2.1, there are different forms of verbs in English language. Let to see them in the below:

Base <b>VB</b>	-S <b>VBZ</b>	Pres <b>VBP</b>	Past <b>VBD</b>	Pres_participle <b>VBG</b>	Past_participle <b>VCN</b>
Write	writes	write	wrote	written	writing
Be	is	am, are	was, were	been	being
Do	does	do	did	done	doing
Have	has	have	had	had	having

Figure 2.3: Verb Forms in English

There are different types of POS tagger in NLP. We discuss about some of them in below.

#### a. NLTK POS tagger

NLTK, the Natural Language Toolkit, is a suite of open source program modules, tutorials and problem sets, providing ready-to-use computational, linguistics courseware [5]. NLTK covers, symbolic and statistical natural language processing, and is interfaced to annotated corpora. The choice of programming language for NLTK is Python. For the choice of Python language is for following criteria:

First, the language must have a shallow learning curve, so that novice programmers get immediate rewards for their efforts. Second, the language must support rapid prototyping and a short develop/test cycle; an obligatory compilation step is a serious detraction. Third, the code should be self-documenting, with a transparent syntax and semantics. Fourth, it should be easy to write structured programs, ideally object-oriented but without the burden associated with languages. Finally, the language must have an easy-to-use graphics library to support the development of graphical user interfaces. After considering all of the criteria, Python is the best for satisfying all of these. So Python is used for NLTK.

The requirements of NLTK are ease of use, consistency, extensibility, documentation, simplicity, modularity.

An example of NLTK POS tagger for tagging a sentence is shown in below:

## NLTK tagging

```
>>> text = nltk.word_tokenize("And now for something  
completely different")  
  
>>> nltk.pos_tag(text)  
  
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something',  
 'NN'), ('completely', 'RB'), ('different', 'JJ')]
```

Figure 2.4: NLTK POS tagging

### b. Brill's Transformation based taggers

In the Brill's supervised learning model "transformation-based error-driven learning" works as follows. The training data is correctly annotated text. The corresponding raw text is fed through an initial-state annotator, which makes a more or less well-informed initial guess of how the text should be annotated. This initial annotation is compared to the training data as a basis for learning a sequence of TRs, which are context dependent 'correction' rules that apply in sequence to modify the initial annotation to better approximate the training data. For POS tagging, the initial-state annotator assigns each known word its most probable tag from amongst those listed in a lexicon, as determined from some training corpus. The initial tagging of unknown words can be handled in a number of ways, using clues such as affixes and capitalism.

An advantage of Brill's approach is that it's learned model is quite compact, consisting of a few hundred rules that can be directly inspected. In Brill's experiments, training on 600K tokens of the PTB tagged Wall Street Journal corpus under the 'closed vocabulary assumption' (where there are no unknown words) gave tagging accuracy of 97.2% [6]. Without this assumption, where performance also depends on the handling unknown words, the score was 96.6%.

The algorithm of Brill's tagger training is shown below:

Load training corpus into memory (array), i.e. words + correct tag, and make initial tag assignment (assign most-probable tag to each word).

Then, for each tag  $x$ , learn list of rules that apply for this tag as follows:

1. Scan corpus, and record 'data points' for this phase, i.e. positions where default tag is  $x$  and word has at least one alternative tag (store these array offsets)
2. Compute initial TR correction scores. Firstly, scan for data points where tag  $x$  is incorrect, and at each score +1 for all possible TRs changing  $x$  to the correct tag (scores stored in a hash). Secondly, at each point where tag  $x$  is correct, score -1 for all possible TRs changing  $x$  to any lexical alternative — but *only* for rules already present in scores hash (i.e. rules effecting at least one correction).
3. Loop acquiring rules as follows:  
Scan scores hash for best rule  $T$  (exit loop if score not above threshold) and add to rule list. Update scores hash as follows: scan for data points where rule  $T$  fires, and update scores for rules that fire at these positions. (Update scoring details given in text.)

Figure 2.5: Training algorithm of Brill's tagger

Figure 2.5 omits the specifics of how rule scores are incrementally updated [6], as this differs between appending and prepending approaches. An 'efficiency feature' of the algorithm is that in computing the initial scores for TRs, we firstly identify the rules that effect at least one correction somewhere, and then subsequently only score negative changes for these rules.

Three major stages (of Brill's Transformation-Based Learning (TBL) algorithm) are:

- It labels every word with its most-likely tag.
- It examines every possible transformation and selects the one that results in the most improved tagging.
- It then re-tags the data according to the rules.

TBL is a supervised learning technique; it assumes a pre-tagged training corpus.

## 2.4 Chunking

Text chunking is a useful preprocessing step for parsing. There has been large interest in recognizing non-overlapping noun phrases but relatively little has been written about identifying phrases of other categories. Text chunking consists of dividing a text into phrases in such a way

that syntactically related words become member of the same phrase [7]. These phrases are non-overlapping which means that one word can only be a member of one chunk. It is also called shallow parsing, chunking aims at labeling segments of a sentence with syntactic constituents such as noun or verb phrases (NP or VP). Each word is assigned only one unique tag, often encoded as a begin-chunk (e.g., B-NP) or inside-chunk tag (e.g., I-NP). Here is an example of sentence:

[NP He ] [VP reckons ] [NP the current  
account deficit ] [VP will narrow ]  
[PP to ] [NP only £ 1.8 billion ]  
[PP in ] [NP September ] .

Chunks have been represented as groups of words between square brackets. A tag next to the open bracket denotes the type of the chunk. As far as we know, there are no annotated corpora available which contain specific information about dividing sentences into chunks of words of arbitrary types.

Chunking is often evaluated using the CoNLL 2000 shared task [7]. Sections 15–18 of WSJ data are used for training and section 20 for testing. Validation is achieved by splitting the training set [3]. Kudoh and Matsumoto (2000) [7] won the CoNLL 2000 challenge on chunking with a F1-score of 93.48%. Their system was based on Support Vector Machines (SVMs). Each SVM was trained in a pairwise classification manner, and fed with a window around the word of interest containing POS and words as features, as well as surrounding tags. They perform dynamic programming at test time. Later, they improved their results up to 93.91% (Kudo and Matsumoto, 2001) using an ensemble of classifiers trained with different tagging conventions.

## 2.5 Name Entity Recognition

Named-entity recognition (NER) (also known as entity identification, entity chunking and entity extraction) is a subtask of information extraction that seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

NER systems have been created that use linguistic grammar-based techniques as well as statistical models, i.e. machine learning. Hand-crafted grammar-based systems typically obtain better precision, but at the cost of lower recall and months of work by experienced computational linguists. Statistical NER systems typically require a large amount of manually annotated training data. Semi-supervised approaches have been suggested to avoid part of the annotation effort. With a substantial amount of annotated data and a strong evaluation methodology in place, the focus of research in this area has almost entirely been on developing language-independent systems that learn statistical models for NER. The competing systems extract terms and patterns indicative of particular ne types, making use of many types of contextual, orthographic, linguistic and external evidence [9]. We can follow an NE (Name Entity) Example:

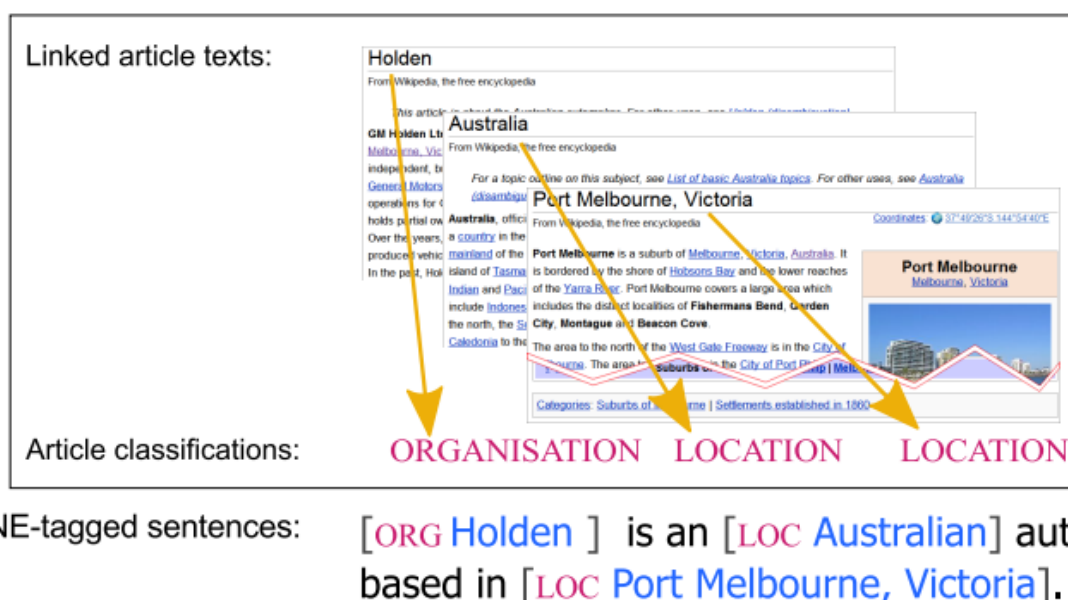


Figure 2.6: Deriving training sentences from Wikipedia text to translate NE categories

## 2.6 Semantic Role Labeling

Semantic role labeling, sometimes also called shallow semantic parsing, is a task in natural language processing consisting of the detection of the semantic arguments associated with the predicate or verb of a sentence and their classification into their specific roles [9]. For example, given a sentence like "Mary sold the book to John", the task would be to recognize the verb "to sell" as representing the predicate, "Mary" as representing the seller (agent), "the book"

as representing the goods (theme), and "John" as representing the recipient. This is an important step towards making sense of the meaning of a sentence. A semantic representation of this sort is at a higher-level of abstraction than a syntax tree. For instance, the sentence "The book was sold by Mary to John" has a different syntactic form, but the same semantic roles.

Given a preprocessed sentence  $x$  and a marked predicate  $t$  with lemma  $l$ , we seek to predict the frame  $f$  instantiated by the predicate. They defined the probability of a frame  $f$  under a conditional log-linear model:

$$p(f | x, t, l) \propto \exp(\psi \cdot h(f, x, t, l))$$

where  $\psi$  denotes the model parameters and  $h(\cdot)$  is the feature function [9].

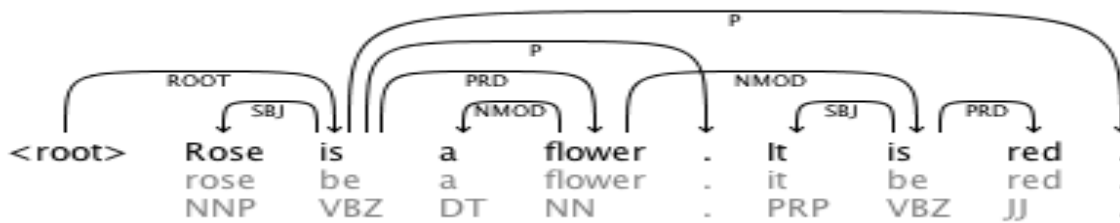


Figure 2.7: Semantic role labeller for an input text

The following figure 2.7 shows a semantic role labeling for a sequence of text. The input text is "Rose is a flower. It is red. ". The role labeller makes the relation between two sentences.

## Conclusion

We have discussed about the area of Natural Language Processing. How NLP can be used in machine, is also described in this chapter. POS tagging, chunking, name entity recognition, semantic role labeling are the some steps of NLP, described briefly. Some example of text processing is shown in this chapter. We have introduced a close idea of NLP through this chapter and other parts of our thesis work are discussed next chapters.

## Chapter 3

### Parsing Techniques

#### Introduction

‘Parsing’ is the term used to describe the process of automatically building syntactic analyses of a sentence in terms of a given grammar and lexicon. The resulting syntactic analyses may be used as input to a process of semantic interpretation, (or perhaps phonological interpretation, where aspects of this, like prosody, are sensitive to syntactic structure). Occasionally, ‘parsing’ is also used to include both syntactic and semantic analysis. We use it in the more conservative sense here, however. In most contemporary grammatical formalisms, the output of parsing is something logically equivalent to a tree, displaying dominance and precedence relations between constituents of a sentence, perhaps with further annotations in the form of attribute-value equations (‘features’) capturing other aspects of linguistic description. However, there are many different possible linguistic formalisms, and many ways of representing each of them, and hence many different ways of representing the results of parsing. We shall assume here a simple tree representation, and an underlying context-free grammatical (CFG) formalism. However, all of the algorithms described here can usually be used for more powerful unification based formalisms, provided these retain a context-free ‘backbone’, although in these cases their complexity and termination properties may be different.

We can see parsing a sentence:

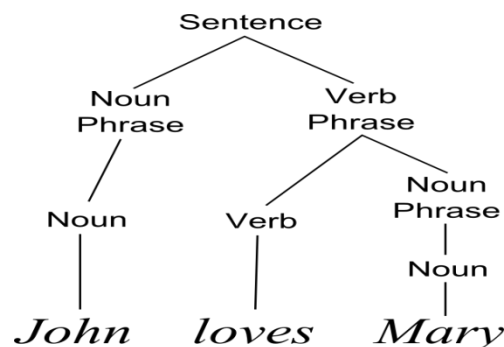


Figure 3.1: A parse tree of a sentence

We introduce both syntactic and semantic parsing. Now both types are discussed in next:

### 3.1 Syntactic Parsing

The syntactic parsing deals with the analysis of the structure of the sentences and texts, for example understanding which part is the predicate and in that which word/s is/are the verbs.

#### 3.1.1 Top-Down Parsing

Top-down parsing is a parsing strategy where one first looks at the highest level of the parse tree and works down the parse tree by using the rewriting rules of a formal grammar. It can be viewed as an attempt to find left-most derivations of an input-stream by searching for parse-trees using a top-down expansion of the given formal grammar rules. Tokens are consumed from left to right.

We can illustrate a simple top down algorithm for a CFG as follows. Let the grammar contain the following rules (which for simplicity also introduce lexical entries instead of presupposing a separate component for this):

$S \rightarrow NP VP$

$NP \rightarrow \text{they} \mid \text{fish}$

$VP \rightarrow \text{Aux } VP$

$VP \rightarrow V_i$

$VP \rightarrow V_t NP$

$\text{Aux} \rightarrow \text{can}$

$V_i \rightarrow \text{fish}$

$V_t \rightarrow \text{can}$

This will assign to the sentence ‘they can fish’ two distinct analyses corresponding to two interpretations ‘they are able/permitted to fish’ and ‘they put fish in cans’. It will also generate several other good sentences and lots of odd ones [11].



The top down algorithm is very simple. We begin with the start symbol, in this case, S, and see what rules it figures in as the mother. Then we look at the daughters of these rules and see whether the first one matches the next word in the input. If it does, we do the same for the remaining daughters. If not, we go through the same loop again, this time matching the daughters of the rules already found with mothers of other rules. Of course, in parsing this sentence we have magically chosen the correct rule at each point, and ignored choices which would not lead to a successful parse. To fully specify the algorithm we would need to eliminate all this magic and provide an exhaustive mechanism for trying out all possibilities. We can do this by non-deterministically trying out all alternatives at each point as they arise, or, more flexibly, by casting our algorithm in the form of a set of operations on representations that encode the state of a parse. Each parsing step takes an ‘item’, as we shall call them, and produces a set of new items.

These alternatives can be pursued in whatever order is convenient. Parsing proceeds from an initial seed item and ends when no more steps are possible. Each remaining item, if there are any, will then represent a successful parse of the input.

In the previous page we see a general structure of a top down parsing. A top down parsing is like that:

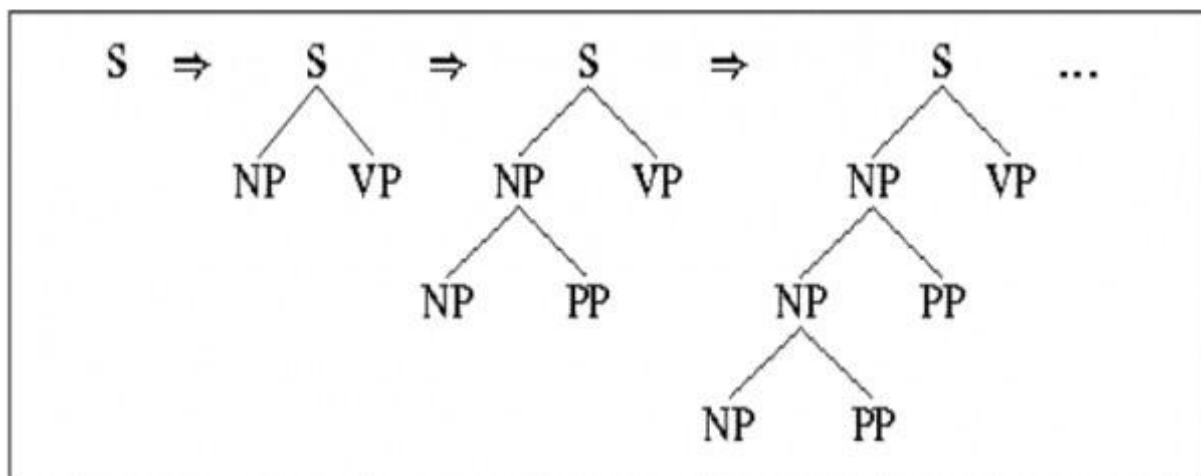


Figure 3.2: Top down parsing

### 3.1.2 Bottom up parsing

Bottom up parsing works opposite of top down parsing. A bottom-up parse discovers and processes that tree starting from the bottom left end, and incrementally works its way upwards and rightwards. Bottom up parsing works opposite of top down parsing. A bottom-up parse discovers and processes that tree starting from the bottom left end, and incrementally works its way upwards and rightwards. A parser may act on the structure hierarchy's low, mid, and highest levels without ever creating an actual data tree, the tree is then merely implicit in the parser's actions. The operation of a bottom up algorithm for CFG can be illustrated by the following sequence of operations for 'they fish':

Structure	Input
so far	remaining
	[they fish]
[NP they]	[fish]
[NP they] [Vi fish]	[]
[NP they] [Vp [Vi fish]]	[]
[S [NP they] [Vp [Vi fish]]]	[]

Here the matching words to the right hand sides of rules, and then matching the resulting symbols, or sequences of symbols, to the right hand sides of rules until we have an S covering the whole sentence. Again, we have magically made the right choices at each point.

A pictorial representation of bottom up parsing is shown below:

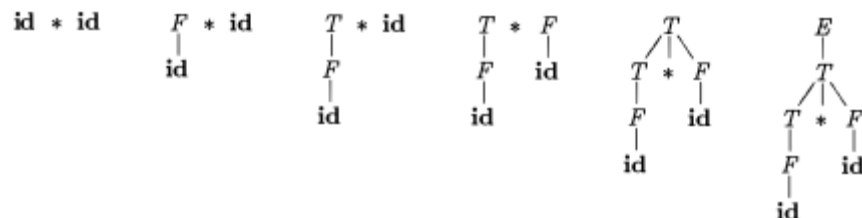


Figure 3.3: Bottom up parsing

### 3.2 Semantic Parsing

Semantic Parsing is probably best defined as the task of representing the meaning of a natural language sentence in some formal knowledge representation language that supports automated inference. A semantic parser is best defined as having three parts, a formal language, an ontology, and an inference mechanism. Both the formal language (e.g. first-order logic) and the ontology define the formal knowledge representation [12]. The formal language uses predicate symbols from the ontology, and the ontology provides them with meanings by defining the relations between them. Semantic and pragmatic analysis make up the most complex phase of language processing as they build up on results of all the above mentioned disciplines. Based on the knowledge about the structure of words and sentences, the meaning of words, phrases, sentences and texts is stipulated, and subsequently also their purpose and consequences. From the computational point of view, no general solutions that would be adequate have been proposed for this area.

The goal of the semantic parser is to analyze the semantic structure of a natural language sentence. Similar in spirit with the syntactic parser – whose goal is to parse a valid natural language sentence into a parse tree indicating how the sentence can be syntactically decomposed into smaller syntactic constituents – the purpose of the semantic parser is to analyze the structure of sentence meaning. Sentence meaning is composed by entities and interactions between entities, where entities are assigned semantic roles, and can be further modified by other modifiers. The meaning of a sentence is decomposed into smaller semantic units connected by various semantic relations by the principle of compositionality, and the parser represents the semantic structure – including semantic units as well as semantic relations, connecting them into a formal format [13].

Some semantics parsing techniques are developed. These semantic parsing techniques are described next:

#### 3.2.1 Lambda Calculus Semantic Parsing

Lambda calculus (also written as  $\lambda$ -calculus) is a formal system in mathematical logic for expressing computation based on a function abstraction and application using variable binding and substitution. Computable functions are a fundamental concept within computer science and

mathematics. The  $\lambda$ -calculus provides a simple semantics for computation, enabling properties of computation to be studied formally. The  $\lambda$ -calculus incorporates two simplifications that make this semantics simple. The first simplification is that the  $\lambda$ -calculus treats functions "anonymously", without giving them explicit names. For example, the function:

$$\text{square\_sum}(x,y) = x^2 + y^2$$

can be written anonymous form

$$(x,y) \rightarrow x^2 + y^2$$

(read as "the pair of  $x$  and  $y$  is mapped to  $x^2 + y^2$ "). Similarly,

$$\text{id}(x) = x$$

can be rewritten in anonymous form as  $x \rightarrow x$ , where the input is simply mapped to itself.

## Lambda Terms

The lambda calculus consists of a language of lambda terms, which is defined by a certain formal syntax, and a set of transformation rules, which allow manipulation of the lambda terms. These transformation rules can be viewed as an equational theory or as an operational definition. The syntax of the lambda calculus defines some expressions as valid lambda calculus expressions and some as invalid, just as some strings of characters are valid C programs and some are not. A valid lambda calculus expression is called a "lambda term".

The following three rules give an inductive definition that can be applied to build all syntactically valid lambda terms:

- a variable,  $x$ , is itself a valid lambda term
- if  $t$  is a lambda term, and  $x$  is a variable, then  $(\lambda x. t)$  is a lambda term (called a **lambda abstraction**)
- if  $t$  and  $s$  are lambda terms, then  $(ts)$  is a lambda term (called an application).

## Dependency tree

Semantic parsers map sentences onto logical forms. Dependency tree can be composed by lambda-calculus expression [14]. We follow the example:

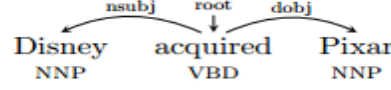


Figure 3.4: Dependency tree for a sentence

The s-expression for the dependency tree is :

(nsubj (dobj acquired Pixar) Disney)

Now finding the composed lamda-calculus expression:

$\lambda x . \exists yz. \text{acquired}(x_e) \wedge \text{Disney}(y_a) \wedge \text{Pixar}(z_a) \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, z_a)$

## Logical Forms

A version of the lambda calculus with three base types: individuals (**Ind**), events (**Event**), and truth values (**Bool**), events (**Event**), and truth values (**Bool**). For types A and B, we use  $A \times B$  to denote the product type, while  $A \rightarrow B$  denotes the type of functions mapping elements of A to elements of B. An important constraint on the lambda calculus system is as follows: All natural language constituents have a lambda-calculus expression of type **Ind**  $\times$  **Event**  $\rightarrow$  **Bool**. Some examples of lambda-calculus expressions for single words (lexical entries) are as follows:

$\text{acquired} \Rightarrow \lambda x. \text{acquired}(x_e)$

$\text{Disney} \Rightarrow \lambda y. \text{Disney}(y_a)$

$\text{Pixar} \Rightarrow \lambda z. \text{Pixar}(z_a)$

An example for a full sentence is as follows:

$\text{Disney acquired Pixar} \Rightarrow \lambda x. \exists yz. \text{acquired}(x_e) \wedge \text{Disney}(y_a) \wedge \text{Pixar}(z_a) \wedge \text{arg}_1(x_e, y_a) \wedge \text{arg}_2(x_e, z_a)$

This is a neo-Davidsonian style of analysis. Verbs such as acquired make use of event variables such as  $x_e$ , whereas nouns such as Disney make use of individual variables such as  $y_a$ . The transformation of a dependency tree to its logical form is accomplished through a series of three steps: binarization, substitution, and composition [14]. Below, we outline these steps, with some additional remarks.

### **Binarization**

A dependency tree is mapped to an s-expression (borrowing terminology from Lisp). For example, Disney acquired Pixar has the s-expression

(nsubj (dobj acquired Pixar) Disney)

### **Substitution**

Each symbol (word or label) in the s-expression is assigned a lambda expression.

### **Composition**

Beta-reduction is used to compose the lambda-expression terms to compute the final semantics for the input sentence. In this step expressions of the form (exp1 exp2 exp3) are interpreted as function exp1 being applied to arguments exp2 and exp3. For example, (dobj acquired Pixar) receives the following expression after composition:

$\lambda z. \exists x. \text{acquired}(z_e) \wedge \text{Pixar}(x_a) \wedge \text{arg}(z_e, x_a)$

To find the dependency tree of a simple sentence by lambda-calculus, we can summarize the algorithm:

- a) take input of a sentence
- b) get the token from each word by POS tagging
- c) set the main verb as root of sentence
- d) make the dependency between subject and object of that sentence

We view the flowchart of lambda-calculus dependency parsing for simple sentence in next page.

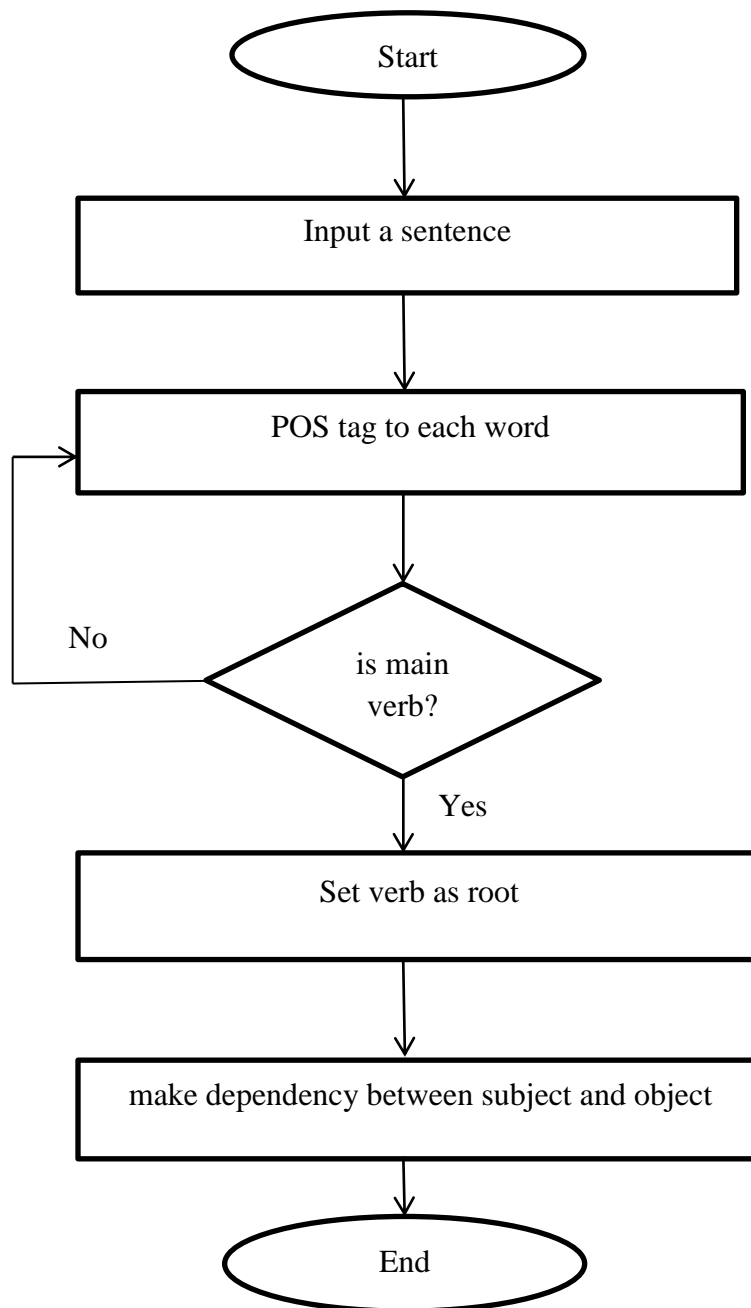


Figure 3.5: flowchart of lambda-calculus dependency parsing

### 3.2.2 Knowledge Bases for Semantic Parsing

One major problem faced by many natural language understanding applications that rely on syntactic analysis of text, is the fact that similar syntactic patterns may introduce different semantic interpretations. Likewise, similar meanings can be syntactically realized in many different ways. The semantic parser attempts to solve this problem, and produces a syntax-independent representation of sentence meaning, so that semantic constituents can be accessed and processed in a more meaningful and flexible way, avoiding the sometimes rigid interpretations produced by a syntactic analyzer. For instance, the sentences I boil water and water boils contain a similar relation between water and boil, even though they have different syntactic structures. To deal with the large number of cases where the same syntactic relation introduces different semantic relations, we need knowledge about how to map syntax to semantics. To this end, we use two main types of knowledge – about words, and about relations between words. The first type of knowledge is drawn from WordNet – a large lexical database with rich information about words and concepts. We refer to this as word-level knowledge. The latter is derived from FrameNet – a resource that contains information about different situations, called frames, in which semantic relations are syntactically realized in natural language sentences. We call this sentence-level knowledge. In addition to these two lexical knowledge bases, the parser also utilizes a set of manually defined rules, which encode mappings from syntactic structures to semantic relations, and which are also used to handle those structures not explicitly addressed by FrameNet or WordNet. For instance, Table 1 lists the syntactic and semantic features [13] extracted from FrameNet for the Sentence I had chased Selden over the moor.

	I	had chased	Selden	over the moor
GF	Ext		obj	comp
PT	NP	Target	NP	PP
Position	before		after	after
Voice		active		
PP				over
Role	Theme		Goal	Path

Table 3.1: Example sentence with syntactic and semantic features



## Conclusion

In this chapter, we have introduced about both syntactic and semantic parsing techniques. Both are needed for NLP. In the syntactic parsing, top down and bottom up parsing method are discussed. In semantic parsing techniques, we have presented a method for converting dependency structures to logical forms using lambda calculus method. The key idea is the use of a single semantic type for every constituent of the dependency tree, which provides us with a robust way of compositionally deriving logical forms. Also an algorithm for open text shallow semantic parsing has shown here. The algorithm has the capability to analyze the semantic structure of a sentence, and show how the meaning of the entire sentence is composed of smaller semantic units, linked by various semantic relations. The parsing process utilizes linguistic knowledge, consisting of rules derived from a frame dataset (FrameNet), a semantic network (WordNet), as well as hand-coded rules of syntax-semantics mappings, which encode natural selectional restrictions. We have briefly described about lambda calculus parsing method because we use that in our work.

## CHAPTER 4

### QA Techniques

#### Introduction

Question answering (QA) is a complex natural language processing task which requires an understanding of the meaning of a text and the ability to reason over relevant facts. It is not only an interesting and challenging application, but also the techniques and methods developed from Question answering inspires new ideas in many closely related areas such as document retrieval, time and named-entity expression recognition, etc. The first type of questions that research focused on was factoid questions. For example, “When was X born?”, “In what year did Y take place?”. The recent research trend is shifting toward more complex types of questions such as definitional questions (biographical questions such as “Who is Hilary Clinton?”, and entity definition questions such as “What is DNA?”), list questions (e.g. “List the countries that have won the World Cup”), scenario-based QA (given a short description of a scenario, answer questions about relations between entities mentioned in the scenario) and why-type questions.

#### 4.1 A Brief Overview of QA systems

Most tasks in natural language processing can be cast into question answering (QA) problems over language input. Factoid question answering is the most widely studied task in question answering. In this section we survey several different techniques to answer extraction for factoid question answering, which aims at accurately pin-pointing the exact answer in retrieved documents. In a QA evaluation track, each system is given a document collection, a set of training questions, a gold-standard answer set, and a set of testing questions. The document collection consists of newswire articles collected from one or multiple news agencies over usually a couple of years. In most cases, these collections contain several million documents. The training set consists of questions taken from past years’ test sets plus some additional ones.

A typical QA system usually employs a pipeline architecture that chains together three main modules [15]:

- **Question analysis module:** this module processes the question, analyzes the question type, and produces a set of keywords for retrieval. Depending on the retrieval and answer extraction strategies, some question analysis module also performs syntactic and semantic analysis of the questions, such as dependency parsing and semantic role labeling.
- **Document or passage retrieval module:** this module takes the keywords produced by the question analysis module, and uses some search engine to perform document or passage retrieval.
- **Answer extraction module:** given the top N relevant documents or passages from the retrieval module, the answer extraction module performs detailed analysis and pin-points the answer to the question. Usually answer extraction module produces a list of answer candidates and ranks them according to some scoring functions.

Now we discuss some of the steps of QA structures:

## 4.2 Answer Extraction by Structural Information Matching

In this section, we will closely examine several answer extraction techniques. At a high abstraction level, different answer extraction approaches can be described in a general way. Under this generic view, we will focus our discussion around the following questions [15]:

- What sources of information are useful for finding answers?
- How do we obtain and represent useful information?
- How do we combine multiple information sources in making a unified decision?

### 4.2.1 Identifying useful information for extracting answers

Before we examine specific methods and models for extracting answers, it is worth spending some time to think about what kind of information is useful in helping us finding answers. Another source of information that is used by almost all question answering systems is named-

entity (NE). The idea is that factoid questions fall into several distinctive types, such as “location”, “date”, “person”, etc. Assuming that we can recognize the question type correctly, then the potential answer candidates can be limited down to a few NE types that correspond to the question type. Intuitively, if the question is asking for a date, then an answer string that is identified to be a location type named-entity is not likely to be the correct answer. However, it is important to bear in mind that neither question type classification nor NE recognition are perfect in the real world.

#### 4.2.2 Structural Information Extraction and Representation

A case of selectively using syntactic features is the work of [16] which has six syntactically motivated heuristic factors [16]:

1. the size of the longest phrase in the question matched in the answer sentence.
2. the surface distance between answer candidate and the main verb in the answer sentence.
3. for “PERSON” type questions, a Boolean feature indicates whether the syntactic relationship (either “passive” or “active”) between the answer candidate and the main verb is the same as the relationship in the question.
4. For “LOCATION” questions, check the possessive formats such as, “Venezuela’s Orinoco”.
5. For “LOCATION” and “DATE” questions, check whether the candidate is inside a prepositional phrase and modifies the main verb.
6. For “PERSON” type questions, check if both the answer candidate and all question key words fall inside an adjective noun phrase.

The three types of linguistic constructs listed in her paper are:

1. **Is-relationship:** a feature that activates when the answer candidate is the subject or object of a “be” verb.
2. **Apposition:** a feature that activates when the answer candidate is followed or preceded immediately by a comma.
3. **Subject-Verb/Verb-Object:** a feature that activates when the answer candidate is the subject or object of a non-stop word verb.

### **4.2.3 Models for combining multiple sources of information in answer scoring**

Nearly all QA systems use evidence from multiple sources to decide what the best answer is for a given question, and then ranked answer candidates based on four types of evidence:

1. the count of NE types that occurred in the question occurring in the answer sentence, normalized by the number of NE types occurred in the question.
2. the count of constraints other than NE type constraint that occurred in the question occurring in the answer sentence, normalized by the number of constraints occurred in the question.
3. a pre-set score when the answer candidate string contains the identified question focus string.
4. a pre-set score the answer candidate string appear in the answer sentence next to a term that contains the question focus string.

### **4.3 NLP-based Problem Solving (Todai-robot Project)**

This system is developed for the ‘Todai Robot Project’, which focuses on benchmarking NLP systems for problem solving [17]. We report the organization of the Todai Robot Project and the results of its first open evaluation. The Todai Robot Project aims to develop problem-solving systems that can attain high scores in the Japanese National Center Test for University Admissions by 2016 and pass the University of Tokyo’s entrance examination by 2021. For this task, the organizers provide several shared resources for task-takers, which comprise mainly questions in university entrance examinations and textbooks that possess information required to solve problems. Some of the resources were translated into English and are open to all task-takers. The questions are primarily collected from the Japanese National Center Test for University Admissions (Center Test) and its trial examination held by a prep school. Questions in university entrance examinations are mostly presented in natural language; therefore, it is clear that having a proper understanding of texts is a prerequisite for solving questions. Such questions are carefully designed to empirically quantify the academic skills of high school students. In this regard, these resources are ideal for evaluating end-to-end NLP systems that read natural language text, perform information processing and output answers.

## Resources

There are eleven subtasks corresponding to each subject in the Center Test: Biology, Chemistry, Ethics, English (foreign language), Japanese (native language), Japanese History, Mathematics, Modern Society, Physics, Politics and Economics and World History. All subjects in the Center Test, excluding Mathematics, consist of multiple choice questions.

## Dataset Preparation

We collected PDFs and source texts from the National Center Test for University Admission in Japan and the Yoyogi Seminar prep school. The examination datasets have been annotated with information required for NLP benchmarking. The types of annotation are as follows:

- Document structure
- Formula
- Image
- Question-type

## Automatic Evaluation

Task-takers can submit their answers to questions on our website. The answers must be in XML format (answer XML). The format of the answer XML is the same as the correct answer table. In answer XML, the indispensable tags in each answer that correspond to each minimal question are ‘answer’ and ‘anscolumn\_ID’. Logs in a solving process (process log) can be optionally described in the answer XML. The result has shown in the table:

Scores	English	Japanese	Japanese History	Math IA	Math IIB	Physics	World History
Task-takers’ system	52 (/200)	62 (/150*)	56 (/100)	57 (/100)	41 (/100)	39 (/100)	58 (/100)
Average of Students	88.3	72.2	45.6	52.0	47.6	42.0	46.6

Table 4.1: Results of the open evaluation of task-takers’ systems

#### 4.4 Dynamic Memory Networks (DMN) for NLP

We discuss about Dynamic Memory Network (DMN) [18], a neural network based framework for general question answering tasks that is trained using raw input-question-answer triplets. Generally, it can solve sequence tagging tasks, classification problems, sequence-to-sequence tasks and question answering tasks that require transitive reasoning. The DMN first computes a representation for all inputs and the question. The question representation then triggers an iterative attention process that searches the inputs and retrieves relevant facts. The DMN memory module then reasons over retrieved facts and provides a vector representation of all relevant information to an answer module which generates the answer.

Example inputs and questions, together with answers generated by a dynamic memory network trained on the corresponding task are shown below:

I: Jane went to the hallway.

I: Mary walked to the bathroom.

I: Sandra went to the garden.

I: Daniel went back to the garden.

I: Sandra took the milk there.

Q: Where is the milk?

A: garden

I: It started boring, but then it got interesting.

Q: What's the sentiment?

A: positive

Q: POS tags?

A: PRP VBD JJ , CC RB PRP VBD JJ .

DMN has four modules:

**Input Module:** The input module encodes raw text inputs from the task into distributed vector representations. They use a gated recurrent network (GRU). The internal mechanics of the GRU is defined as:

$$z_t = \sigma (W^{(z)} x_t + U^{(z)} h_{t-1} + b^{(z)}) \dots\dots\dots (1)$$

$$r_t = \sigma (W^{(r)} x_t + U^{(r)} h_{t-1} + b^{(r)}) \dots\dots\dots (2)$$

$$\bar{h}_t = \tanh (W x_t + r_t \circ U h_{t-1} + b^{(h)}) \dots\dots\dots (3)$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \bar{h}_t \dots\dots\dots (4)$$

where  $\circ$  is an element-wise product,  $W^{(z)}, W^{(r)}, W \in R^{n_H \times n_I}$  and  $U^{(z)}, U^{(r)}, U \in R^{n_H \times n_H}$ . The dimensions are hyperparameters. We abbreviate the above computation with  $h_t = \text{GRU}(h_t, h_{t-1})$ .

**Question Module:** Like the input module, the question module encodes the question of the task into a distributed vector representation. For example, in the case of question answering, the question may be a sentence such as Where did the author first fly?.

**Episodic Memory Module:** Given a collection of input representations, the episodic memory module chooses which parts of the inputs to focus on through the attention mechanism. It then produces a “memory” vector representation taking into account the question as well as the previous memory. Each iteration provides the module with newly relevant information about the input. In other words, the module has the ability to retrieve new information, in the form of input representations, which were thought to be irrelevant in previous iterations.



**Answer Module:** The answer module generates an answer from the final memory vector of the memory module. Depending on the type of task, the answer module is either triggered once at the end of the episodic memory or at each time step.

The network is like that:

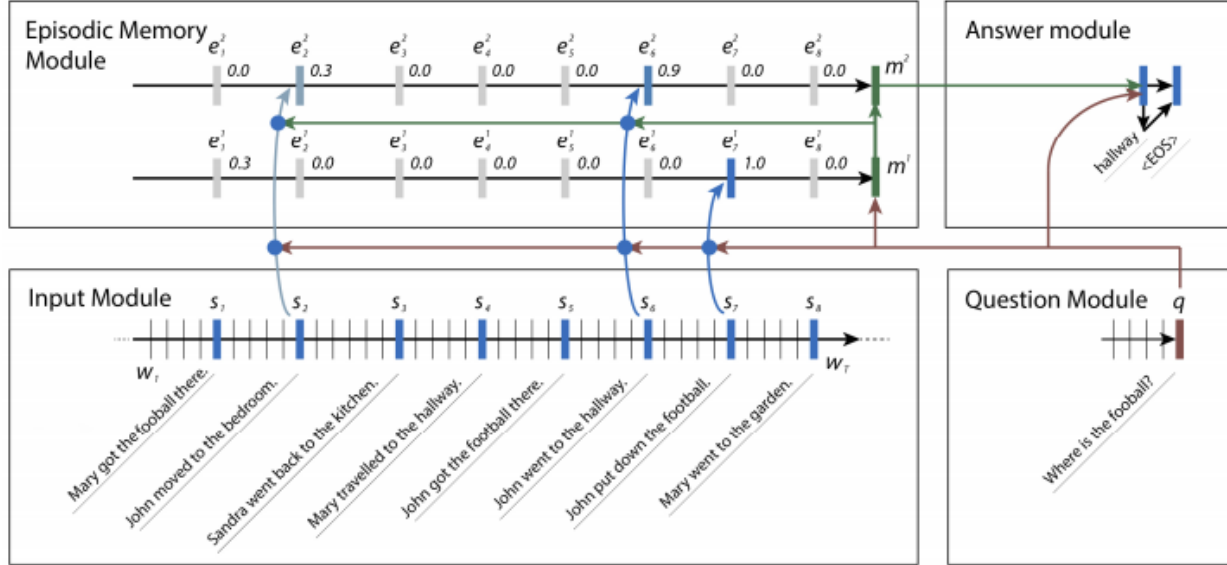


Figure 4.1: Real example of an input list of sentences and the attention gates those are triggered by a specific question from the bAbI tasks.

## Experiments

They include experiments on question answering, part-of speech tagging, and sentiment analysis. The model is trained independently for each problem, while the architecture remains the same except for the answer module and input fact subsampling (words vs sentences). The answer module, as described in Section 2.4, is triggered either once at the end or for each token. For all datasets we used either the official train, development, test splits or if no development set was defined, we used 10% of the training set for development. Hyper parameter tuning and model selection (with early stopping) is done on the development set. The DMN is trained via backpropagation.

## Results

They use Facebook bAbI dataset that is a synthetic dataset for testing a model’s ability to retrieve facts and reason over them. Each task tests a different skill that a question answering model ought to have, such as coreference resolution, deduction, and induction. Showing an ability exists here is not sufficient to conclude a model would also exhibit it on real world text data. It is, however, a necessary condition.

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

Table 4.2: Test accuracies on the bAbI dataset

### 4.5 DMN for Visual and Textual Question Answering

Neural network architectures with memory and attention mechanisms exhibit certain reasoning capabilities required for question answering. One such architecture, the dynamic memory network (DMN), obtained high accuracy on a variety of language tasks. However, it was not shown whether the architecture achieves strong results for question answering when supporting facts are not marked during training or whether it could be applied to other modalities such as

images. Based on an analysis of the DMN, they propose several improvements to its memory and input modules. Their new DMN+ model [19] improves the state of the art on both the Visual Question Answering dataset and the bAbI-10k text question-answering dataset without supporting fact supervision.

### Improved Dynamic Memory Networks: DMN+

They proposed and compared several modeling choices for two crucial components: input representation, attention mechanism and memory update.

#### Input Module for Text QA

They speculate that there are two main reasons for this performance disparity, all exacerbated by the removal of supporting facts. First, the GRU only allows sentences to have context from sentences before them, but not after them.

#### Input Fusion Layer

For the DMN+, they propose replacing this single GRU with two different components. The first component is a sentence reader, responsible only for encoding the words into a sentence embedding. The second component is the input fusion layer, allowing for interactions between sentences

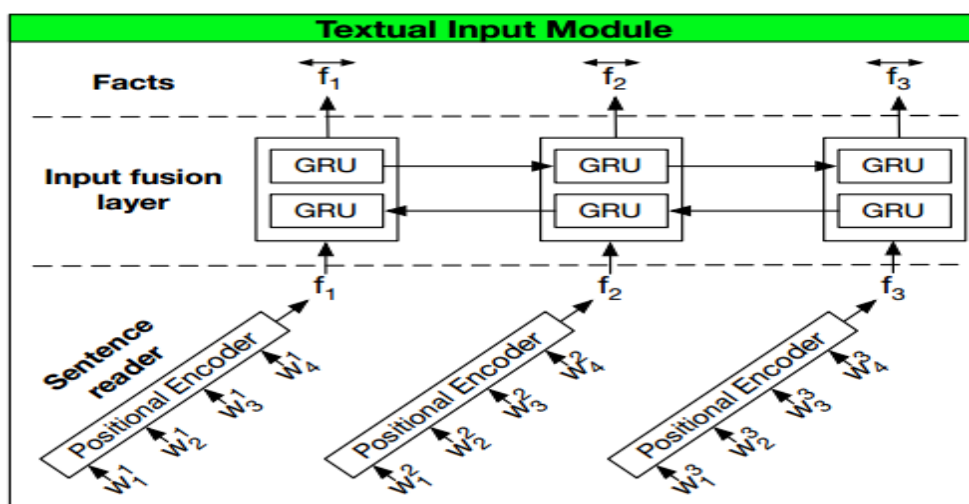


Figure 4.2: The input module with a “fusion layer”

## Experiments

To understand the impact of the proposed module changes, they analyze the performance of a variety of DMN models on textual and visual question answering datasets.

## Results

They compare their performing approach, DMN+, to two state of the art question answering architectures: the end to end memory network (E2E) [20]. Neither approach use supporting facts for training. they select the model from the paper that achieves the lowest mean error over the bAbI-10k dataset.

Task	DMN+	E2E	NR
2: 2 supporting facts	0.3	0.3	-
3: 3 supporting facts	1.1	2.1	-
5: 3 argument relations	0.5	0.8	-
6: yes/no questions	0.0	0.1	-
7: counting	2.4	2.0	-
8: lists/sets	0.0	0.9	-
9: simple negation	0.0	0.3	-
11: basic coreference	0.0	0.1	-
14: time reasoning	0.2	0.1	-
16: basic induction	45.3	51.8	-
17: positional reasoning	4.2	18.6	0.9
18: size reasoning	2.1	5.3	-
19: path finding	0.0	2.3	1.6
Mean error (%)	2.8	4.2	-
Failed tasks (err >5%)	1	3	-

Table 4.3: Test error rates of various model architectures on tasks from the bAbI English 10k dataset.

## 4.6 Query-Regression Networks

We present Query-Regression Network (QRN) [20], a variant of Recurrent Neural Network (RNN) that is suitable for end-to-end machine comprehension. QRN is a single recurrent unit with internal memory and local sigmoid attention. QRN is able to effectively handle long-term dependencies and is highly parallelizable. In our experiments we show that QRN obtains the state-of-the-art result in end-to-end bAbI QA tasks. Query-Regression Network (QRN), is a single recurrent unit that addresses the long-term dependency problem of most RNN-based models by simplifying the recurrent update, while taking the full advantage of RNN's capability to model sequential data (Figure 1). QRN considers the sentences (story) as a sequence of state-changing triggers, and QRN transforms (regresses) the original question (query) to an easier-to-answer query as it observes each trigger through time. For instance, consider the question-story pair in Figure 1b. The original question, "Where is the apple?" cannot be directly answered by any single sentence from the story. Hence, after observing the first sentence "Sandra got the apple there.", QRN transforms the original question to "Where is Sandra?", which is presumably easier to answer. This mechanism is akin to logic regression in situation calculus. The query regression in our model is performed locally, so it can better encode locality information.

## Conclusion

Various types of question answering techniques in NLP are discussed in this chapter. We find different types of procedure for answering the questions from a given dataset of natural language. Some papers discuss about neural networks algorithm for QA. DMN , DMN+ and QRN are based on neural network algorithm. At the beginning of this chapter we introduce to answer the question by syntactic structures and some reasons from the questions. We follow that method for our thesis work because it is less complex than neural network algorithm.

## CHAPTER 5

### Implementation

#### Introduction

Our system methodology and work flow are discussed briefly in this chapter. Our main focus is question answering from given text. We have been studied different procedures for question answering in NLP. Those techniques are helpful to design our new system. Our designed system is too easy to understand and less complex than others.

#### 5.1 Overview of System

We now give the overview of the system that makes up our QA techniques in NLP. Then each module would be tested by inputs in NLP. We have four modules in our system. The high illustration shown in the below:

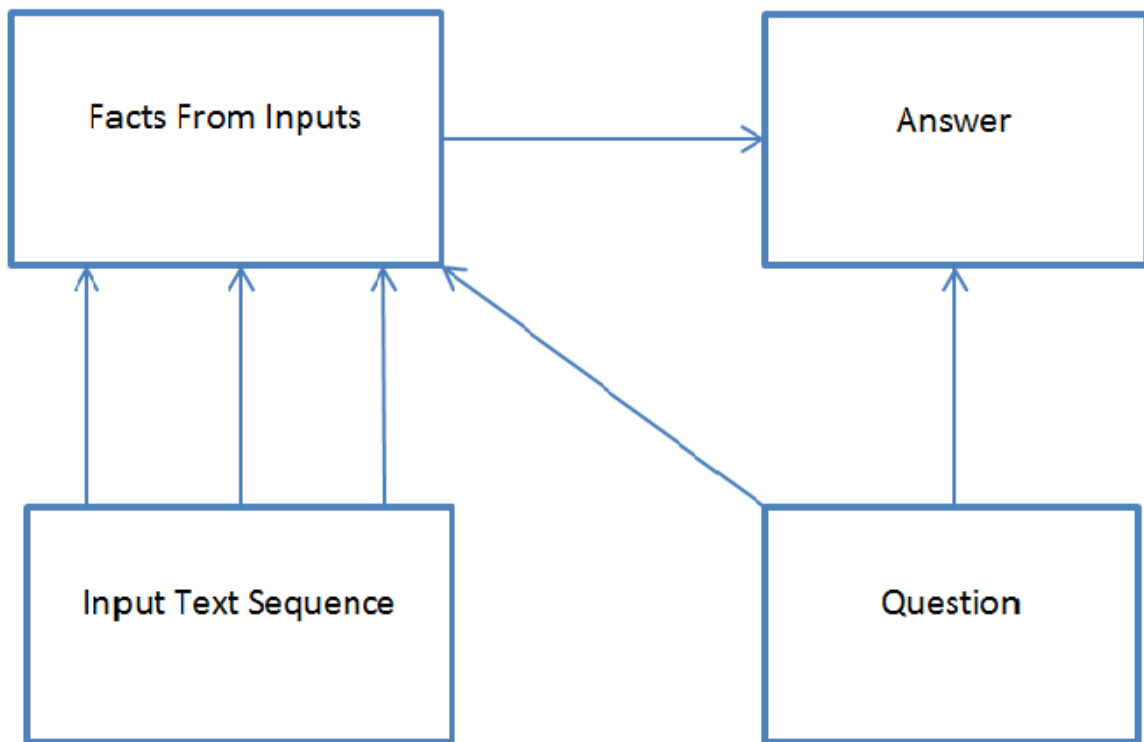


Figure 5.1: Overview of QA techniques

From figure 5.1 example of text answering method is given by us. In next figure 5.2, an example of our system has been given:

I: John travelled to the office.  
I: John discarded the apple there.  
I: Sandra left the milk.  
I: Daniel went to the bathroom.  
Q: Where is the apple?  
A: office  
I: Mary went to the bedroom.  
I: Mary moved to the office.  
I: John travelled to the hallway.  
I: Mary moved to the kitchen.  
Q: Was marry in the kitchen?  
A: Yes

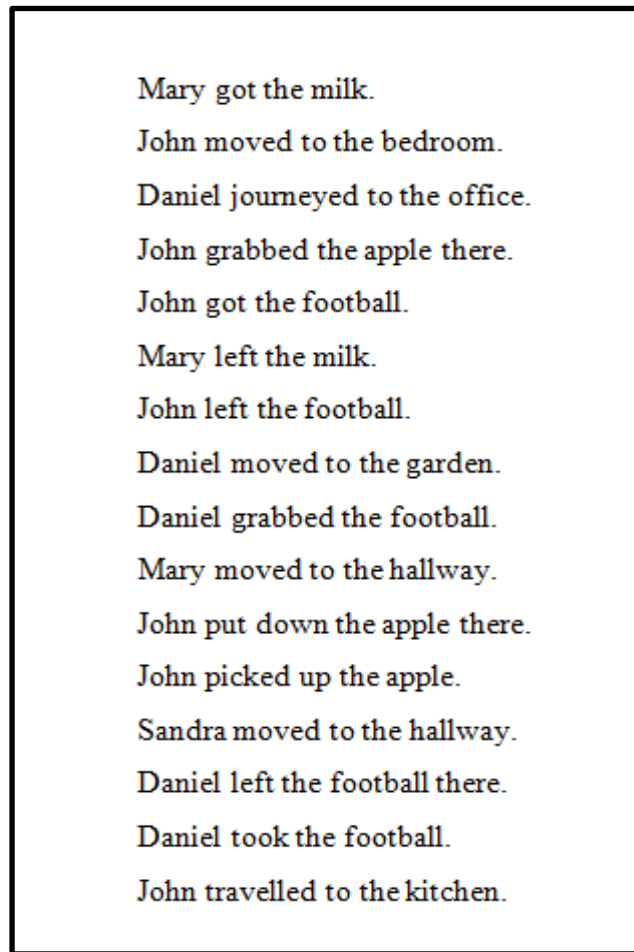
Figure 5.2: Examples of inputs and questions

In the figure 5.2, we examine a sequence of input text (indicated by I) and questions (indicated by Q) corresponding this text and then finding answers (indicated by A). Our main challenge is to find correct answers from the text. Text is a sequence of sentences. First question is “Where is the apple?”. We find that in the second sentence “John discarded the apple there”. And “John travelled to the office” is at the first sentence. So the last position of the apple before asks the question is “office”. In the next, we also find the answers for yes/no type questions.

## 5.2 Input Structure and Feature Extraction

The input is a sequence sentences  $S$  which each has words  $w_1, w_2, \dots, w_n$ , where  $n$  is the number

of words in a sentence. Here each sentence in the input text should have a sequential meaning. We give an example of the input sequence:



Mary got the milk.  
John moved to the bedroom.  
Daniel journeyed to the office.  
John grabbed the apple there.  
John got the football.  
Mary left the milk.  
John left the football.  
Daniel moved to the garden.  
Daniel grabbed the football.  
Mary moved to the hallway.  
John put down the apple there.  
John picked up the apple.  
Sandra moved to the hallway.  
Daniel left the football there.  
Daniel took the football.  
John travelled to the kitchen.

Figure 5.3: Example of inputs

For finding the main features to analyze the text we follow the processes:

### 5.2.1 Parse the input text

After giving of a sequence of text, we parse the sentences to partition each of them. Then we can pick each sentence from the text. We use NLTK based `sent_tokenize` in the Python language [5]. That works like:



Daniel grabbed the football. Mary moved to the hallway. John put down the apple there. John picked up the apple. Sandra moved to the hallway. Daniel left the football there. Daniel took the football.

Figure 5.4: Input text to tokenize the sentences

We separate the each sentence by using `nltk_sentokenize()` function and find the number of sentences. We see in figure 5.5:

```
Number of sentences: 7
The sentences are:

Daniel grabbed the football.
Mary moved to the hallway.
John put down the apple there.
John picked up the apple.
Sandra moved to the hallway.
Daniel left the football there.
Daniel took the football.
```

Figure 5.5: Parsed text to find sentences

### 5.2.2 Finding dependency structure from the text

To find out the semantic structures of text, it is needed to find dependency structures of every sentence to each other. So we firstly parse the each sentence by `word_tokenize()` function and tagged each word by nltk POS tagger [5]. Let an example of POS tagging by nltk POS tagger:

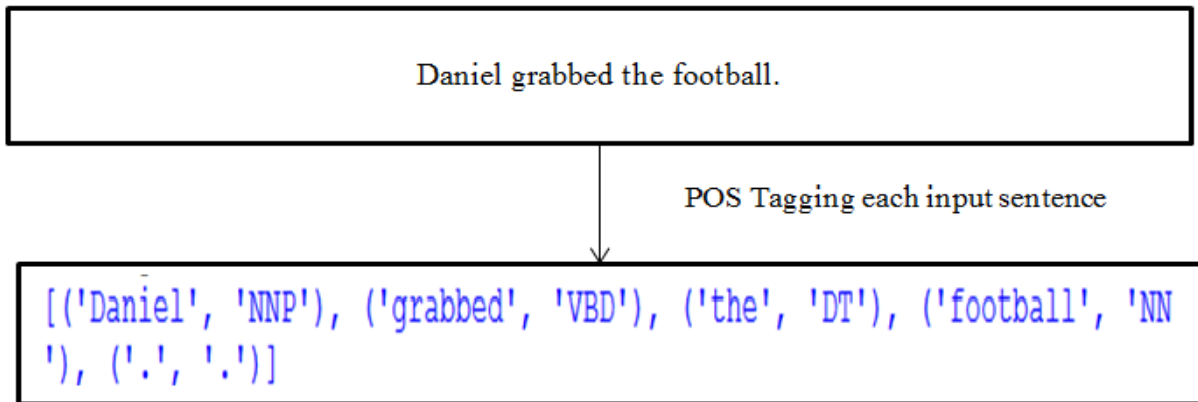


Figure 5.6: POS tagging of a sentence

In figure 5.6, we find each word and corresponding to its parts-of-speech. NLTK has better accuracy than other POS taggers.

### Semantic Parsing

After tagging each word to its corresponding part-of-speech, we need to find logical forms of each sentence. Then we can make relationship of each sentence with others. This method has introduced to the semantic parsing. We use lambda calculus parsing [14] method for dependency parsing. It also reduces our features for database from which we can retrieve answers.

### Lambda calculus parsing

In chapter 3 we discuss about lambda calculus parsing method. We also describe that method for our purpose of work. Firstly we can see a lambda calculus parsing of a sentence:

Root	Part1	Part2
Verb	Subject	Object

Figure 5.7: lambda calculus parsing for our work

Let take a sentence “He is a boy”. Its lambda calculus representation is:

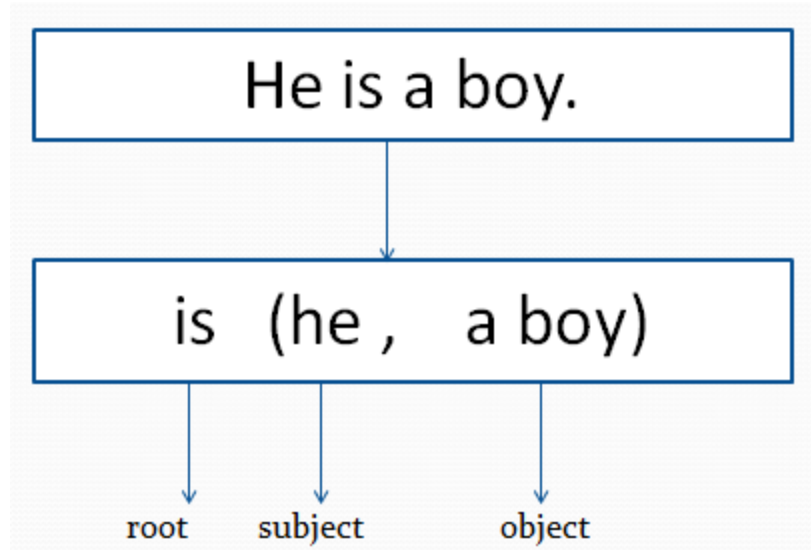


Figure 5.8: Lambda calculus representation of a sentence

By lambda calculus parsing method we find the dependency of a sentence. We see in the figure 5.8, verb is set as root and makes dependency with subject and object of the sentence.

### 5.2.3 Store the features in database

Our last step for processing input is to store the features. We find three things from each sentence. They are:

- i. **Verb**
- ii. **Subject**
- iii. **Object**

But there is a question- “How can we detect a word from a sentence as verb or subject or object?”. We find this from the POS tagging. Let to see the figure 5.6, the input sentence is “Daniel grabbed the football”. The tagged from is like:

Daniel is tagged to “NNP”, grabbed to “VBD” , the to “DT” and football to “NN”. Here the Daniel is selected as subject because it is noun. Then verb is also selected as to find “VB” in the POS tagging. Object of the sentence is also selected to like as subject where the word is POS tagged to ‘NN\_’ or ‘PR\_’.

We follow the figure 5.9 to realize how main features are extracted from text:

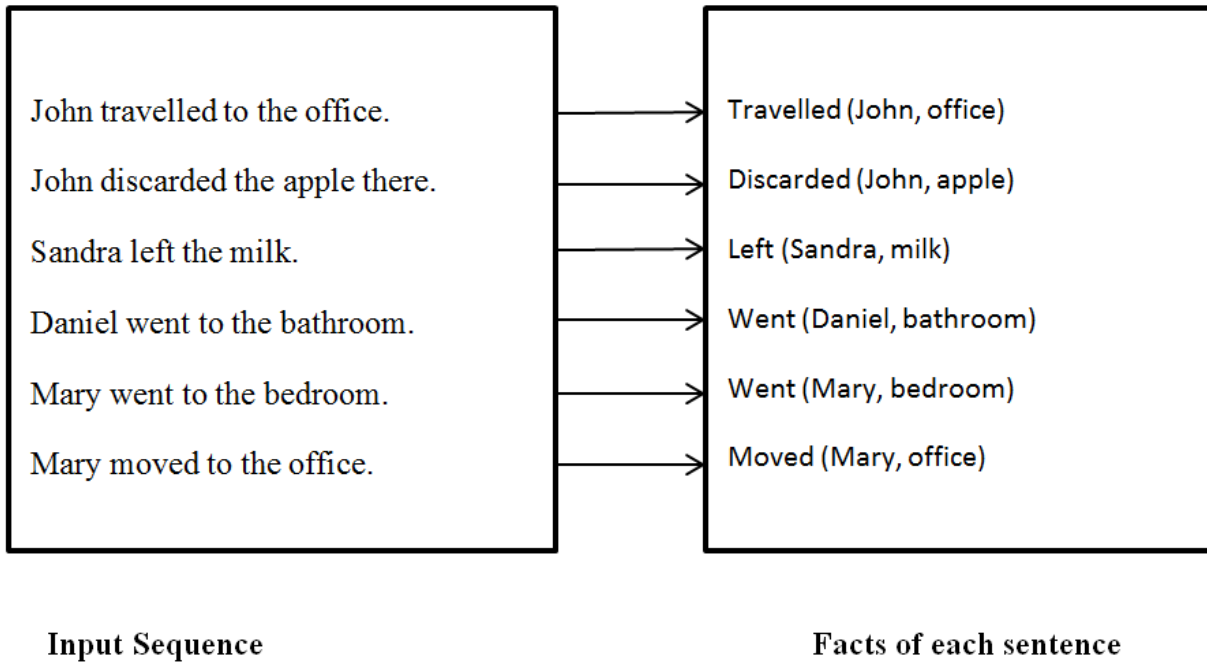


Figure 5.9: Finding facts from input sequence by lambda calculus parsing

Previously we discuss that we find three main features from each sentence. We store them to different databases. From the figure 5.8, we make the databases for verb, subject and object. They are following:

```
['travelled', 'discarded', 'left', 'went', 'went', 'moved']
```

Figure 5.10: Verb database

```
['John', 'John', 'Sandra', 'Daniel', 'Mary', 'Mary']
```

Figure 5.11: Subject database

```
['office', 'apple', 'milk', 'bathroom', 'bedroom', 'office']
```

Figure 5.12: Object database

We find three databases have same number of components. These verb, subject and object are corresponding to the sentence position of the text. So the first position of the verb, subject and object are in the first sentence. For that we can easily find that which features are from which sentences.

### **5.3 Question Analysis & feature extraction**

Similar to the input sequence, the question is also most commonly given as a sequence of words in natural language processing problems. The questions have been needed to be analyzed to find answers. Question must have a meaning to the corresponding text. The procedures followed for question analysis are described in the next.

#### **5.3.1 Parse the question**

We parse the question with POS tagging. Question can be different types. Like

- W/H question
- Yes/No question

We mainly work on these two types of question. **W/H** question starts with ‘WR’ tagged word. There are different types of ‘WR’ tagged word like:

- How
- What
- Where
- Which
- Whom
- When
- Whose

**Yes/ No** question generally start with

- Am
- Is
- Are

- Was
- Were
- Do
- Does
- Did
- Etc.

After parsing and tagging each word of a sentence we find three main features from that:

- 'WR' word
- Verb
- Object

Sometimes we need to find more features than that. Like question has two parts with 'before/after' word in the question. For that reason then we need to find two object words form the question.

If we give a question "Where is Mary ?" and try to find out the main features from it , we tagged each of the word and find the common features. From the question W/H word is "Where" and object word is "Marry", it is shown in figure 5.13.

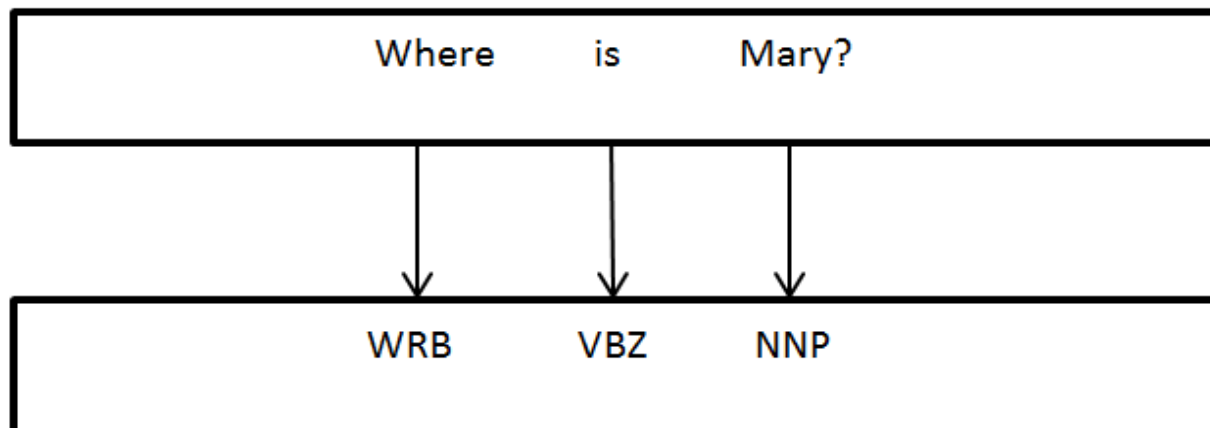


Figure 5.13: Parse the question to extract features by POS tagging

Here question word is set to 'Where' and object word is set to 'Mary'. The word 'where' means that to find out the location of object word 'Mary'. Another features verb is 'is' which means the be verb.

Another type of W/H question is with ‘before/after’ word. We discuss this with a figure:

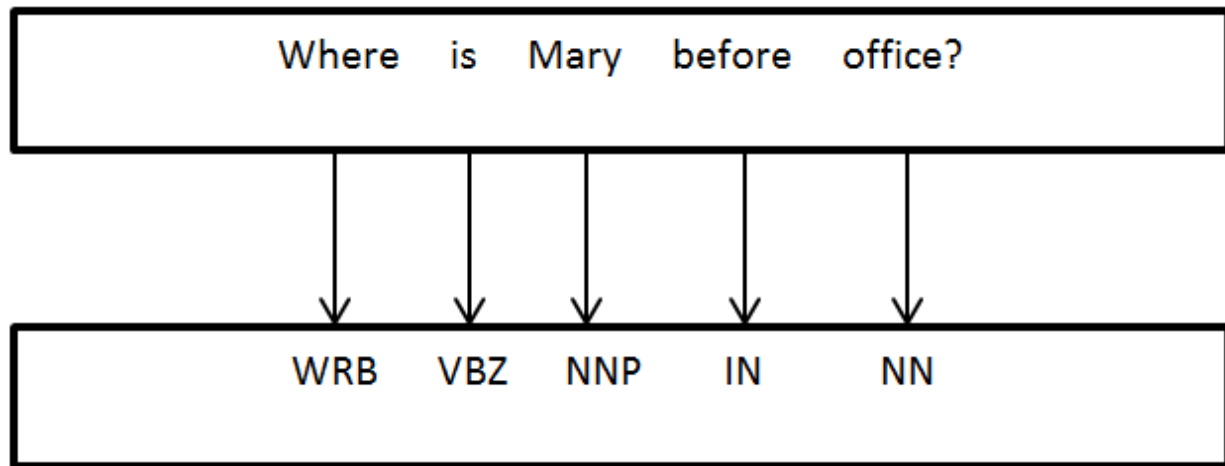


Figure 5.14: W/H question with ‘before/after’ word and POS tagging

For this type question in figure 5.14 object words are ‘Mary’ and ‘office’.

Object\_1= ‘Mary’

Object\_2=‘office’

We introduce another type of question is Yes/No question. We see the structure and features of Yes/No question in figure 5.15:

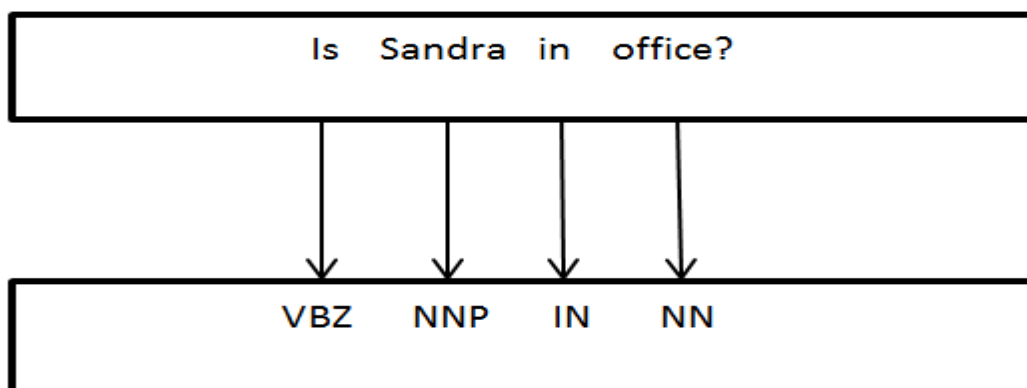


Figure 5.15: Yes/No question structure and POS tagging

In yes/no question, two object features are extracted. For question in figure 5.15 the object words are:

Object\_1= Sandra

Object\_2= office

We already discuss the question types and how to extract main features from that for answering. The extracting features are important for analyzing the questions.

## **5.4 Answering Module**

In the previous lessons in this chapter, we discuss the procedures of analyzing input text and questions and also find the features from them. Now we introduce our proposed system for question answering. To find the answer easily we make databases from input text. These databases are the main sources to find out the answers. But for finding answers, we need to know the semantic meaning of the question. For that we find the main features from the questions. From the corresponding features we focus to find out the questions from databases. We also discuss that question can be many types. We introduce two types of questions. In the next we discuss our method to find the answers from those questions.

### **5.4.1 W/H Question Answering**

W/H question can be various types. We discuss about the question that start with ‘where’ word:

#### **Question with ‘where’**

The question that starts with ‘where’ means that answer will be the place to its corresponding object. We give an example “Where is Mary?”. To find the answer of that question, we find the place of object word ‘Mary’.

Now in the input text that’s features are stored in databases, we need to find out which sentences means the location/place of that subject. To find the location we considered the verb of that sentence. For that we can only consider the verb database of input text that is processed previously. To find the location/place we define some verb that indicates position of subject. Like in the figure 5.16:



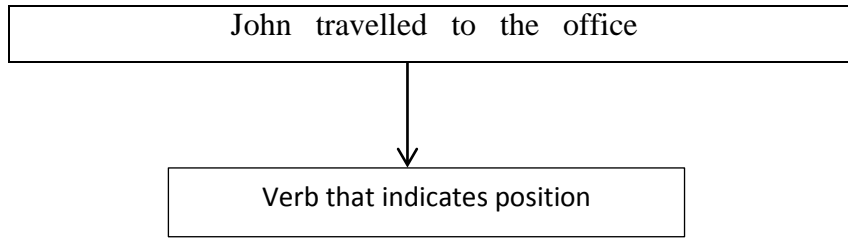


Figure 5.16: Locational verb in a sentence

We make a list of some positional verbs those are saved in database. The positional verbs those are listed by us are:

`['moved', 'go', 'went', 'gone', 'stay', 'travelled', 'journeyed']`

Figure 5.17: List of positional verb

Let these verb is in PV and our verb database contents are in V then we find the matching of them by following algorithm:

1. For  $i$  is 0 to length of verb database
2. If  $V_i$  is in  $PV_j$  then set  $T_i = 1$ , otherwise  $T_i = 0$  [ where  $0 \leq j \leq \text{len}(PV)$  ]

Now we can observe the  $T_i$  array for the input sequence in figure 5.9 and verb database in figure 5.10:

`['1', '0', '0', '1', '1', '1']`

Figure 5.18: Finding positional verb

We see in figure 5.18, the position is '1' where positional verb exists. That means that the object of that position indicates location. The object word from the question is also found previously. Let to take object word of question in OBW, subject database in SB and object database in OB

Now answering algorithm is:

1. For  $i = \text{length\_of\_input\_text}$  to 0
2. If OBW is in  $SB_i$  and  $T_i = 1$  ,  
Then answer =  $OB_i$
3. else if OBW is in  $OB_i$  and Subject\_OBW= $SB_i$  and  $T_i = 1$  ,  
Then answer =  $OB_i$

The flowchart is:

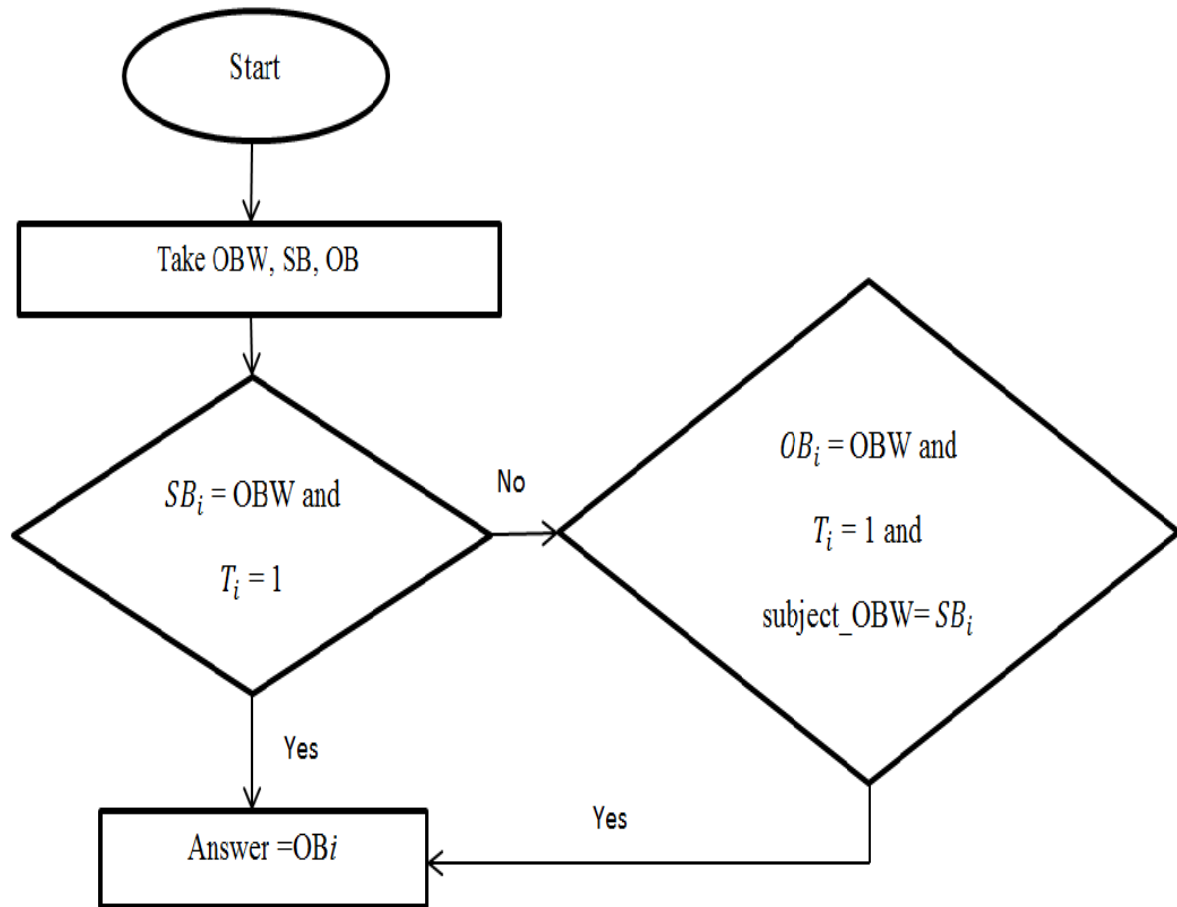


Figure 5.19: flowchart of 'where' question answering

Now if we give the question “Where is the Mary?” the answer would be in the below:

```
Question is : ['Where', 'is', 'the', 'Mary']
Where ..... WRB
is ..... VBZ
the ..... DT
Mary ..... NNP
Where
Question is ok
Noun is: ['Mary']
Questions features are : ['Mary'] []
1
Subj 5
Yes 5
Answer is: office
```

Figure 5.20: Question answering of finding position

In figure 5.19, this output works upon the previously defined algorithm in figure 5.19. It finds out the last position of ‘Mary’ by place finding. So by this algorithm we find the correct answer.

### Question with ‘where’ included ‘before/after’

Let a question is like “Where is the Mary before office?”. We consider here two things.

- OBW\_1= Mary
- OBW\_2=office

For that we need to find the location of OBW\_1 before at OBW\_2. So our defined algorithm is as like previous but changed simply:

1. For  $k = \text{length\_of\_input\_text}$  to 0
  - i. If OBW\_1 is in  $OB_k$  and OBW\_2 is in  $SB_k$   
For  $i = \text{position\_of\_subject\_OBW\_1}$  to 0  
If  $\text{subject\_OBW\_1} = SB_i$  and  $T_i = 1$  ,  
  
Then answer =  $OB_i$

- ii. If OBW\_1 is in  $OB_k$  and OBW\_2 is in  $OB_k$   
For i=position\_OBW\_2 to 0  
If subject\_OBW\_1 =  $SB_i$  and  $T_i=1$  ,  
Then answer =  $OB_i$
- iii. If OBW\_1 is in  $SB_k$  and OBW\_2 is in  $OB_k$   
For i=position\_OBW\_2 to 0  
If subject\_OBW\_1 =  $SB_i$  and  $T_i = 1$  ,  
Then answer =  $OB_i$
- iv. If OBW\_1 is in  $SB_k$  and OBW\_2 is in  $SB_k$   
For i=position\_OBW\_2 to 0  
If subject\_OBW\_1 =  $SB_i$  and  $T_i = 1$  ,  
Then answer =  $OB_i$

2. Else answer is not found

Form this algorithm we can give a question "Where is the Mary before office?". The answer is "bedroom" because the position of Mary before office is in "bedroom". We see in the figure 5.20 of the answer of that question:

```

Question is : ['Where', 'is', 'the', 'Mary', 'before', 'office']
Where ..... WRB
is ..... VBZ
the ..... DT
Mary ..... NNP
before ..... IN
office ..... NN
Where
Question is ok
Noun is: ['Mary']
Noun is: ['Mary', 'office']
Questions features are : ['Mary', 'office'] ['before']
2
Yap, 2 input subject
Subj 5
['sbj_1', 'Obj_2']
Answer is: 4 th position:  bedroom

```

Figure 5.21: Answering the question with 'where' included 'before'

### 5.4.2 YES/NO Question Answering

Yes/No question answering is a simple task than W/H question answering. It provides the true or false answers to the corresponding question. Previously we discuss the types of Yes/No questions. We design an algorithm for Yes/No question answering for place finding. For that we need the array  $T_i$  that is previously defined. Let a question is “Is Mary in the hallway?”. Two features are extracted from there. In  $OBW\_1$ =Mary and  $OBW\_2$ =hallway. Then we search the matching for  $OBW\_1$  and  $OBW\_2$  in the same sentence. If it is occurred in same sentences the answer will be ‘Yes’, otherwise ‘No’. The algorithm is:

1. For  $i$ =length\_of\_input\_text to 0
2. If  $OBW\_1$  in  $SB_i$  and  $OBW\_2$  in  $OB_i$  and  $T_i = 1$   
Then answer is ‘Yes’
3. Else answer is ‘False’

We can see the flowchart of this Yes/No question answering algorithm.

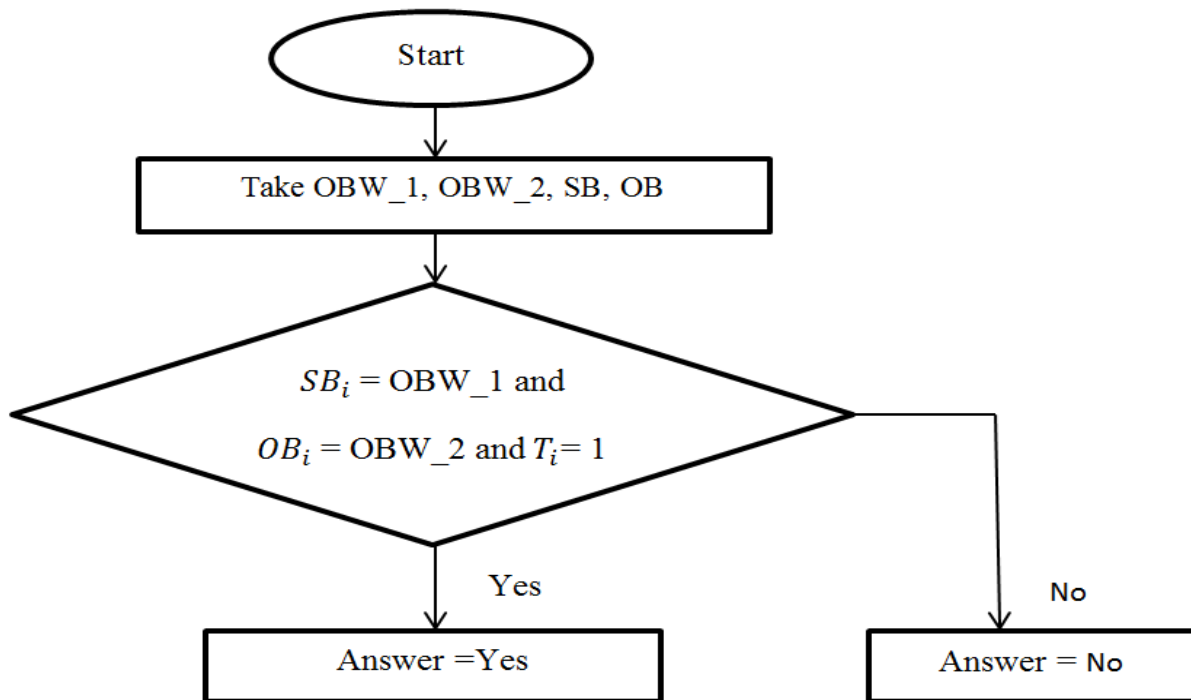


Figure 5.22: flowchart of Yes/No question answering

Now we test the system by giving a question “Is Mary in Hallway?” and see the result in figure 5.23:

```
Question is : ['Is', 'Mary', 'in', 'the', 'hallway']
Is ..... VBZ
Mary ..... NNP
in ..... IN
the ..... DT
hallway ..... NN
Is
Question is ok
['Mary', 'hallway']
Subj 5
Answer is: No
```

Figure 5.23: ‘Yes/No’ question answering implementation

We can observe in our input text that there is no position for Mary where she is in the hallway. So following our algorithm the answer is ‘No’ and this is correct.

## Conclusion

In this chapter, we describe in details about question answering techniques. Firstly we give the overview of our system. We introduce about parsing method, features extraction and saving in database for the input text. At final stage we discuss about our question answering algorithm for both W/H question and Yes/No question. Our algorithm is easy than other QA algorithms and anyone can easily use this algorithm.

## CHAPTER 6

### Experimental Result Analysis

#### Introduction

To analyze our proposed model, textual question answering datasets have been used. We find the accuracy and error rate of answering the questions.

#### 6.1 Experimental Datasets

To analyze our proposed model, we use bAbI-10k English [19], a synthetic dataset which features 20 different tasks. The dataset comes in two sizes, referring to the number of training examples each task has: bAbI-1k and bAbI-10k. We test our system by 5 different tasks of bAbI-10k.

#### 6.2 Working Environment

Our working language is Python because it is object oriented language and flexible for text processing. We use Python version of 3.4.1. We also use NLTK module in Python. NLTK gives support for word\_tokenize, sent\_tokenize, POS\_tagging.

#### 6.3 Parsing Accuracy

We observe the parsing accuracy to find the accurate verb, subject and object of each sentence for specific tasks. The output is shown in the table 6.1:

Task	No. of Sentences	Verb	Subject	Object
1. Single Supporting Facts	13	100%	100%	77%
2. Two Supporting Facts	25	100%	100%	92%
3.Three Supporting Facts	37	100%	100%	95%
4. Yes/No questions	40	100%	100%	96%
5. List/Sets	60	100%	100%	97%
Mean Accuracy (%)		100%	100%	94.28%

Table 6.1: Parsing accuracy of sentences

We observe that the system gives better performance for parsing than previously we studied. Accuracy of parsing for different input tasks are shown in table 6.1. Sometime it falls

performance for false POS tagging. We show the graph in figure 6.1 for parsing accuracy of our system.

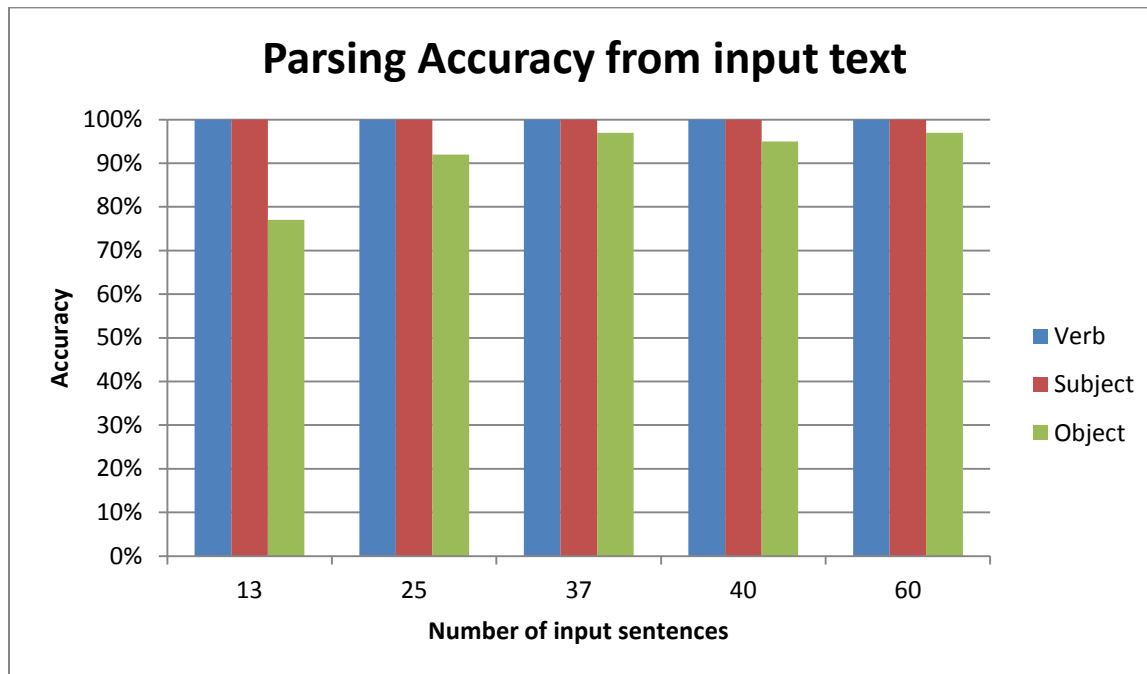


Figure 6.1: Graphical representation of parsing accuracy for input sentences

## 6.4 Question Answering Results

Now we test our system by given questionnaires to the specific task. We can calculate accuracy by the:

Task	No of sentences	No of questions	System accuracy
1. Single Supporting Facts (SSF)	13	10	90%
2. Two Supporting Facts (TSF)	25	10	90%
3. Three Supporting Facts (ThSF)	37	10	80%
4. Yes/No questions (Y/NQ)	40	20	95%
5. List/Sets (L/S)	60	10	60%
Mean Accuracy (%)			83%

Table 6.2: Accuracy of QA for different tasks



In table 6.2, we observe that the accuracy degrades for adding new task as an input of the system. It gives high performance for first four types of task but does not give better performance for the task List/Sets. The main reason is that our algorithm gives high performance for first four numbers of tasks. We show graphically in figure 6.2 for the accuracy of QA:

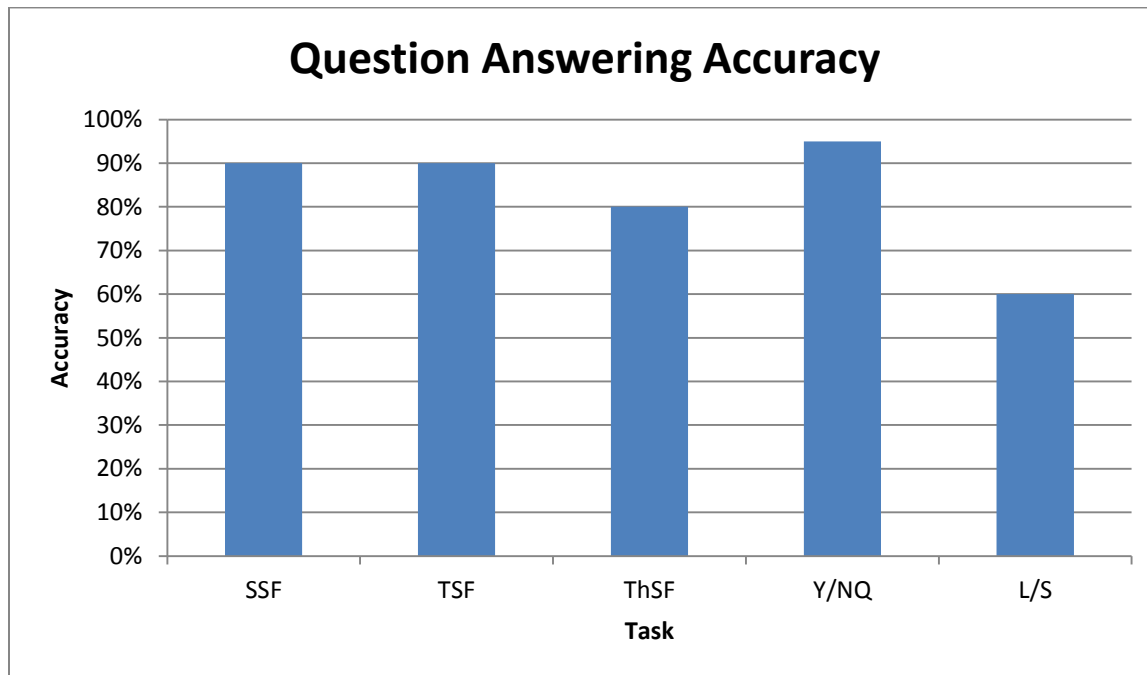


Figure 6.2: Graphical representation of QA accuracy for different tasks

## 6.5 Result Analysis

We can easily understand from the table 6.1 about the parsing accuracy. It gives very high performance for our testing input. In table 6.2, we find the question answering accuracy for different task. The accuracy degrades for adding new type of input task. But we can say that our system performance is very good for question answering. It gives the mean accuracy of 83%.

## Conclusion

In this chapter we show the output results of parsing and question answering and find out the accuracy from them. We show that our system gives very high performance for those tasks and we can easily find correct answers form the system.

## **CHAPTER 7**

### **Conclusions**

#### **7.1 Summary of our work**

In this paper, we have introduced a system for question answering techniques in NLP. A model is given that has been described overall the system. We used the lambda calculus parsing method for finding logical forms from a sentence. The semantic structure of the sentence is found by that parsing. Then we have introduced our question answering algorithm. The steps of algorithm are described briefly in the previous chapters. We test our system by five different types of tasks in QA dataset. Our tested dataset are contained of simple sentences. Our model gives a better accuracy than other existing system for simple sentences and it is relatively easy to use. The results of testing accuracy are shown in graphical representation in chapter 6.

#### **7.2 Future Work**

In the next, we try to explore our system. Here we have introduced the algorithm for simple sentence domain. In future we will have worked following:

**i. Testing by more questionnaires**

We will design the system that can be given more accuracy for more number of questionnaires.

**ii. Implement the system for answering questions from complex and compound sentences of text**

Our current system works only for the text containing of simple sentences. We will make our system that can work for all type of sentences.

**iii. Sentiment analysis of text**

Sentiment analysis means the positive or negative or neutral expression of the sentence. We will make our system to analyze the sentiment of the sentence.

**iv. System learning deeply and getting closer to strong AI**

Our main focus is to learn the system deeply and to get closer to strong AI (Artificial intelligence).

## **Conclusion**

A new model for question answering technique in NLP is proposed by us. We show the POS tagging and semantic parsing technique. Then we discuss about finding answers from a given text. Our system is easy and less complex than any others for input of simple sentences in question answering.

## REFERENCES

- [1] Liddy, Elizabeth D. "Natural language processing." (2001).
- [2] Chowdhury, Gobinda G. "Natural language processing." *Annual review of information science and technology* 37.1 (2003): 51-89.
- [3] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug), 2493-2537.
- [4] Mohanty, Rajat Kumar. "Parts-of-Speech Tagging." *2nd Asian Regional Training on Local Language Computing* (2005).
- [5] Loper, Edward, and Steven Bird. "NLTK: The natural language toolkit." *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*. Association for Computational Linguistics, 2002.
- [6] Hepple, Mark. "Independence and commitment: Assumptions for rapid training and execution of rule-based POS taggers." *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2000.
- [7] Tjong Kim Sang, Erik F., and Sabine Buchholz. "Introduction to the CoNLL-2000 shared task: Chunking." *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*. Association for Computational Linguistics, 2000.
- [8] Tjong Kim Sang, Erik F., and Sabine Buchholz. "Introduction to the CoNLL-2000 shared task: Chunking." *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*. Association for Computational Linguistics, 2000.
- [9] Nothman, Joel, et al. "Learning multilingual named entity recognition from Wikipedia." *Artificial Intelligence* 194 (2013): 151-175.
- [10] Täckström, Oscar, Kuzman Ganchev, and Dipanjan Das. "Efficient inference and structured learning for semantic role labeling." *Transactions of the Association for Computational Linguistics* 3 (2015): 29-41.
- [11] Varile, Giovanni Battista, and Antonio Zampolli. *Survey of the state of the art in human language technology*. Vol. 13. Cambridge University Press, 1997.
- [12] Kaur, Parneet, Vandana Pushe, and Rupinderdeep Kaur. "A Review of NLP Based Upon Semantic Analysis." (2014).
- [13] Shi, Lei, and Rada Mihalcea. "Open text semantic parsing using FrameNet and WordNet." *Demonstration Papers at HLT-NAACL 2004*. Association for Computational Linguistics, 2004.

- [14] Reddy, Siva, et al. "Transforming Dependency Structures to Logical Forms for Semantic Parsing." *Transactions of the Association for Computational Linguistics* 4 (2016): 127-140.
- [15] Wang, Mengqiu. "A survey of answer extraction techniques in factoid question answering." *Computational Linguistics* 1.1 (2006).
- [16] Li, Xiaoyan. "Syntactic features in question answering." *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. ACM, 2003.
- [17] Fujita, Akira, et al. "Overview of Todai Robot Project and Evaluation Framework of its NLP-based Problem Solving." *World History* 36 (2014): 36.
- [18] Kumar, Ankit, et al. "Ask me anything: Dynamic memory networks for natural language processing." *arXiv preprint arXiv:1506.07285* (2015).
- [19] Xiong, Caiming, Stephen Merity, and Richard Socher. "Dynamic memory networks for visual and textual question answering." *arXiv preprint arXiv:1603.01417* (2016).
- [20] Sukhbaatar, Sainbayar, Jason Weston, and Rob Fergus. "End-to-end memory networks." *Advances in neural information processing systems*. 2015. [] Bansal, Mohit. "Surface Web Semantics for Structured Natural Language Processing." (2015).
- [21] Seo, Minjoon, Hannaneh Hajishirzi, and Ali Farhadi. "Query-Regression Networks for Machine Comprehension." *arXiv preprint arXiv:1606.04582*(2016).