DESIGN AND DEVELOPMENT OF A NAMED ENTITY BASED QUESTION ANSWERING SYSTEM FOR MALAYALAM LANGUAGE

Thesis submitted by
Ms. Bindu.M.S
in fulfilment of the requirements for
the award of the degree of

DOCTOR OF PHILOSOPHY
under the
Faculty of Technology

DEPT. OF COMPUTER SCIENCE
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY
KOCHI 682022
2012
Certificate

This is to certify that the thesis entitled “DESIGN AND DEVELOPMENT OF A NAMED ENTITY BASED QUESTION ANSWERING SYSTEM FOR MALAYALAM LANGUAGE” submitted to Cochin University of Science and Technology, in partial fulfilment of the requirements for the award of the Degree of Doctor of Philosophy in Computer Science is a record of original research work done by Mrs. Bindu.M.S (REG NO:3705), during the period (2005-2012) of her study in the Department of Computer Science at Cochin University of Science and Technology, under my supervision and guidance and the work has not formed the basis for the award of any Degree or Diploma.

Signature of the Guide

Kochi- 22
Date:
I, Bindu.M.S hereby declare that the thesis entitled "DESIGN AND DEVELOPMENT OF A NAMED ENTITY BASED QUESTION ANSWERING SYSTEM FOR MALAYALAM LANGUAGE" submitted to Cochin University of Science and Technology, in partial fulfilment of the requirements for the award of the Degree of Doctor of Philosophy in Computer Science is a record of original and independent research work done by me during the period 2005-2012 under the supervision of Dr. Sumam Mary Idicula, Professor, Department of Computer Science, Cochin University of Science and Technology, and it has not formed the basis for the award of any Degree or Diploma.

Signature of the Candidate

Kochi-22
Date:
Dedicated To
My Lord and Saviour
Jesus Christ
Acknowledgement

This thesis would not have been possible without the help and support of many people:

First of all I thank God for strengthening me physically and mentally without stumbling before the difficulties I faced during this PhD work.

I express my sincere gratitude and indebtedness to my guide Dr.Sumam Mary Idicula, Professor, Department of Computer Science, Cochin University of Science and Technology for her valuable advice and support throughout.

I would like to express my heartfelt gratitude to Dr. K. Paulose Jacob, Head, Department of Computer Science, Cochin University of Science and Technology for providing me all help and facilities from the department.

I want to thank all the staff members of Computer Science Department who have facilitated me to complete this work.

I am highly obliged to all my friends and all those who have prayed for me.

Finally I would like to thank my husband and children for bearing my difficulties, problems and burdens for the last six years.

Bindu.M.S
ABSTRACT

This is a Named Entity Based Question Answering System for Malayalam Language. Although a vast amount of information is available today in digital form, no effective information access mechanism exists to provide humans with convenient information access. Information Retrieval and Question Answering systems are the two mechanisms available now for information access. Information systems typically return a long list of documents in response to a user's query which are to be skimmed by the user to determine whether they contain an answer. But a Question Answering System allows the user to state his/her information need as a natural language question and receives most appropriate answer in a word or a sentence or a paragraph.

This system is based on Named Entity Tagging and Question Classification. Document tagging extracts useful information from the documents which will be used in finding the answer to the question. Question Classification extracts useful information from the question to determine the type of the question and the way in which the question is to be answered. Various Machine Learning methods are used to tag the documents. Rule-Based Approach is used for Question Classification.

Malayalam belongs to the Dravidian family of languages and is one of the four major languages of this family. It is one of the 22 Scheduled Languages of India with official language status in the state of Kerala. It is spoken by 40 million people. Malayalam is a morphologically rich agglutinative language and relatively of free word order. Also Malayalam has a productive morphology that allows the creation of complex words which are often highly ambiguous.

Document tagging tools such as Parts-of-Speech Tagger, Phrase Chunker, Named Entity Tagger, and Compound Word Splitter are developed as a part of this research work. No such tools were available for Malayalam language.
System Principles, and Support Vector Machines are the techniques used for the design of these document preprocessing tools.

This research work describes how the Named Entity is used to represent the documents. Single sentence questions are used to test the system. Overall Precision and Recall obtained are 88.5% and 85.9% respectively. This work can be extended in several directions. The coverage of non-factoid questions can be increased and also it can be extended to include open domain applications. Reference Resolution and Word Sense Disambiguation techniques are suggested as the future enhancements.
# Contents

<table>
<thead>
<tr>
<th>Chapter No</th>
<th>Title</th>
<th>Page No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Introduction ....................................................................</td>
<td>1-8</td>
</tr>
<tr>
<td>1.1</td>
<td>What is Question Answering?</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>General Architecture of a Question Answering System</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Motivation and Scope</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Objectives</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>Road Map</td>
<td>7</td>
</tr>
<tr>
<td>1.6</td>
<td>Chapter Summary</td>
<td>8</td>
</tr>
<tr>
<td>2.</td>
<td>Literature Survey</td>
<td>9-25</td>
</tr>
<tr>
<td>2.1</td>
<td>Approaches to Question Answering Systems</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>History of Question Answering Systems</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>Modern Question Answering Systems</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>QA Systems for Indian languages</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Named Entity based QA Systems</td>
<td>22</td>
</tr>
<tr>
<td>2.6</td>
<td>Chapter Summary</td>
<td>25</td>
</tr>
<tr>
<td>3.</td>
<td>Overview of Malayalam Language</td>
<td>27-41</td>
</tr>
<tr>
<td>3.1</td>
<td>Basic Word Types</td>
<td>28</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Nouns</td>
<td>29</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Pronouns</td>
<td>30</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Verbs</td>
<td>31</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Qualifiers</td>
<td>32</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Dhvyadhakam</td>
<td>32</td>
</tr>
<tr>
<td>3.1.6</td>
<td>Affixes</td>
<td>32</td>
</tr>
<tr>
<td>3.2</td>
<td>Phrase Types</td>
<td>36</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Noun Phrase</td>
<td>36</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Verb Phrase</td>
<td>37</td>
</tr>
</tbody>
</table>
3.2.3 Adverbial Phrase ................................................................. 37
3.2.4 Adjectival Phrase ............................................................... 37
3.2.5 Postpositional Phrase ......................................................... 38
3.3 Malayalam Sentences ............................................................ 38
  3.3.1 Sentence Classification Based on Behaviour ...................... 38
  3.3.2 Sentence Classification Based on Construction ............... 39
3.4 Malayalam Sandhi ................................................................. 40
3.5 Chapter Summary ................................................................. 41

4. MaQAS: A Malayalam Question Answering System ............ 43-66
  4.1 System Architecture ............................................................. 43
    4.1.1 Indexing Module ........................................................... 45
    4.1.2 Question Analysis Module ............................................ 50
    4.1.3 Answer Extraction Module ......................................... 62
  4.2 Chapter Summary ................................................................. 66

5. Compound Word Splitter ......................................................... 67-85
  5.1 Malayalam Compound Word ............................................... 68
  5.2 Methods for Compound Word Splitting ............................... 69
    5.2.1 Most Probable Word Technique ................................. 69
    5.2.2 N-Gram Technique .................................................... 70
    5.2.3 Longest Match Technique .......................................... 71
    5.2.4 Baseline Technique ................................................... 71
    5.2.5 Finite State Transducer ............................................. 72
    5.2.6 Ad Hoc Rules ............................................................ 73
    5.2.7 Hybrid Method ......................................................... 74
    5.2.8 Weighted FST .......................................................... 75
  5.3 Compound Word Analyzer for Indian Language .................. 78
  5.4 Description of Malayalam Compound Word Splitter .......... 82
  5.5 Performance Evaluation ..................................................... 90
  5.6 Chapter Summary ................................................................. 90
6. **Part-of-Speech Tagger** ................................................................. 91-119
   6.1 Related Work ............................................................................... 92
   6.2 Malayalam POS Tagging ................................................................. 98
      6.2.1 POS Tag Set for Malayalam ..................................................... 99
      6.2.2 POS Tagger for Malayalam .................................................... 104
   6.3 Results and Discussions ................................................................ 116
   6.4 Chapter Summary ......................................................................... 119

7. **Phrase Chunker** ................................................................. 121-135
   7.1 Related Work ............................................................................... 122
   7.2 Malayalam Phrase Chunker ............................................................. 126
      7.2.1 Clause Identifier ...................................................................... 126
      7.2.2 Phrase Separator ..................................................................... 127
      7.2.3 Phrase Tagger .......................................................................... 129
   7.3 Performance Evaluation ................................................................. 133
   7.4 Chapter Summary ......................................................................... 135

8. **Named Entity Tagger** ............................................................ 137-154
   8.1 Related Work ............................................................................... 138
   8.2 Difficulties in Finding Named Entities in Malayalam Language ....... 144
   8.3 Methodology- Support Vector Machines ........................................ 144
   8.4 Malayalam NE Tagger .................................................................... 146
      8.4.1 NE Marker ................................................................................ 147
      8.4.2 NE Identifier ............................................................................ 147
      8.4.3 NE Classifier ............................................................................. 148
      8.4.4 NE Disambiguator ..................................................................... 151
   8.5 Performance Evaluation ................................................................. 151
   8.6 Chapter Summary ......................................................................... 153

9. **Performance Evaluation** ....................................................... 155-167
   9.1 General Methods of Evaluation .................................................... 156
Contents

9.2 Evaluation Metrics .......................................................................................... 157
9.3 MaQAS-Implementation .............................................................................. 159
9.4 Performance Evaluation of MaQAS ............................................................. 162
9.5 Analysis and Discussion of Results ............................................................. 163
9.6 Chapter Summary ......................................................................................... 167

10. Conclusion and Future Work .................................................. 169-172
10.1 Contributions .............................................................................................. 170
10.2 Future Work ................................................................................................. 171

References .......................................................................................................... 173-197

List of Publications .............................................................................................. 199-200

Appendices ........................................................................................................... 201-214

Appendix A  Stop Word List .............................................................................. 201
Appendix B  Malayalam ISCIel-Unicode Mapping Table .................................. 203
Appendix C  A View of Lexicon used in MaQAS .............................................. 204
Appendix D  Performance of POS Tagger .......................................................... 205
Appendix E  Patterns for Phrase Identification ................................................. 207
Appendix F  Performance Evaluation of the AIS-based Phrase chunker .......... 208
Appendix G  A Sample Malayalam Document .................................................. 209
Appendix H  List of Sample Questions ............................................................... 211
Appendix I  Screen shots showing Output of MaQAS ..................................... 214
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page No</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Gender Suffixes of Nouns and Pronouns</td>
<td>34</td>
</tr>
<tr>
<td>3.2</td>
<td>Case-forms in Malayalam</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>Verbs and Tenses</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>List of Possible Questions and Answers w.r.t Example A</td>
<td>46</td>
</tr>
<tr>
<td>4.2</td>
<td>Patterns for Keyword Identification</td>
<td>52</td>
</tr>
<tr>
<td>4.3</td>
<td>Classification of Question</td>
<td>56</td>
</tr>
<tr>
<td>4.4</td>
<td>Outputs of Various Stages of Answer Extraction Module</td>
<td>65</td>
</tr>
<tr>
<td>5.1</td>
<td>Accuracy Comparison of Compound Word Splitting Methods</td>
<td>77</td>
</tr>
<tr>
<td>5.2</td>
<td>Examples of Transformation and Addition/Deletion Algorithm</td>
<td>84</td>
</tr>
<tr>
<td>5.3</td>
<td>State Table for the FST in Fig 5.1</td>
<td>86</td>
</tr>
<tr>
<td>5.4</td>
<td>Examples of Compound Words and Components</td>
<td>89</td>
</tr>
<tr>
<td>6.1</td>
<td>Example of Case Relations</td>
<td>101</td>
</tr>
<tr>
<td>6.2</td>
<td>Examples of VERB Tags</td>
<td>101</td>
</tr>
<tr>
<td>6.3</td>
<td>POS Tags</td>
<td>102</td>
</tr>
<tr>
<td>6.4</td>
<td>Examples of Postpositions</td>
<td>103</td>
</tr>
<tr>
<td>6.5</td>
<td>Output of Word Analyzer for example 3</td>
<td>107</td>
</tr>
<tr>
<td>6.6</td>
<td>Output of Tag Marker</td>
<td>109</td>
</tr>
<tr>
<td>6.7</td>
<td>Tagging Features</td>
<td>113</td>
</tr>
<tr>
<td>6.8</td>
<td>Example of POS Tag Disambiguation</td>
<td>114</td>
</tr>
<tr>
<td>6.9</td>
<td>Output of POS Tagger</td>
<td>115</td>
</tr>
<tr>
<td>6.10</td>
<td>A Typical Contingency Table</td>
<td>117</td>
</tr>
<tr>
<td>6.11</td>
<td>Overall Performance of POS Tagger</td>
<td>118</td>
</tr>
<tr>
<td>7.1</td>
<td>Phrase Tags</td>
<td>129</td>
</tr>
<tr>
<td>7.2</td>
<td>Phrase Chunker output for Example 1</td>
<td>131</td>
</tr>
<tr>
<td>7.3</td>
<td>Phrase Chunker output for Example 2</td>
<td>132</td>
</tr>
</tbody>
</table>
List of tables

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4</td>
<td>Final Output of Phrase Chunker</td>
<td>133</td>
</tr>
<tr>
<td>7.5</td>
<td>Performance Evaluation of AIS based Phrase Chunker</td>
<td>134</td>
</tr>
<tr>
<td>7.6</td>
<td>Overall Performance of Phrase Chunker</td>
<td>134</td>
</tr>
<tr>
<td>8.1</td>
<td>Named Entity Tag Set</td>
<td>148</td>
</tr>
<tr>
<td>8.2</td>
<td>NER Performance by Named Entity type</td>
<td>152</td>
</tr>
<tr>
<td>8.3</td>
<td>Overall Performance of the Named Entity Tagger</td>
<td>152</td>
</tr>
<tr>
<td>8.4</td>
<td>NE Tagging Examples</td>
<td>154</td>
</tr>
<tr>
<td>9.1</td>
<td>Contingency Table</td>
<td>158</td>
</tr>
<tr>
<td>9.2</td>
<td>Contingency Table Showing MaQAS output</td>
<td>162</td>
</tr>
<tr>
<td>9.3</td>
<td>Performance of MaQAS</td>
<td>163</td>
</tr>
<tr>
<td>9.4</td>
<td>Performance According to Question Type</td>
<td>165</td>
</tr>
<tr>
<td>9.5</td>
<td>Performance According to Answer Type</td>
<td>166</td>
</tr>
<tr>
<td>9.6</td>
<td>Performance of Different Runs</td>
<td>167</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

1.1 A Standard Question Answering System .............................................................. 4
3.1 Basic Parts-of-Speech in Malayalam Language ..................................................... 28
4.1 System Architecture of MaQAS ......................................................................... 44
4.2 Named Entities and their Occurrences in a Document D1 ................................. 48
4.3 Index Preparation .............................................................................................. 49
4.4 Question Analysis Module .................................................................................. 51
4.5 Answer Extraction Module .................................................................................. 62
4.6 Output of MaQAS ............................................................................................... 66
5.1 FST for Compound Word Splitter ....................................................................... 85
5.2 Output of Compound Word Splitter ................................................................... 88
6.1 Block Diagram of POS Tagger ............................................................................ 105
6.2 FST for Tag Marker ............................................................................................ 108
6.3 Graphical structure of chain-structured CRF’s .................................................. 109
6.4 Screen Shot of POS Tagger Output ................................................................. 116
7.1 Detailed Architecture of Phrase Chunker ........................................................... 126
7.2 Working of Phrase Separator .............................................................................. 128
7.3 Screen shot of Phrase Chunker .......................................................................... 132
8.1 Pairwise SVM Decision boundaries on a basic Problem .................................... 146
8.2 Block diagram of NE Tagger ................................................................................ 147
9.1 Set Diagram Showing Elements of Precision and Recall ................................... 158
This introductory chapter provides essential background to the area of Question Answering. General Architecture of a Question Answering System is discussed. It also provides the motivation behind this research work and concludes the chapter with a description on organization of this thesis.

Communication with computers is a dream of mankind since the beginning of computer era. Since its inception in 1960 several key developments had happened. The notable developments are Natural Language Database Front Ends, Dialog systems, and Natural Language Understanding systems. To have accurate and effective communication, computer must understand Natural Language. Also it must produce responses in a natural way. Natural Language Understanding (NLU) is a branch of Artificial Intelligence (AI) that deals with the issues concerned with man-machine interface [1]. Information Retrieval (IR) and Question Answering Systems (QAS) are two examples of NLU systems.

Information Retrieval, an example of human computer interaction, is the art and science of retrieving from a collection of documents a subset that serves the user’s purpose [2]. A system which has the capability to synthesize
an answer to a query by drawing on bodies of information which reside in various parts of the knowledge base is called a Question Answering System [2]. IR systems do not have such deduction capability.

1.1 What is Question Answering?

Natural Language Processing (NLP) is a theoretically motivated range of computational techniques for analysing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human like language processing for a range of tasks or applications. It began as a branch of Artificial Intelligence. NLP gives machines the ability to read and understand the human language [1].

Information Retrieval is the area of study concerned with searching of documents for information within documents and information about documents (metadata). Information Extraction (IE) is a type of IR and its goal is to automatically extract structured information from unstructured or semi structured machine readable format. In most cases IE requires Natural Language Processing [2].

But a Question Answering System aims at automatically finding concise answers to arbitrary questions phrased in natural language. Compared to standard document retrieval systems which just return relevant documents to a query, a QAS has to respond with a specific answer to Natural Language (NL) query. Traditionally IR concentrates on finding whole documents while QAS tries to provide only one or a small set of specific answers to an input question [2].

The idea of using computers to search for relevant pieces of information was popularized through the article “As We May Think” [3] by Venevar Bush in 1945. Early IR systems came into existence in 1950s and
1960s. By 1970, several different techniques were evolved. The QA System JASPER [4] was built for providing real time financial news to financial traders. By the beginning of 1987, IE was spurred by a series of Message Understanding Conferences (MUC). MUC [5] is a competition based conference that focuses on the domains such as Naval, Terrorism, Satellite Launch etc.

A question may be either a linguistic expression or a request made by such an expression. Questions are normally asked using interrogatives. Questions can be of different forms. This research work considers Factoid, List, Definition, and Descriptive questions [6]. Factoids are those for which the answer is a single fact. List questions are factoid questions that require more than one answer. Unlike definition questions descriptive questions require a more complex answer, usually constructed from multiple source documents.

Answers are given in response to questions or request for information. There are many ways of describing an answer. As per Text REtrieval Conference (TREC-8) definition, answer size is restricted to a string of up to 50 or 250 characters. This work also restricts the answer size according to TREC norms [6].

1.2 General Architecture of a Question Answering System

The major goal of a QA System is to provide an accurate answer to a user’s question. General Architecture of a QA System is shown in Fig 1.1. The question analysis stage analyses the NL question and determines the type of expected answer type. Based on the question analysis, a retrieval query is formulated and posed to the retrieval stage. The retrieval component returns a set of ranked list of documents which are further analysed by the document
Chapter-1

analyzer based on the expected answer type. This component passes a list of candidate answers to the answer selection module. This final stage returns a single answer or a sorted list of answers to the user.

1.3 Motivation and Scope

During Literature Survey it was noticed that no Question Answering System existed for Malayalam language. The QAS of other languages are not suitable for Malayalam due to the special complex features of this language. Most of the QAS available in other languages are only document retrieval systems whereas focus of this study is to develop an answer retrieval system for Malayalam. Many of the QAS existing in other languages employ keyword matching rather than Natural Language Understanding techniques.
Document preprocessing is an essential step in a QAS. But no tools were available for the preprocessing of Malayalam language. Hence preprocessing tools such as Part-of-Speech (POS) Tagger and Phrase Chunker are to be developed. Also these tools required word level analysis as most of the words in Malayalam language are compound words. Hence a Compound Word Splitter for Malayalam language was also essential without which evolution of above tools was impossible.

Named Entity (NE) is a meaning bearing word in a sentence. Meaning or semantics is an important point to be considered in a QAS. It is possible to include this fact in the development of NE-based index used for answer identification. But, to the best of our knowledge, a Malayalam NE Tagger was not available for this purpose.

This Question Answering System is designed to impart knowledge and insight in Malayalam to a naive user who knows only the regional language. Domain selected for this research work is the medical field dealing with the health issues, causes and remedies of lifestyle and infectious diseases.

The above ideas led to the formulation of certain important research questions like:

- How to develop a QAS for Malayalam language?
- How to store documents in a system?
- How to retrieve information?
- How to test and evaluate the system?

To solve the above given main research questions, following sub questions were also formulated
Chapter 1

- What levels of NLP processing are required for this work such as morphological, syntactic, and semantic analysis?
- Which are the tools to be developed for the above NLP processing?
- What kind of document representation should be followed?
- How to analyse different types of Malayalam questions?
- What is the type of the answer to be returned, word, sentence or passage?
- What is the retrieval strategy to be followed?
- What are the metrics to be used for performance evaluation?

1.4 Objectives

As pointed out in the earlier section, this research work is an attempt to design and develop a closed domain, monolingual QAS which is capable of providing answers in a word/sentence for factoid and for a few non-factoid types of questions by the deep linguistic analysis of Malayalam documents in the corpus.

Therefore the main objectives of this research work are

- Conduct a detailed Literature Survey to understand the state of art in the field of QAS
- Development of language processing tools like
  - Compound Word Splitter
  - Parts-of-Speech Tagger
  - Phrase Chunker
Introduction

- Named Entity Tagger
  - Creation of Malayalam Lexicon
  - Collection and storage of Malayalam documents pertaining to lifestyle and infectious diseases
  - Identification of pattern sets for question analysis
  - Design of an answer retrieval scheme using double level index search.

1.5 Roadmap

This thesis is organized in ten chapters.

- **Chapter 1** provides the description of a standard Question Answering System.

- **Chapter 2** deals with the existing approaches to QAS. Also a few QAS available in other languages are discussed.

- An overview of Malayalam language is given in **chapter 3**.

- In **Chapter 4**, the architecture of Malayalam Question Answering System, MaQAS is described. MaQAS has three main modules, each one is explained with their design steps and working principles.

- A Compound Word Splitter using Finite State Transducer was developed and its details are given in **chapter 5**.

- **Chapter 6** describes Part-of-Speech Tagger, Tagset developed, methodology adopted and its performance evaluation.
Chapter 7 describes Phrase Chunker implementation. Artificial Immunity System Principles and its application for the development of Phrase Chunker are discussed in this chapter.

• Named Entity Tagger is described in chapter 8. Named Entity Tagset with 26 tags was identified for medical domain.

• Experimental environment and results are discussed in Chapter 9. Performance of MaQAS is mainly evaluated using the metrics precision and recall which is also explained in this chapter.

• Chapter 10 concludes this work by summarizing the research achievements and suggesting directions for future research.

1.6 Chapter Summary

The background, motivation and objectives of the work are clearly mentioned in this chapter. General organization of a QAS is described. The chapter concludes with a layout of the thesis. Brief description of different approaches and types of QAS are discussed in chapter 2.
Various approaches to Question Answering Systems are investigated. Both modern and earlier systems are discussed to have a clear distinction of their features, issues in their implementations, and the way in which they are handled.

Question Answering (QA), an important field of NLP, enables users to ask questions in natural language and get precise answers instead of long list of documents usually returned by search engines. When we trace back the history of computers we could come across the research stories and various developments that happened in various languages worldwide. QA Systems are also available in Indian languages such as Hindi [7], and Telugu [8]. However, no known work is available in Malayalam. Approaches to QA Systems and a few systems developed in various languages are described below.

2.1 Approaches to Question Answering Systems

QA System can be classified based on various factors such as domain of QA, language used for input query and the retrieved answer, types of question asked, kind of retrieved answers, levels of linguistics applied to the documents in the corpus, and the answer resources. Accordingly QA Systems fall into open
domain or closed domain, monolingual or multilingual, factoid or non-factoid, document or passage or answer retrieval, deep or shallow, and database or Frequently Asked Questions (FAQ) or web QA Systems.

Open domain Question Answering System is an area of Natural Language Processing research, aimed at providing human users with a convenient and natural interface for accessing information. It deals with questions about nearly everything [9]. These systems usually have much more data available from which to extract the answer. ASKJEEVES [10] is the most well-known open domain QA System. To answer unrestricted questions, general ontologies and world knowledge would be useful. WordNet [11] and Cyc [12] are two popular general resources used in many systems. WordNet is a computational lexicon of English based on psycho-linguistic principles, created and maintained at Princeton University. It encodes concepts in terms of sets of synonyms called synsets. Cyc is an AI project that attempts to assemble a comprehensive ontology, a knowledge base of everyday commonsense knowledge with the goal of enabling AI applications to perform human-like reasoning. Closed domain Question Answering Systems deal with questions under a specific domain (for example, medicine or automotive maintenance), and can be seen as an easier task because NLP systems can exploit domain-specific knowledge frequently formalized in ontologies [9]. Closed domain refers to a situation where only limited types of questions are accepted. In a closed domain QA, correct answers to a question may often be found in only very few documents since the system does not have large retrieval set. Green’s BASEBALL [13] system is a restricted domain QA System that only answers questions about the US baseball league over a period of one year.
In a QA System, questions and answers are given in natural languages. Hence QA Systems can be characterized by the source (question) and the target (answer) languages. Based on these languages QA Systems are named as monolingual [14], multilingual or cross-lingual systems [15]. TREC QA [6] Track and New Trends in Computing and Informatics Research Question Answering Challenge (NTCIR QAC) [16] are all monolingual QA Systems which use the same source and target languages. Multilingual or cross lingual systems allow users to interact with machines in their own language thus providing easier and faster information access, but the documents in the corpus are in a different language. This idea emerged in the year 2000.

Question types can also be used to categorize QAS. Different question types may require different strategies to deal with them. There are three question types– Factoid, List and Description. Factoid QA is the simplest as the answers are named entities such as Location, Person, Organization etc. Some Factoid QA Systems return short passages as answers while others return exact answers. List QA is similar to Factoid QA except that a question may have more than one answer.

Description QA [17] is more complex because it needs answers that contain definitional information about the search term or describe some special events. Special summarization techniques are required to minimize the answer size.

Another classification is Shallow or Deep Systems, based on the level of processing applied to the questions and documents [18]. Some Shallow QAS use keyword based techniques to locate interesting passages and sentences from the retrieved documents based on the answer type. Ranking is then done based on syntactic features such as word order, location or similarity to query. But
question reformulation is not sufficient for Deep QA; more sophisticated syntactic, semantic, and contextual processing must be performed to extract or construct the answer. These techniques include Named Entity Recognition and Classification (NERC).

The answer source is an important factor in designing a QA System. Databases are the most popular answer sources that store structured data. Structured Query Language (SQL) is used to retrieve data from databases. LUNAR [19] developed to answer NL questions about the geological analysis of rocks returned by the Apollo Moon Missions is an example of such a database system. The performance of this system was excellent in terms of accuracy achieved. FAQ represent another answer resource in various commercial/business customer service systems. FAQ systems only focus on processing input questions and matching them with FAQs. Like other systems they don’t require question analysis and answer generation stages. For an input question, if an appropriate FAQ is found, then using lookup table method corresponding answer is retrieved [20]. Web QA uses search engines like Google, Yahoo, Alta-Vista etc. to retrieve web pages that contain answers to the questions. Some systems combine the web information with other answer resources to achieve better QA performance. The Web based QA Systems such as MULDER [21], NSIR [22], and ANSWERBUS [23] fall into the category of domain independent QA Systems while START [24] is referred as domain specific QA System. AQUAINT [25] corpus used in TREC QA Track consists of newswire text data drawn from three sources. These kinds of corpora are good sources for QA System research as the quality and quantity of the data are good. News Papers are good sources for open domain QA research as their contents are general and cover different domains.
Current QA Systems are either document/passage/sentence or answer retrieval systems. In these systems, operation starts when a user posts a question into the QA System. The QA System then analyses the question and finds one or more answer candidates from its input sources. Once the answer candidates are retrieved the QAS then evaluates the content of each one and scores them based on the quality of their content. For these QA Systems, output is a document or a passage or a sentence or the exact answer.

2.2 History of Question Answering Systems

Work on early QA Systems began in early 1960s. Two of the most famous early systems are SHRDLU [26] and ELIZA [27]. SHRDLU simulated the operation of a robot in a toy world (the "blocks world"), and it offered the possibility to ask the robot questions about the state of the world. Again, the strength of this system was the choice of a very specific domain and a very simple world with rules of physics that were easy to encode in a computer program. ELIZA, in contrast, simulated a conversation with a psychologist. ELIZA was able to converse on any topic by resorting to very simple rules that detected important words in the person's input. It had a very rudimentary way to answer questions, and on its own it led to the development of a series of chatterbots such as the ones that participate in the annual Loebner prize. These are examples of Dialog systems mostly influenced by Turing test suggested by Alan Turing [28].

Two of the most famous restricted domain QA Systems developed in the 1960s were BASEBALL and LUNAR. These systems were interfaced against databases. BASEBALL system developed by Green Chomsky and Laughery answered questions about the US baseball league over a period of one year. This was done by using shallow language parsing techniques. Another system similar to
BASEBALL was developed by Woods and was named LUNAR. Both QA Systems were very effective in their chosen domains. In fact, LUNAR was demonstrated at a lunar science convention in 1971 and it was able to answer 90% of the questions in its domain posed by people untrained on the system. Further restricted-domain QA Systems were developed in the following years. The common feature of all these systems is that they had a core database or knowledge system that was hand-written by experts of the chosen domain.

The 1970s and 1980s saw the development of comprehensive theories in computational linguistics, which led to the development of ambitious projects in text comprehension and question answering. One example of such a system was the Unix Consultant (UC), a system that answered questions pertaining to the UNIX operating system [29]. The system had a comprehensive hand-crafted knowledge base of its domain, and it aimed at phrasing the answer to accommodate various types of users. Another project was LILOG [30], a text-understanding system that operated on the domain of tourism information in a German city. The systems developed in the UC and LILOG projects never went past the stage of simple demonstrations, but they helped the development of theories on computational linguistics and reasoning [23].

Over a period of time many open domain QA Systems have been developed that allows questions on a multiple range of topics. Such systems included START, ANSERBUS, BrainBoost [31], EPHYRA [32] and Qualim [33]. START utilized a knowledge base to answer user’s question. Knowledge base was first created automatically from unstructured Internet data. Then it was used to answer natural language questions.

With the increased popularity of QA Systems TREC started the QA track in 1999. In the late 1990s the annual Text Retrieval Conference included a
question-answering track which has been running till today. Systems participating in this competition were expected to answer questions on any topic by searching a corpus of text that varied from year to year. This competition fostered research and development in open-domain text based question answering. The best system of the 2004 competition achieved 77% correct fact-based questions [34].

In 2007 the annual TREC included a blog data corpus for question answering. The blog data corpus contained both "clean" English as well as noisy text that include badly-formed English and spam. The introduction of noisy text moved the question answering to a more realistic setting. Real-life data is inherently noisy as people are less careful when writing in spontaneous media like blogs. In early years the TREC data corpus consisted of only newswire data that was very clean [35].

An increasing number of systems include the World Wide Web as one more corpus of text. Currently there is an increasing interest in the integration of question answering with web search. Ask.com is an early example of a system, which was followed in subsequent years by other natural language search engines. Google and Microsoft have also started to integrate question-answering facilities in their search engines. However, these tools mostly work by using shallow methods and return a list of documents.

### 2.3 Modern Question Answering Systems

Early systems mostly used keyword matching techniques while current systems are based on linguistic principles.

Information Retrieval community has investigated many different techniques to retrieve passages from large collections of documents for question
The work discussed in [36] quantitatively compares the impact of sliding windows and disjoint windows on the passage retrieval for question answering. For the TREC factoid QA task, retrieval of sliding windows outperforms retrieval of disjoint windows. For the task of retrieving answers to why-questions from wikipedia data, the best retrieval model is Term Frequency Inverse Document Frequency (TFIDF) and sliding windows give significantly better results than disjoint windows. Here experiment is conducted with three retrieval models, TFIDF, Okapi, and a language model based on the Killback-Leibler divergence [37].

In IR4QA system [38], first phase is query processing. This phase produces a set of keywords. They are passed to the retrieval model which outputs a list of relevant documents in order. The re-rank module adjusts the ranking of these relevant documents by considering various features such as frequency, position in paragraph, term’s distribution etc. This system just provides satisfactory performance due to lack of query terms.

In the above systems, query processing units separate keywords and other unimportant words without considering any syntax or semantics of the question. First phase of the work described in [39] is natural language query processing which build the syntax representation of query, and transform it into a semantic representation using transformation rules. This phase is designed using fundamental ideas of W Cafe [40]. According to Chafe, syntax model is built on the concept of relationship between words such as object, subject, verb, and their Parts-of-Speech. The semantic model is built on Chafe’s point of view about semantic structure; it is defined as the relationship between verb and its arguments. Syntax structure is transformed into semantic structures by the Semantic Deductive Generator component using predefined transformation.
rules. From the semantic representation model of query, database queries generator module will generate a set of database queries or SQL commands. These commands are executed to get results.

In [41] authors discuss a new model for question answering, which improved the two main modules of QA System, question processing and answer validation. This is an answer extraction model while previous systems were either document or passage retrieval systems. In this model, first of all questions are processed syntactically and semantically. This information is used to classify the question and to determine answer type. Then the query reformulation component converts the question into SQL statement. Then the search engine finds candidate answer documents and sends them to answer processing module to extract correct answer.

ASQA [42] is Question Answering System for complex questions. The question processing module of this system uses surface text patterns to retrieve a question’s topic. Documents are indexed by character based Indexing scheme. This scheme is similar to the one used by the open source IR engine Lucene [43]. Lucene is a high performance, full featured text search engine library. Boolean search using AND, as well as OR, as keywords, are used to retrieve documents relevant to the query. After retrieving the documents they are split into several sentences. Sentence selection module uses co-occurrence based or entropy based methods to find relevant sentences. This system is a sentence retrieval system based on topics and uses Boolean strategy for retrieval where character index is adopted. Retrieval performance is only 65%. No syntax or semantic processing is used in this system.

A Question Answering System for Japanese is presented in [44]. First stage of this system is a passage selection module. Each passage is of size three
consecutive sentences. After performing a preliminary analysis a passage selection algorithm is used for ranking all passages in each document and selects top \( N \) passages for further processing. Then each passage is scored using the count of query terms, their occurrence in the passages, and their inverse document frequency.

One of the important requirements for a QA System is to predict what type of answer the question requires: a person name, location or organization. In the above system they have defined 62 answer types and developed a method that classifies questions into the answer types using the LSP's (Lexico-Semantic Pattern). LSP is a pattern that is expressed by lexical entries, part-of-speech (POS), syntactic categories and semantic categories. Once answer type of a question is determined entities belonging to the answer type within the passages selected in the previous step are extracted. LSP grammar is constructed for this purpose. After extracting answer candidates, some of them are filtered out and the remaining answers are scored using a specific expression.

Work presented in [45] is a web based QA System which retrieves answers from web documents. The user’s question is transformed into an IR query and delivered to the web search engines or ports. The retrieved documents are linguistically analysed and prepared a semantic representation. The semantic representations of questions and answers are compared to find answers.

DefArabicQA [46] is a definitional QA System for Arabic language. This system uses a pattern approach to identify exact and accurate definitions about organization using web resources. Question analysis module identifies expected answer type and topic using certain question patterns and interrogative pronoun of
the question. The passage retrieval module collects the top n snippets retrieved by the web search engine. Then the definition extraction module extracts candidate definitions from these snippets based on the question topic. Definitions are identified with the help of lexical patterns. Definitions are ranked using statistical approach and top-5 definitions are presented to the user.

QArabPro [47] is a rule based QA System for Arabic. Question reformulation section process the input question and formulate the query. An IR system is used to search and retrieve relevant documents which are constructed using salton’s statistical Vector Space Model (VSM). Then the rules for each WH questions are applied to the candidate document that contains the answer. Each rule awards a certain number of points to each sentence in the document. After applying the rules the sentence with the highest score is marked as the answer.

Samir Tartir et al. [48] presented a Hybrid Natural Language Question Answering System (NLQAS) on Scientific Linked Data sets as well as Scientific Literature in the form of publications. SemanticQA processes information need expressed in the form of NL query. Then it retrieves relevant answers from well established Linked Data Sets (LDS). If the answer is not found in LDS system it gathers all the relevant clues and conducts a semantic search on relevant publication. The answers are extracted from these documents and ranked using a novel measure the Semantic Answer score. This score returns the best answer from relevant documents.

A QA System for Portuguese language is described in [49]. Once the question is submitted, it is categorized according to question typology and through an internal query a set of potentially relevant documents is retrieved. Each document contains a list of sentences which were assigned the same
category as the questions. Sentences are weighted according to their semantic relevance and similarity with the question. Next through specific answer patterns these sentences are again examined and the parts containing possible answers are extracted and weighted. Finally a single answer is chosen among all candidates.

MAYA [50] is a QA System for Korean language that uses a predictive answer indexer. Answer indexer extracts all answer candidates in a document in indexing time. Then it gives scores to the adjacent content words that are closely related with each answer candidate. Next it stores the weighted content words with each candidate into a database. During retrieval time MAYA just calculates the similarity between a user’s query and the candidates. Therefore it minimizes the retrieval time and enhances the precision.

Log Answer [51] is QA System for German language. User enters a question into the interface and Log Answer presents a list of answers to the user. These are derived from an extensive knowledge base. This knowledge base is obtained by translating a snapshot of entire German Wikipedia into a semantic network representation in the Multi Net formalism. The question is analysed by linguistic methods and then translated into a Multi Net [52] and First Order Logic (FOL) representation. The Wikipedia contents are matched against the given query combining retrieval and shallow linguistic methods. They compute lists of features like the number of lexemes matching between passage and question or the occurrence of proper names in the passage. A Machine Learning based ranking technique uses these features to filter out the most promising text passages resulting in upto 200 text passages which might be relevant to the query. The FOL representation of these passages is individually tested by theorem prover E-KRHyper [53] each in conjunction
with the background knowledge base and the logical query representation. The proofs are ranked by a classifier and the highest ranked proofs or candidates are translated back into NL answers that are displayed to the user.

RitsQA [54] is a system for non-factoid questions developed for Japanese language. Question analyzer analyses the question pattern and determines its type. IR module called Namazu is used to retrieve the top 100 documents. Also clue words are used to retrieve 10 snippets of Google search. Then similarities between these two results are measured to reorder the retrieved documents. Answer Extraction Module extracts paragraphs which include linguistic clues and some clue words of question sentence. Extracted paragraphs will be a target for answer strings.

Marsha QAS [55] is a Chinese Question Answering System. The query processing module recognizes known question types and formulates queries for the search engine. Most of these question types correspond to typical Named Entity classes used in IE systems.

### 2.4 QA Systems for Indian Languages

QA System described in [7] was developed for HINDI language. Here the question submitted by the user is analysed to identify its type. The question is then parsed to separate the important keywords by identifying the domain entities and filtering out stop words. The query formulation translates the question into a set of queries that is given to the retrieval engine. The engine returns top passages after weighting and ranking them. Finally answer selection is done by selected passage analysis. The retrieval process is carried out using a word level inverted index using all of the terms in the generated query. The selected documents are ranked by locality based similarity heuristic. The
similarity between query and document is measured using the distance between the keywords.

Rami Reddy et al. [8] discusses a keyword based QA System for a huge domain (i.e. for Railways), which aims at replying user’s questions in Telugu. Telugu is an important language in India belonging to the Dravidian family and spoken by the second large population in India. In this keyword based approach the input query statement is analysed by the query analyzer which uses domain ontology stored as knowledge base. The appropriate query frame is selected based on the keywords and the tokens in the query statement. Each query frame is associated with an SQL generation procedure which generates an SQL statement. Once the SQL statement is generated it is triggered on the database and the answer is retrieved. Each query frame has its corresponding answer generator. Template based answer generation method is used for answer generation. Each template consists of several slots. Those are filled by the retrieved answer and tokens generated from the query. The answer will be sent to the Dialogue Manager which will further send it to the user. This system showed 96.34% of precision and 88.66% of dialogue success rate.

2.5 Named Entity Based QA Systems

Some NE based QA Systems are described in this section.

The main objective of QA4MRE [56] is to develop a methodology for evaluating machine reading systems through question answering and reading comprehension tests. Machine reading task obtains an in-depth understanding of just one or a small number of texts. The task focuses on the reading of single documents and identification of the correct answer to a question from a set of possible answer options. The Conditional Random Field (CRF) -based Stanford
Named Entity Tagger 5 (NE Tagger) has been used to identify and mark the named entities in the documents and queries.

The system discussed in [57] is an Answer Retrieval system based on named entities. Here the NL question entered by the user is analysed and processed which determines the kind of answer expected. The first phase is a document retrieval phase that finds documents relevant to the question. Next is the sentence selection phase. From the relevant documents found by the first phase, all sentences are scored against the question. The sentences remaining after the sentence selection phase are then analysed for named entities. All named entities found in the sentences are considered to be possible answers to the user question. The best answer (i.e. with the highest score and matching the question type) is returned to the user. Instead of a list of relevant documents, this QA System tries to find an exact answer to the question.

ArabiQA [58] is a named entity based QA System for Arabic language. Question analysis module of this system determines the type of the given question, question keywords and the named entities appearing in the question. Passage retrieval module retrieves passages which are estimated as relevant to contain the answer. It uses a Distance Density Model to compare the n-grams extracted from the question and the passage to determine the relevant passages. Java Information Retrieval System (JIRS) searches for relevant passages and assigns a weight to each one of them. The weight of a passage depends mainly on the relevant question terms appearing in the passage. Named Entity Recognition system tags all named entities within the relevant passage. Candidate answers are selected eliminating NE which do not correspond to the expected type of answer. Final list of candidate answers is decided by means of a set of patterns.
IRSAW [59] is a system that combines IR with a deep linguistic analysis of texts to obtain answers to NL questions in German language. The NL question is transformed into an IR query and Meta information such as question type and expected answer type is determined. Question and answer types are calculated using a Naïve Bayer’s Classifier trained on features representing the first N words of the question [60]. Answer types are Locations (LOC), Persons (PER), Organizations (ORG) etc. Question types include yes-no questions, essay questions and questions starting with WH words. IR query is sent to external web sources which return result pages containing Uniform Resource Locators (URL). The web contents referred to by an URL are retrieved and converted into text. These texts are segmented into units and indexed and fed into the local database. Several methods are employed to pinpoint answers. In the InSicht subsystem a linguistic parser analyses the text segments and prepares semantic network representation. Then the representations of questions and texts are compared to find answers. The shallow technique for finding answers in IRSAW is based on pattern matching. Each word in the passage is assigned a set of tags including Part-of-Speech. These sequences of symbols are analysed using context window to locate certain patterns. The pattern matching returns an instantiation of the answer variables.

TextractQA [61] explains the role of IE in a QA application. There are two components in this system, the question processor and text processor. The question processing results are a list of keywords plus the information for asking point. The question processor scans the question to search for question words and maps them into corresponding NE types. On the text processing side the question is first sent to a search engine and obtains top 200 documents for further IE processing. This processing includes tokenization, POS Tagging, and NE Tagging. Then the text matcher attempts to match the question template.
with the processed documents for both the asking point and the keywords. There are three levels of ranking schemes. Primary ranking is a count of how many unique keywords are contained within a sentence. The secondary ranking is based on the order in which the keywords appear in the sentence compared to their order in the question. Tertiary ranking is based on whether there is an exact match or a variant match for the key verb.

2.6 Chapter Summary

QA Systems are categorized based on various input and output factors. The origin of QAS, and several systems developed later are described in detail. Certain QA Systems developed were mainly different in the methodology used and in the output. Even though many systems are available in English and European languages, no such systems are available in Malayalam. Since this research work is a NE based QA System, similar systems are also discussed in this chapter.
Malayalam is the principal language of Kerala, the southern most state of India. The word Malayalam probably originated from the Malayalam/Tamil words “mala” meaning hill, and “elam” meaning region. The word “malaelam” (hill region) was used to refer to the land of Chera Kingdom. Kerala was a part of ancient Chera Kingdom and when Kerala became a separate entity “malaelam” became the name of its language ie “Malayalam”. The name “Kerala” was derived from the word “Cheralam”.

Dravidian Languages were first recognized as an independent family in 1816 by Francis W Ellis, a British Civil servant. The term Dravidian (adjective form of Dravida) was first employed by Robert A Caldwell. Dravidian Languages, a family of some 75 languages is spoken primarily in South Asia. These languages are divided into South, South-Central, Central and North groups; these groups are further organized into 24 subgroups. The four major literary languages – Telugu, Tamil, Malayalam, and Kannada – are recognized by the constitution of India [62].

Malayalam belongs to the Dravidian family of languages and is spoken by the people of Kerala. It is one of the 22 Scheduled Languages of India with
official language status in the State of Kerala and Lakshadweep Islands. It is spoken by about 40 million people. In terms of the number of speakers Malayalam ranks eighth among the fifteen major languages of India [63].

Malayalam first appeared in writing in the vazappalli inscription which dates back about 830 AD. The ancient Malayalam script originated in 13\textsuperscript{th} century from a script known as vattezhuthu (round writing) a descendant of the Brahmiscript. Now Malayalam character set consists of 73 basic letters [64] [65].

Malayalam is a morphologically rich agglutinative language and is relatively of free order. Also Malayalam has a productive morphology that allows the creation of complex words which are often highly ambiguous. Due to its complexity, development of an NLP system for Malayalam is a tedious and time consuming task. No tagged corpus or tag set is available for this language. NLP systems developed in other languages are not suitable for Malayalam language due to its differences in morphology, syntax, and lexical semantics.

![Fig 3.1 Basic Parts-of-Speech in Malayalam Language](image)

### 3.1 Basic Word Types

According to Keralapanineeyam written by Sri. A R Rajarajavarma [66], a Malayalam word may fall into one of the categories given in Fig 3.1.
In this thesis, words in Malayalam are represented in three forms—using Malayalam font, transliterated version, and in English where transliterated version is given in italics.

As given in Fig 3.1 Malayalam words (Sadbams) are classified as ‘vaachakam’ and ‘dyOthakam’. ‘vaachakam’ is further classified into namam (noun), sarvanamam (pronoun), kriya (verb), and bhEdakam (qualifier). ‘dyOthakam’ has three sub-categories namely gathi (preposition), ghaTakam (conjunction), and vyaakshEpakam (interjection). But most of the words in Malayalam are compound words and such a word may consist of an arbitrary number of prefixes, stems (nouns, verbs, pronouns etc.) and an arbitrary number of suffixes [67]. Unlike English, Malayalam does not contain spaces or other word boundaries between the constituents of the compound word.

Example of a compound word

െപാളിച്െചഴു തണെമŋാണ് (poLicchezhuthaNamennaN~) (It must be revised)

This word is a combination of five atoms as shown below.

കെപാളി + എഴു + ത് + അണ + ആണു (verb +verb +suffix + dyOthakam +Aux-Verb)

Basic POS shown in Fig 3.1 are explained in the sub-sections below.

3.1.1 Nouns

Noun is classified into concrete noun and abstract noun. The subclasses of concrete nouns are proper nouns, common nouns, material nouns, and collective nouns. Abstract noun is further classified into quality nouns and verbal nouns.
Examples

1. Concrete Noun

- **Ko** (kochchi) (Cochin) – Proper Noun
- **PaTTa** (paTTaNam) (city) – Common Noun
- **MaNN~** (maNN~) (sand) – Material Noun
- **KooTTa** (kooTTam) (group) – Collective Noun

2. Abstract Noun

- **Chiri** (chiri) (laugh) – Quality Noun
- **NaTaththam** (naTaththam) (walk) – Verbal Noun

3.1.2 Pronouns

A pronoun is a word used instead of a Noun. This POS is mainly of three types; First Person, Second Person, and Third Person.

- **Njaan** (njaan) (I) – First Person
- **NingngaL** (ningngaL) (you) – Second Person
- **Avan** (avan) (he) – Third Person

Third Person is again classified into ten different forms [66].

Definite Pronoun can be of four types.

- **Eth~** (Eth~) (which) – Interrogative Pronoun
- **A** (a) (that) – Demonstrative Pronoun
- **Ae** (ae) (who) – Relative Pronoun
- **Thante** (thante) (your) – Reflexive Pronoun
Indefinite Pronoun has six forms in Malayalam language.

<table>
<thead>
<tr>
<th>Indefinite Pronoun</th>
<th>Meaning (Malayalam)</th>
</tr>
</thead>
<tbody>
<tr>
<td>നീ (chila)</td>
<td>(some)</td>
</tr>
<tr>
<td>നാനാ (naanaa)</td>
<td>നാനാ (naanaa sarvanaamam)</td>
</tr>
<tr>
<td>നിന്ന് (inna)</td>
<td>(what)</td>
</tr>
<tr>
<td>നിര്ദിശ്താവാച്ചി (nirddishTavaachi)</td>
<td></td>
</tr>
<tr>
<td>എല്ലാ (ella)</td>
<td>(all)</td>
</tr>
<tr>
<td>സാര്വ്വാചി (sarvvavaachi)</td>
<td></td>
</tr>
<tr>
<td>മിക്ക (mikka)</td>
<td>(most)</td>
</tr>
<tr>
<td>അമ്മാവി (amSavaachi)</td>
<td></td>
</tr>
<tr>
<td>മറ്റ് (mat–)</td>
<td>(another)</td>
</tr>
<tr>
<td>അന്യാർത്ഥ്വാചി (anyaarththhakam)</td>
<td></td>
</tr>
<tr>
<td>വള്ള (valla)</td>
<td>(any)</td>
</tr>
<tr>
<td>അനാസ്ഥ്വാചി (anaasthavaachi)</td>
<td></td>
</tr>
</tbody>
</table>

3.1.3 Verbs

Verbs are divided into four categories based on their meaning, behaviour, feature, and importance. The first classification is transitive verbs and intransitive verbs. Another classification based on the behaviour is simple verbs and causatives. Third type of classification is strong and weak verbs. Last division is according to its importance and is named as finite and infinite verbs.

<table>
<thead>
<tr>
<th>Verb</th>
<th>Meaning (Malayalam)</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>കണ്ടു (kaNTu)</td>
<td>(saw)</td>
<td>Transitive Verb</td>
</tr>
<tr>
<td>കുറിക്കുന്ന് (kurakkunnu)</td>
<td>(barking)</td>
<td>Intransitive Verb</td>
</tr>
<tr>
<td>പാടുന്ന് (paatunnu)</td>
<td>(singing)</td>
<td>Simple Verb</td>
</tr>
<tr>
<td>പാടിക്കുന്ന് (paatikkunnu)</td>
<td>(make one sung)</td>
<td>Causatives</td>
</tr>
<tr>
<td>വായിക്കുന്ന് (vaayikkunnu)</td>
<td>(reading)</td>
<td>Strong Verb</td>
</tr>
<tr>
<td>തുണ്നുന്ന് (thunnunnu)</td>
<td>(stitching)</td>
<td>Weak Verb</td>
</tr>
<tr>
<td>പരാജണി (paRanjnu)</td>
<td>(told)</td>
<td>Finite Verb</td>
</tr>
<tr>
<td>ഓടഞ്ഞാ (OTunna)</td>
<td>(running)</td>
<td>Infinite Verb</td>
</tr>
</tbody>
</table>
Infinite verb or participle is divided into പരേച്ചം (pErechcham) (Adjectival Participle), and വിനയേച്ചം (vinayechcham) (Adverbal Participle).

### 3.1.4 Qualifiers

Three types of qualifiers are there in Malayalam—qualifiers of nouns (adjective), qualifiers of verbs (adverb), and qualifiers of qualifiers.

- സ്നേഹം (minnunna) (glittering) – Adjective
- ഉറക്കെ (uRakke) (loudly) – Adverb
- വലരെ (vaLare) (too) – Qualifier of Qualifier

### 3.1.5 Dhyodhakam

‘dyOthakam’ is classified into prepositions, conjunctions and interjections [66]. In this work they are commonly referred as ‘dhyodhakams’.

- മുതൽ (muthal) (from) – gathi (preposition)
- ഉം (um) (and) – ghaTakam (conjunction)
- ആഹ (aaha) (sound showing wonder) – vyaakshEpakam (interjection)

### 3.1.6 Affixes

Malayalam words are combinations of the above mentioned basic word types and affixes. Affixes are of three types- Prefix, Postfix, and Suffix [68].

**Prefixes** are used to obtain a subdam (sound) from a root word with same or different meanings. Sometimes a new subdam might have an entirely different or opposite meaning. There are three types of prefixes.
First type - with opposite meaning.

Example - സതിപക്ഷം (prathipaksham) (opposite party) where ‘ഭാ’ (prathi) is the prefix.

Second type - same meaning but with emphasis

Example - സുസാദ്യം (susaadhyam) (that which is certainly possible) ‘സു’ (su) is the prefix

Third type - same meaning.

Example - പ്രഭാഷണം (prabhaashaNam) (speech) ‘പ്രഭ’ (pra) is the prefix

Postfix is mainly used for completing or changing the meaning of verbs. They are of four types. In the following examples underlined portion of the words are the postfixes.

1. കണ്ണുകളേമ (kaTannukaLanju) (escaped) – ശേഖദകാൽ (bhEdakaanuprayOgam)

2. വരുകെട്ടി (varunnuNT–) (coming) – കാലാൽ (kaalaanuprayOgam)

3. ഇല്ലിരുന്നെ (illayirunnu) (was not available) – പൂര്ണ (pooraNaanuprayOgam)

4. വരുത്ത് (vararuth) (should not come) – നിര്‌ദേശത്ത് (nishEdhaanuprayOgam)

Suffix - Words in Malayalam have a strong inflectional component. For verbs these inflections are based on tense, mood, aspect etc. For nouns and pronouns inflections distinguish the categories of gender, number, and case. These inflections called Suffixes are briefly described below.
A. NOUN- Suffixes

Table 3.1 Gender Suffixes of Nouns and Pronouns

<table>
<thead>
<tr>
<th>Gender Suffixes of Nouns and Pronouns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word type</td>
</tr>
<tr>
<td>Noun</td>
</tr>
<tr>
<td>Pronoun</td>
</tr>
</tbody>
</table>

1. Nouns- Gender

In Malayalam language the gender of nouns can be masculine, feminine, common, or neuter. Common gender suffixes are listed in Table 3.1.

- അച്ചച്ചൻ (achchhan) (father) – masculine
- അമ്മ (amma) (mother) – feminine
- വെല്ല (veLLam) (water) – neuter
- പാക്ശി (pakshi) (bird) – common

2. Nouns- Number

A noun can be either singular or plural. No number suffixes are required in singular form. അ (a), അര (aR), മാ (maaR), and കാ (kaL) are the suffixes used to obtain plural forms of the nouns.

Table 3.2 Case-forms in Malayalam

<table>
<thead>
<tr>
<th>Case</th>
<th>suffix</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>നിരീഡ്സിക (niriddESika) (Nominative)</td>
<td>നിരീഡ്സിക (niriddESika)</td>
<td>മരം (maram)</td>
</tr>
<tr>
<td>പ്രഥിഗ്രാഹിക (prathigraahika) (Accusative)</td>
<td>പ്രഥിഗ്രാഹിക (prathigraahika)</td>
<td>മരത്തി (maraththe)</td>
</tr>
<tr>
<td>സാമ്യോജിക (samyOjika) (Sociative)</td>
<td>സാമ്യോജിക (samyOjika)</td>
<td>മരത്താൽ (maraththaal)</td>
</tr>
<tr>
<td>ഉദ്ദിസിക (uddESika) (Dative)</td>
<td>ഉദ്ദിസിക (uddESika)</td>
<td>മരത്തിന്ന് (maraththin~)</td>
</tr>
<tr>
<td>പ്രയോജിക (prayOjika) (Instrumental)</td>
<td>പ്രയോജിക (prayOjika)</td>
<td>മരത്തിക്ക് (maraththinte)</td>
</tr>
<tr>
<td>സംബന്ധിക (sambandhika) (Possessive)</td>
<td>സംബന്ധിക (sambandhika)</td>
<td>മരത്തില (maraththil)</td>
</tr>
<tr>
<td>ആഡ്‌ഹാരിക (aadhaarika) (Locative)</td>
<td>ആഡ്‌ഹാരിക (aadhaarika)</td>
<td>മരത്തിൽ (maraththil)</td>
</tr>
</tbody>
</table>
3. Nouns- Case

Suffixes used, to show the relationships of a noun to other words in the sentence are called case suffixes. There are seven case-forms possible for a noun. Table 3.2 shows the Case–forms and case suffixes available in Malayalam.

B. VERB- Suffixes

1. Verb- Tense

There are mainly three tenses- Present Tense, Past Tense, and Future Tense.

Table 3.3 gives a few examples of root verbs, their various tenses, and suffixes used.

<table>
<thead>
<tr>
<th>Root Verb</th>
<th>भूथम (bhootham) (Past)</th>
<th>वर्थ्थामानम (varththamaanam) (Present)</th>
<th>भावि (bhaavi) (Future)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) കൊട് (koT) (give)</td>
<td>കൊട്ടഥു (koTuthhu) (gave)</td>
<td>കൊട്ടഥുക്കണ്ണ് (koTukkunnu) (gives)</td>
<td>കൊട്ടഥുക്കണ്ണ് (koTukkum) (will give)</td>
</tr>
<tr>
<td>2) ഉണ്ട് (uRangng) (sleep)</td>
<td>ഉണ്ടെ (uRangngi) (slept)</td>
<td>ഉണ്ടെങ്ങാണ് (uRangngunu) (sleeps)</td>
<td>ഉണ്ടെങ്ങാണ് (uRangngum) (will sleep)</td>
</tr>
</tbody>
</table>

Suffixes used in the first example are  തു (thu),  ന്ന് (unnu), and  മ (um) for Past, Present, and Future tenses. In the second example suffixes are  ഇ (i),  ന്ന് (unnu), and  മ (um) respectively.

2. Verb- Mood

The different modes or manners in which a Verb may be used to express an action are called Moods. There are four Moods in Malayalam—Indicative, Imperative, Potential, and Permissive.
The Indicative Mood is used to make a statement of fact and to ask a question. No suffix is required for this purpose. Imperative Mood is used to express a command, order, advice, request or prayer. To make such a statement suffixes such as கூடி (aaTTe), நல்லூர் (aalum), பின் (in), உகா (uka) etc. are used. Potential Mood is to create a feeling of duty, fate or habit by way of adding suffix நம் (aNam) to the root verb. Mood which shows permission is ‘Permissive’ and the suffix required is ஏம் (aam).

- எழுதி (ezhuthi) (wrote) – Indicative
- எழுதக்கூடி (ezhuthaTTe) (shall I write) – Imperative
- எழுதப்படுவது (ezhuthaNam) (I have to write) – Potential
- எழுதுவது (ezhuthaam) (I can write) – Permissive

3. Verb- Voice

Two voices are there for a verb – active and passive.

- ராமன் ரவனனை கொடுத்தது (raaman raavaNane konnu) (Rama killed Ravana) – Active Voice
- ரவனனை ராமன் கொடுத்து (raavaNan raamanaal kollappeTTu) (Ravana was killed by Raman) – Passive Voice

A sentence in Active Voice is converted to Passive Voice by adding ‘ஏல்’ (aal) suffix to the Subject and ‘பெட்டு’ (ppettu) suffix to the verb.

3.2 Phrase Types

Words are grouped together into phrases which are then joined to form sentences. A sentence consists of different types of phrases.
3.2.1 Noun Phrase

A noun phrase is a group of words that does the work of a noun. It contains information about the noun. Normally this noun is a participant of the action described by the verb. In the sentence ഇലയുെട പച്ചനിറം ചെടി ഭക്ഷണം തയയ്ാറാ ര ക (ilayuTe pachchaniRam cheTikku bhakshaNam thayyaARaakkunnu)(Green color of the leaves prepare food for the plant), the underlined phrase is an example of noun phrase.

3.2.2 Verb Phrase

Verb phrase is a constituent of a sentence that contains at least one verb and its complements, objects or other modifiers that function syntactically as a verb. In the previous example ഭക്ഷണം തയയ്ാറാ ര ക (bhakshaNam thayyaARaakkunnu) (prepare food) is the verb phrase.

3.2.3 Adverbial Phrase

An adverbial phrase is a group of related words which plays the role of an adverb. In the example കുളത്തിലെ വെള്ളം വറ്റിയാൽ മല്ലിക മഞ്ഞയം കിട പിടച്ചു (kuLaththile veLllLLam vatiyappOL malsyangaL kiTannu piTachchu) (When the water in the pond dried up the fishes became restless), കുളത്തിലെ വെള്ളം വറ്റിയാൽ (kuLaththile veLllLLam vatiyappOL) is an example of adverbial clause of time.

3.2.4 Adjectival Phrase

An adjective phrase is a group of words in a sentence that can function in the same way as an adjective. Adjectives are used to modify nouns or pronouns.
They give additional details about the meaning of a noun. In the sentence below, underlined phrase is an example of an adjectival phrase.

(Raman who returned to the city after marriage, went to the forest according to the decision of his father)

3.2.5 Postpositional Phrase

In Malayalam language, prepositions come after the dependant noun phrase and are called as postpositions. The postpositions indicate semantic relationship of the dependant noun phrase to the context.

In the sentence (achchhanum ammayum kooTi vannu) (father and mother came together), um and kooTi are the postpositions. (achchhanum ammayum kooTi) is the postpositional phrase.

3.3 Malayalam Sentences

A sentence is a group of words that makes complete sense. They can be classified according to their behaviour or construction.

3.3.1 Sentence Classification Based on Behaviour

Based on the behaviour, sentences can be classified into four types.
Overview of Malayalam Language

**Assertive sentence**: This is a sentence that makes a statement.

*e.g.* കളിസ ഥലിൽ നീളവും വീതിയും ഉം

*(kaListhhalaththinu– vEnTathra nILavum vIthIyum uNt)*

(Play ground has sufficient length and width.)

**Interrogative sentence**: This asks a question

*e.g.* ഹരി നിന്റെ അയൽ ക്കാരനേണാ? *(Hari ninte ayal_kkaaranaNO)* (Is Hari your neighbor?)

**Imperative sentence**: This expresses a command, a request or a prayer.

*e.g.* ഞാർ ക്കാവശ്യമുള്ള ആഹാരം ഞാർ ക്കിണ് തേരണേമ

*(njaNGaLkkaavaSyamuLLa Ahaaram njaNGaLKinn tharENamE)*

(Give us this day our daily bread.)

**Exclamatory sentence**: They are used to express strong feeling, extreme happiness, sorrow, and wonder.

*e.g.* അല്ലാ! ആരാണ് ഈ വഴി വൃത്തം

*(allaa! aaraaN~ ee vazhi varunnath)*

(Oh! Who is coming this way?)

**3.3.2 Sentence Classification Based on Construction**

Sentences in Malayalam can be of three types based on the construction – simple, complex and compound.

**Simple sentences**: This type contains only one main clause.

*e.g.* അക്ബര്‍ മഹാനായ ചക്രവര്‍ത്തി ഉം

*(Akbar mahaanaaya chakravarththi aayirunnu)* (Akbar was a great emperor.)
Complex sentences: These sentences contain one principal clause and any number of subordinate clauses.

られています との ぼうずう づらい るい でら よびに (kuLaththile veLllLam vatiyappOL malsyangal kitannu pitachchu) (When the water in the pond dried up the fishes became restless)

Compound sentences: Compound sentences can have any number of principal clauses.

e.g. あおも いつも おはようございます てつおき てつおき じ ゆうこ じ ゆうこ じ ゆうこ (ayaAL vITtileththi chaaya thayyaaRAyiTTuNTO enn bhaaryayOT chOdichchu) (He reached home and asked his wife whether tea was ready)

3.4 Malayalam Sandhi

‘Sandhi’ means ‘harmony’. When two words are united, certain changes occur to the last letter of the first word and the first letter of the second word. These changes are usually of four types.

1. (dvithvam) (Gemination)

( dvithvam) means two-fold; i.e. while joining two words, first letter of the second word doubles.

(maram) + (komp) = marakkomp

2. (Opam) (Elision)

If an elision of vowel or consonant occurs during the union of two words such a union is called ‘Opam sandhi’.

(pOyi) + (illa) = pOyilla
3. **aagamam** (Augmentation)

A new consonant is coming between the words and this consonant is either ‘yre’ (ya), ‘ma’ (ma), or ‘na’ (na).

\[
\text{ala} + (aazhi) = \text{ayaaazhi}
\]

\[
\text{thiru} + (aathira) = \text{thiruaathira}
\]

\[
\text{kari} + \text{chantha} = \text{karinchantha}
\]

4. **aadESam** (Substitution)

Substitution of a vowel or consonant occurs between the two words when they are combined and that is called **aadESam**.

\[
\text{viT} + (thu) = \text{vTTu}
\]

\[
\text{kuLam} + (il) = \text{kuLathil}
\]

3.5 Chapter Summary

Malayalam is a morphologically rich language. This chapter briefly describes the Parts-of-Speech, phrases and different kinds of sentences available in Malayalam language.
Modern QA Systems are mostly document or passage retrieval systems. But users of QA Systems expect exact answers rather than long list of documents or snippets of documents. In this chapter, a QAS for Malayalam language which retrieves exact answers is described.

A monolingual QAS named MaQAS is developed and this work is capable of providing answers in a word or a sentence for factoid and a few non-factoid type questions by the deep linguistic analysis of Malayalam documents in the medical domain.

This work imparts knowledge to naive users regarding the causes, reasons, remedies, etc. of lifestyle and infectious diseases. MaQAS analyses the question given by the user and returns most relevant answer in a word or a phrase or a sentence. Since named entities help us to extract the essence of the text, MaQAS is developed as a Named Entity-based Question Answering System. The Architecture of MaQAS is explained in the following sections.

4.1 System Architecture

Overall Architecture of MaQAS is shown in Fig 4.1. It consists of three main modules— Indexing Module, Question Analysis Module, and Answer Extraction Module.
Fig 4.1 System Architecture of MaQAS

Indexing Module processes all the documents in the corpus and prepares a NE-based index. When a query is entered by the user, the Question Analysis Module analyses the query sentence and determines its type. Also it
identifies keywords and other significant words. Keywords are useful in finding the entity type of the expected answer and significant words help in determining the final answer. These details are passed to the Answer Extraction Module. Answer Extraction Module uses this information to search through the index and extracts the most relevant answer from the corpus. Each module of MaQAS is described below.

4.1.1 Indexing Module

This module processes all the documents entered into the system and prepares an indexing scheme based on named entities.

There are two stages in this indexing module.

- Document Preprocessing
- Indexing Engine

**Document Preprocessing**

A general Question Answering System normally extracts keywords from the user query and retrieves documents in which the keywords are present. These retrieved documents are the candidate documents from which answer sentences are selected and returned to the user. But sometimes, keywords may not be directly present in the documents. At certain times, meanings of the keywords and that of the words present in the documents may not be the same. In such situations the results obtained are not accurate.

Keyword-matching technique is not used in MaQAS as Malayalam is an agglutinative language. Instead of using keyword matching technique, keywords meanings are used for answer retrieval. Such a system is implemented with the help of Named Entity principles. Named Entities are the
important meaning bearing words in a sentence or a document. They aid in extracting word meanings in the user query as well as in the documents.

Example A ਰాము పిండిక నాయకాబడించాడు (raamu paampine vaTikoNTu konnu) (Ramu killed the snake with a stick)

The questions that can be asked, the expected answer type and respective answers with respect to Example A are listed in Table 4.1

Table 4.1 List of Possible Questions and Answers w.r.t. Example A

<table>
<thead>
<tr>
<th>Questions</th>
<th>Expected Answer Type</th>
<th>Expected Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. మీరు నాయకాబడించాడు? (aaraaN~ paampine konnath~?) (Who did kill the snake?)</td>
<td>AGENT</td>
<td>Ramu</td>
</tr>
<tr>
<td>2. మీరు నాయకాబడించాడు పిండిక గాడాడు (raamu paampine enthukoNTaaN~ konnath~) (How did Ramu kill the snake?)</td>
<td>INSTRUMENT</td>
<td>Stick</td>
</tr>
<tr>
<td>3. మీరు నాయకాబడించాడు బిందిక గాడాడు (raamu enthineyaaN~ konnath~) (What did Ramu kill?)</td>
<td>OBJECT</td>
<td>Snake</td>
</tr>
</tbody>
</table>

For all the questions given in Table 4.1, answer words are nouns but their roles in the sentences are different. These words are Named Entities and their roles are AGENT, INSTRUMENT, and OBJECT respectively.

Mostly role-carrying agents are Nouns (Proper Noun, Common Noun, Pronoun), Adjectives and Adverbs. There are two stages in Named Entity-based systems – NE Recognition (NER) and NE Classification (NEC). NER stage identifies the Named Entity words and NEC assigns their roles. With respect to
MaQAS: a Malayalam Question Answering System

the example A, function of NER is to identify the Named Entities ‘Ramu’, ‘Stick’, and ‘Snake’. NEC finds out their roles as ‘AGENT’, ‘INSTRUMENT’, and ‘OBJECT’.

Example B

(žeerāram kázhičchaSešham raamu uRangngi) (Ramu slept after food)

Suppose the question asked with respect to Example B is ‘eppOzhaaN~ raamu uRangngiyath~?’ (When did Ramu sleep?), the expected answer is ‘žeerāram kázhičchaSešham’ (aahaaram kazhichchaSEsham) (After food) which is a phrase indicating ‘TIME’. This example shows that Named Entity can also be a phrase while in Example A it was a word.

From the above examples it is clear that, NER require tools to identify nouns, adjectives, adverbs and phrases. Tools to identify these POS and phrase chunks are developed for Malayalam language. Word-level and context-level features are essential for POS Tagging and Phrase Chunking. Owing to the compounding nature, this feature extraction requires a Compound Word Splitter which is also developed. Implementation details of Compound Word Splitter are given in chapter 4. POS Tagger, Phrase Chunker, and NE Tagger are described in the chapters 5, 6, and 7 respectively.

Document preprocessing stage preprocesses the documents with the help of these tools and prepares NE tagged documents. Then the Indexing Engine scans these documents to prepare the NE-based index.
Indexing Engine

To facilitate fast and accurate Information Retrieval, document indexing techniques are used. Inverted Index is the most popular data structure used in document retrieval systems. Since MaQAS is a NE-based QAS, Named Entity-based indexing scheme is used.

Procedure for the preparation of NE-based index is given below. For every NE tagged document, named entity list (N), and sentence occurrence lists (O) are prepared and as shown in Fig 4.2.

<table>
<thead>
<tr>
<th>Named Entity List (N1)</th>
<th>Occurrences (O1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERSON</td>
<td>1, 10, 15, 65…..</td>
</tr>
<tr>
<td>ORGANISATION</td>
<td>14, 32…</td>
</tr>
<tr>
<td>DISEASE</td>
<td>0, 4, 17, 23…</td>
</tr>
<tr>
<td>VIRUS</td>
<td>11, 16…</td>
</tr>
<tr>
<td>DATE</td>
<td>6, 21…</td>
</tr>
</tbody>
</table>

**Fig 4.2 Named Entities and their Occurrences in a Document D1**

N1 and O1 are the lists corresponding to document D1. N1 shows the Named Entities that are identified in the document D1. Their point of occurrences (the sentences in which these NEs are present) in the document D1 are given in O1. For example, PERSON entity occurs in the sentences 1, 10, 15, 65 etc. DISEASE entity occurs in sentences 0, 4, 17, 23 etc. In the same way, NE lists and occurrence lists are prepared for the all 120 documents in the corpus. They are N1 to N120 and O1 to O120. By analysing the Named Entity lists N1 to N120 a common NE list E is prepared. For every entry in the list E, a list of documents in which that
entity appears is maintained in the document list D. Fig 4.3 explains the preparation of NE-based index.

In short, document pre-processing stage, processes all the documents in the corpus and detects the named entities that occur in each one of them. Indexing stage comprehends in which documents and in which sentences these entities occur.

List E in Fig 4.3 contains 6 NEs. For each NE in E, a document list is given in D. For example corresponding to PERSON entity in E, the list in D is 1, 2, 4, 9, and 109, i.e., PERSON entity exists in the documents D1, D2, D4, D9, and D109. Then corresponding to D1, N1 contains the PERSON entity and O1 contains the occurrence list. Thus the lists E and D form the first level index and the lists N and O form the second level index.
entities reside. Thus the document numbers and sentence numbers together create a double-level index.

4.1.2 Question Analysis Module

Question processing is one of the main tasks of Question Answering System and this is an initial step in the retrieval of relevant information. In order to answer a question correctly one needs to understand what type of information the question asks for. To identify the purpose of the question, it is analysed in a number of ways and all those features which are helpful to determine the expected answer type are extracted.

Main function of Question Analysis Module given in Fig 4.4 is to take the input set of questions and convert them into a form that can be processed by the Answer Extraction Module. This module extracts main keywords, expands keyword terms, determines question type, and builds the semantic context representation of the expected answer. This module consists of stages for Question Preprocessing, Keyword Selection, and Question Classification.

Question Preprocessor removes unimportant words, from the question that is to be submitted to the Keyword Selector stage. Function of Keyword Selector is to identify primary and secondary keywords. These keywords help the question classifier stage in determining the type of the question and the expected answer type. Each of these stages are described below.
Question Preprocessing

The Query/Question preprocessing includes stop word removal and stemming. Stop words [69] are words which are filtered out prior to or after processing of NL data or text. Any group of words can be chosen as the stop words. Normally less important or short function words are removed as stop words. In IR terms, common words which would appear to be of little value in helping select documents matching a user need are called stop words. Elimination of such words reduces the size of the index. Process of removing such words is called stop word removal. There are many such words in Malayalam language and a few are listed in Appendix A.

A stem is the portion of the word which is left after the removal of its affixes. Frequently the user specifies a word in a query but only a variant of this word is present in a relevant document. This reduces the chances of answer
Chapter-4

detection and retrieval. Stemming [69] is useful for improving retrieval performance because they reduce variants of the same root word to a common concept. Also compound words in the question are decomposed to obtain their root forms.

**Keyword Selector**

This module analyses each and every word of the question and identifies the keywords. Keywords are identified by the Pattern Matching technique. Table 4.2 shows various first level keywords and patterns used for their detection. These keywords are used by the Answer Extraction Module to find out the answers relevant to the question. Keywords are mainly used for detecting the question type.

<table>
<thead>
<tr>
<th>Question Type</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>(aar~) (who)</td>
<td>&quot;\u201a\u00a5[\u00af\u00b8].*&quot;</td>
</tr>
<tr>
<td>(aeth~) (which)</td>
<td>&quot;\u2021 [\u00b5]? \u02dc.*&quot;</td>
</tr>
<tr>
<td>(aaRkk) (whom)</td>
<td>&quot;\u201a\u00dc.*&quot;</td>
</tr>
<tr>
<td>(ethRa) (how much)</td>
<td>&quot;\u2020'\u00bd\u00af.*&quot;</td>
</tr>
<tr>
<td>(enthukonT~) (why)</td>
<td>&quot;\u2020[u00b5]? [u00d0].*&quot;</td>
</tr>
<tr>
<td>(eppOL) (when)</td>
<td>&quot;\u2020[u00b6][u00d4][u00af].*&quot;</td>
</tr>
<tr>
<td>(eNGine)(how)</td>
<td>&quot;\u2020[u00c4][\u00b0][u00b5][u201c].*&quot;</td>
</tr>
<tr>
<td>(eviTe) (where)</td>
<td>&quot;\u2020[u00b5]? [u00d0].*&quot;</td>
</tr>
<tr>
<td>(enth~)(what)</td>
<td>&quot;\u2020[u00b5]?[u00d2][u00af].?[\u00de].*&quot;</td>
</tr>
<tr>
<td>(enn~)(when)</td>
<td>&quot;\u2020[u00b5]?[u00d2][u00af].?[\u00de].*&quot;</td>
</tr>
</tbody>
</table>

Malayalam language keyboard and character encoding has been standardized as Indian Script Code for Information Interchange (ISCII) which is an 8-bit code. ISCII is defined in such a way that all Indian languages can use a single character encoding scheme. User interface of MaQAS is developed using Java that uses two byte Unicode characters. The Unicode standard is the universal character-encoding standard used for representation of text for computer processing. To render Malayalam text that uses MLB-TTIndulekha font developed by C-DAC to Java Graphical User Interface (GUI), a mapping
from ISCII to Unicode is essential. This mapping table is given in Appendix B. Patterns for keyword identification are prepared by referring ISCII to Unicode mapping table.

Keyword Selector scans the input query and identifies primary keywords such as പേര് (aar~) (who), ചെയ്‌ത (Eth-) (which), നിയമം (eppOL) (when), etc. These primary keywords give clear indication of expected answer type. But words like സൂചി (ethRa) (how much), ചെയ്‌ത (aeth) (which), etc. are less clear about expected answer type. To solve such situation and to obtain most relevant answers for the question, additional keywords are required and they are called secondary or second level keywords. Second level keywords are also called Question Focus.

Question Focus [70] is a word or a phrase in the question that solves the question type ambiguity. This is a word close to the interrogative term that supplies additional information about the type of the expected answer. Detection of focus is important for extracting the answer from candidate paragraphs. In the absence of expected answer type, correct answer is retrieved using the noun phrase attached to the question phrase.

Working of Keyword Selector is explained with the following examples.

Example 1

എരിയോത്തെ മലയാളിക്കും വൈരസിന് കാണിയിൽത്തെങ്ങ്?

(aaraaN– malERiyuTe vaaksin kaNTupiTichchath~?)

(Who invented the Vaccine for Malaria?)

Keyword Selector scans this question against the key patterns and identifies എരിയോത്തെ (aaraaN–) (who) as the primary keyword. This primary
keyword helps us to find out the question type as ‘who’ and expected answer type as ‘PERSON’.

**Example 2**

*(Eth~ raajyakkaaraaN~ kOLaRayuTe prathividhi kaNTupiTichchath~)*

(Which country-men discovered the preventive for Cholera?)

Keyword selector identifies the primary keyword *(Eth~)* (which). But this keyword does not give any clue regarding the question or answer type. Hence it is necessary to obtain secondary keywords. In the Example 2 ‘raajyakkaaraaN~’ (country-men) is the secondary keyword. Primary and secondary keyword together solves the problem of question type identification. Here, the words *(Eth~ raajyakkaaraaN~)* (which country-men) determines the question type as ‘which’ and question focus as ‘COUNTRY’.

Primary and secondary keywords determine the type of the question and the expected answer. But to obtain the most appropriate answer, another group of keywords are required. These keywords are given more weightage than the normal keywords which are called significant keywords [71]. Normally words referring to the question object or noun phrases in the absence of question object are considered as significant. Words that refers to the object of the question is termed as Question object. In certain queries, Question focus and question objects are the same.
Example 3

Which virus causes Influenza?

\((\text{Eth~ vaiRasaaN~ inphLuvensa uNTaakkuth~?})\)

(Which virus causes Influenza?)

The keyword selector separates the primary keyword ‘\(\text{Eth~}\)’ (Which) and the secondary keyword ‘\(\text{vaiRasaaN~}\)’ (virus) from the question given in **Example 3**. For the same question, the word \(\text{inphLuvensa}\) (Influenza) is the question object. The keywords \(\text{Eth~ vaiRasaaN~}\) (which virus) gives a clear indication of both the question type and the expected answer type. But these keywords are not indicating anything about “which virus it is asking for”. To answer such a question correctly, information regarding the question object is essential.

Example 4

What kind of food can a Jauntice patient eat?

\((\text{Eth~ tharaththiluLLa bhakshaNamaaN~ manjnjjappiththa rOgi kazhikkENTath~})\)

(What kind of food can a Jauntice patient eat?)

Primary keyword – \(\text{Eth~}\) (what)

Secondary keyword – \(\text{bhakshaNamaaN~}\) (food)

Question object – \(\text{manjnjjappiththa rOgi kazhikkENTath~}\) (Jauntice patient can take)

Again in Example 4, primary and secondary keywords determine the question and answer types but they do not give any indication of exact answer.
Question Classification

Keyword Selector obtains the primary, secondary, and the significant keywords from the user query and sends them to the ‘Question Classification’ stage.

<table>
<thead>
<tr>
<th>Question Type</th>
<th>Question Focus</th>
<th>Category of Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>वर्तमान (aar~) (who)</td>
<td>Not required</td>
<td>PERSON</td>
</tr>
<tr>
<td>वर्तमान (aeth~) (which)</td>
<td>रोग (rOgam) (disease)</td>
<td>DISEASE</td>
</tr>
<tr>
<td>वर्तमान (vaiRas) (virus)</td>
<td>वजन (kaaraNam) (reason)</td>
<td>REASON</td>
</tr>
<tr>
<td>वर्तमान (aahaaram) (food)</td>
<td></td>
<td>FOOD</td>
</tr>
<tr>
<td>वर्तमान (ethRa) (how much)</td>
<td>दूर (duuram) (distance)</td>
<td>DISTANCE</td>
</tr>
<tr>
<td>वर्तमान (divasam) (days)</td>
<td></td>
<td>DAYS</td>
</tr>
<tr>
<td>वर्तमान (enthukoNt~) (why)</td>
<td>Not required</td>
<td>REASON</td>
</tr>
<tr>
<td>वर्तमान (enthaN~) (what)</td>
<td>Not required</td>
<td>DEFINITION/REASON</td>
</tr>
<tr>
<td>वर्तमान (aaraaN~) (who is)</td>
<td>Not required</td>
<td>DEFINITION/PERSON</td>
</tr>
<tr>
<td>वर्तमान (aaRkk~) (whom)</td>
<td>Not required</td>
<td>PERSON</td>
</tr>
<tr>
<td>वर्तमान (eppOL) (when)</td>
<td>Not required</td>
<td>TIME</td>
</tr>
<tr>
<td>वर्तमान (eNGine) (how)</td>
<td>Not required</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>वर्तमान (eviTe) (where)</td>
<td>Not required</td>
<td>LOCATION</td>
</tr>
<tr>
<td>वर्तमान (enn~) (when)</td>
<td>Not required</td>
<td>DAY</td>
</tr>
</tbody>
</table>

Question Classification is the task that maps a question into one of the k predetermined classes. This stage is needed to understand the expected answer type before returning an answer. Assigning question classes can be accomplished in a variety of ways. In this work a rule-based classifier is used that classifies a question into fine grained categories and their corresponding
coarse categories. Classification [72] is done for finding the type of questions and answers. The expected answer type is the semantic type of the entity expected as the answer to the question. The answer type is helpful for factual questions, but not for questions that require complex explanations.

Assigning a question type to the question is a crucial task as the entire answer extraction relies on finding the correct question type and hence the correct answer type. In some cases there are words that indicate the question type directly, i.e., “who”, “where” etc. Some of these words can represent more than one type. In such situations, other words (the words that can indicate meaning) in the question need to be considered. Table 4.3 gives a list of question categories that is decided with the help of various keywords separated in the Keyword Selector stage.

Examples

A. ‘തൊഴ്സ്’ (eppOL) (when) type Questions

These questions are of two types and always ask for a named entity DATE. There are few questions which require a day or days as the answer.

1. ഇന്നാം കുത്തിവെയ്പ് എത്തിക്കേണ്ടത്?
   
   (ennaan~ kuththiveyp~ eTukkENTath~?) (when to take the injection?)

2. ഇന്നോക്കേയാണ്‌ മാരുൻ ലഭിക്കുന്നത്?
   
   (ennoakkeyaAN~ marunn~ labhikkuth~?) (on what days medicine is available?)

3. എന്ന് മുതൽ രോഗം പകരുന്ന്?
   
   (ennu muthal rOgam pakarum?) (from which day disease will spread?)

For these three questions given above, primary keywords are എന്ന് (ennu) (when) and referring Table 4.3 the question category is determined as
‘DAY’. In this type of questions secondary keywords are not required to find out the question and answer types.

B. ‘_aspect’ (aar~) (who) type Questions

There are three types of ‘who’ questions. Details are given below.

Who Definition

The question given below expects a ‘DEFINITION’ type answer.

4. ‘_aspect’ (aar~) (who) (who was Asoka?)

The primary keyword ‘_aspect’ (aar~) (who) is used to identify the question type.

Who List

5. ‘_aspect’ (aar~) (who) (who all are participated in the cholera experiments?)

(answer type to ‘question 5’ is a ‘PERSON’ entity and it requires a list of persons. In this example the word ‘all’ (okke) is very important. The classifier uses the primary keyword ‘_aspect’ (aar~) (who) and the secondary word ‘all’ (okke) (all) to identify the answer type. Then the significant keywords (participated in the cholera experiments) aids in answer extraction.

Who factoid

6. ‘_aspect’ (aar~) (who discovered Cholera?)

(answer type to ‘question 6’ is a ‘PERSON’ entity and it requires a list of persons. In this example the word ‘all’ (okke) is very important. The classifier uses the primary keyword ‘_aspect’ (aar~) (who) and the secondary word ‘all’ (okke) (all) to identify the answer type. Then the significant keywords (participated in the cholera experiments) aids in answer extraction.
Classification module assigns PERSON category to ‘question 6’. Even though the primary keyword is the same as in ‘question 4’, answer type identified is different since there is a significant keyword \(kantupi\text{tichchath}~\) (discovered) after the secondary keyword \(kolara\) (Cholera).

C. ‘ജോണ്‍ / അരുതെ’ (aare /aaruTe) (Whom/Whose) type Questions

Answer type for these questions are either OBJ or PERSON.

7. സ്നാനിച്ച് കൊളാര അദ്യമയി വന്നത്

\(aarkkaan\text{~} kolara aadyamaayi vannath~\) (Who was affected by the cholera the first time?)

8. സെക്ഷൻ ഇളന്തീതികളിൽ വെളുത്തനു തൂക്കിയത്

\(aaruTe meInO'ththilaaN~ pareekshaNam thuTangngiyath~\)

(Under whose supervision was experiments started?)

Classifier assigns ‘OBJ’ category to question 7 and ‘PERSON’ category to question 8.

D. ‘ഇവിടെ’ (eviTe) (Where) type Questions

Answer type to this question is ‘LOCATION’.

9. ഇവിടെയാണ് കരാൽ സ്ഥിതിചെയ്യുന്നത്

\(eviTeyaaN~ karaL sthithicheyyuth~\) (Where in the human body liver lies?)

The primary keyword ‘ഇവിടെ’ (eviTe) (Where) is sufficient to identify the answer type. Then the remaining words in the question (significant words) help in answer extraction.
E. ‘aeth’ (Which) type Questions

Answer type is decided by the question focus.

10. முதலிய விருது தானெங்கு பற்றுகிறது?

(Eth~ vaiRasaaN~ phLu uNTaakkuth~?) (Which virus causes flu?)

11. என்று முலக்கோன் பாது உள்ளை அமைந்து கான்தார்?

(Eth~ sthalalthhaaN~ chikkan guniya aadyamaayi kaNTath~)

(At which place Chikunguniya was first noticed?)

12. என்று தானெங்கு காட்டு குறிக்கும்?

(Eth~ samayaththaaN~ manjnppiththam pkaruth~?)

(In which season Jauntice spreads?)

In each of these interrogatives 10, 11 and 12, underlined words determines the expected answer type. They are VIRUS, LOCATION and TIME respectively.

F. ‘enth~’ (What) type Questions

13. என்று தானெங்கு காட்டு குறிக்கும்? (enthaaN~ manjnppiththam?) (what is Jauntice?)

14. என்று கோலையறு காட்டு குறிக்கும்? (enthaaN~ kOLaRayuTe kaaraNam?)

(what is the cause of cholera?)

In the questions 13 and 14, answer types are decided by the key words and the question focus. ‘Question 13’ expects ‘DEFINITION’ type answer while 14 expects ‘REASON’ type.
G. ‘ഏത്’ (eethRa) (How Many) type Questions

15. ഏത് നിര്‍മിച്ചത് രോഗം നിലനില്ക്കുന്നോ?
   
   (ethra divasam rOgam nilanikkum?) (How many days the disease last?)

16. ഏത് ദൂരം പോകാം?
   
   (ethra dooram pOkaNam?) (How much distance to cover?)

17. ഏത് സമയം സിദ്ധാന്തം?
   
   (ethra samayam viSramikkaNam?) (How much time to take rest?)

In questions 15, 16, and 17 the underlined words is the question focus but, that itself is not sufficient to finalize the answer. Significant keywords in the question are essential to determine the exact result.

H. ‘എന്തുകാരന്‍’ (enthukoNT~) (Why) questions/ ‘എന്നാണ്’ (engninge) (How) questions

These are the easiest question words to process and answer type to these questions are REASON/DESCRIPTION.

18. എന്തുകാരന്‍ ആര്‍മിന്റടുത്ത് മാനിക്കുന്നു?
   
   (enthukoNT~ eyDs~ baadhikkunnu) (Why AIDS affects?)

19. എന്താണ്‌ ചെലിക്കുന്നു കാണാം?
   
   (engngineyaaN~ kOLaRa pakaruth~) (How Cholera spreads?)

Primary keywords in the questions 18 and 19 are used to determine the question type and classifier assigns their answer types. This information along with the significant keywords is sent to the answer extractor.
As explained above, Question Analysis Module scrutinizes the input question and extracts all the information that can aid the answer retrieval process. This module also finds out the question and answer types. Knowledge that collected and derived from the input is then transferred to the Answer Extraction Module which is described in the next section.

4.1.3 Answer Extraction Module

In many QA Systems answer extraction identifies the documents or paragraphs in the document set that are likely to contain an answer [73]. Also in the absence of indexing scheme the search engine scans every document in the corpus which would take a considerable amount of time and computing power. Answer Extraction Module in Fig 4.1 is redrawn and shown below.

![Diagram of Answer Extraction Module](image)

**Fig 4.5 Answer Extraction Module**
In this work, function of Answer Extraction Module is to produce exact responses rather than paragraphs. This aim is facilitated by the four stages of this module namely Search Engine, Sentence Retrieval, Ranking Module, and Answer Retrieval.

**Search Engine**

A search engine’s job is to retrieve and present links to information as relevant to search query as possible.

**Suppose the input query is** अरान-मन्जनपिभथच-कान्तीचिचघ- (Who discovered Jaundice?)

Question Analysis stage analyses the above question and identifies the primary keyword अरान (aar~) (who). This information is further used, to determine the question type अर (aar~) (who) and expected answer type ‘PERSON’.

The search engine refers the index table given in Fig 4.3 and finds the named entity ‘PERSON’ in the entity list E. For every entry in the entity list E there is a corresponding entry in the D list. The D list entry shows that the ‘PERSON’ entity is present in the documents D1, D2, D4, D9 and D109. This list of document numbers is fed to the Sentence Retrieval stage.

Search Engine stage uses first level index. First level index contains Named Entities and pointer to documents where that entity exist.

**Sentence Retrieval**

The sentence retrieval stage retrieves the sentences using the second level index. Second level index contains Named Entity and pointer to sentences. The entries in the document list D are used for identifying entity lists and respective sentence lists. This sentence list is used for retrieving candidate sentences from
the documents. For the example mentioned above, the document numbers 1, 2, 4, 9 and 109 are used for identifying entity lists N1, N2, N4, N9, and N109 and sentence lists O1, O2, O4, O9, and O109. By referring these lists all the sentences are retrieved in which the entity ‘PERSON’ is present. From these five documents, 16 sentences are retrieved which are ranked by the ranking module.

**Ranking Module**

Significant keywords are used for ranking the retrieved sentences. The sentences retrieved are weighted according to the presence of these significant words or terms. For Example, sentence in which 6 significant terms are present is assigned a weight 6; a sentence with 3 terms is assigned a weight 3, and so on. Then these sentences are assigned a rank based on the weight. The top ranked 5 sentences are selected and used for the final answer retrieval.

**Answer Retrieval**

This final component of Answer Extraction Module selects the answer from the top ranked sentence. The word or phrase who’s named entity matches with the expected answer type is selected as the answer.

In short, Answer Retrieval Module identifies candidate sentences by searching through the prior prepared double level index using the facts obtained from the Question Analysis stage. These sentences are ranked based on various factors and top ranked sentence is selected as the final candidate. With the help of expected answer type answer is derived from the candidate sentence.

Consider the question

**කොළcdr විශේෂජනාධිපති අංකීය? (karaL ssthithicheyunnath– eviTe)**

(where does Liver exist?).
For the above question, sentence retrieval stage retrieves the following five sentences.

**Table 4.4 Outputs of Various Stages of Answer Extraction Module**

<table>
<thead>
<tr>
<th>Sentences Retrieved</th>
<th>Entity Identified</th>
<th>Weight</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. മാസാൻ‌ടെയ്യോഡിൽ ദോഷാവശേഷം ഗണിതാളക്കിലായി വിശ്വസിക്കുന്നു. ദോഷാവശേഷത്തിന്റെ അനുസരണം കൃണി ബിട്ടിയാമ്പാ.</td>
<td>മാസാൻ‌ടെയ്യോഡിൽ ദോഷാവശേഷം ഗണിതാളക്കിലായി വിശ്വസിക്കുന്നു.</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2. ഒരു പൊതുജനങ്ങൾക്കും മെയിൻ-പോസ്റ്റിലെ സെലക്ഷൻ ബിന്ദുകളും സാമൂഹിക സംവിധാനങ്ങളും വിശ്വസിക്കാനാകും.</td>
<td>ഒരു പൊതുജനങ്ങൾക്കും മെയിൻ-പോസ്റ്റിലെ സെലക്ഷൻ ബിന്ദുകളും സാമൂഹിക സംവിധാനങ്ങളും</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>3. വിദ്യാഭ്യാസ മേഖലയിലെ അഭിനായക ഉദ്യോഗത്തിന്റെ വിശ്വസിക്കാനാകുമ്പോലെ ടെൽഡി ബിൻ്ദുകളും അവസാനം കാണാമ്പാ.</td>
<td>വിദ്യാഭ്യാസ മേഖലയിലെ അഭിനായക ഉദ്യോഗത്തിന്റെ വിശ്വസിക്കാനാകുമ്പോലെ</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>4. കടലാക്കിയ മാസാൻ‌ടെയ്യോഡിൽ ദോഷാവശേഷം ഗണിതാളക്കിലായി വിശ്വസിക്കുന്നു.</td>
<td>കടലാക്കിയ മാസാൻ‌ടെയ്യോഡിൽ ദോഷാവശേഷം ഗണിതാളക്കിലായി</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>5. മാസാൻ‌ടെയ്യോഡിൽ ദോഷാവശേഷം ഗണിതാളക്കിലായി (മാസാൻ‌ടെയ്യോഡിൽ ദോഷാവശേഷം മേഖലയിലെ അഭിനായക ഉദ്യോഗത്തിന്റെ വിശ്വസിക്കാനാകും)</td>
<td>മാസാൻ‌ടെയ്യോഡിൽ ദോഷാവശേഷം ഗണിതാളക്കിലായി</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Sentences retrieved, their weights and ranks assigned are given Table 4.4. Sentences with zero rank (no significant words present) are assigned a bigger number as Rank. In this case it is 100. From the top ranked sentence (rank=1), the phrase corresponding to the expected named entity ‘LOCATION’ is separated and it is displayed as the answer. Screen shot for the above example is given in Fig 4.6. A few screenshots are given in Appendix I.
4.2 Chapter Summary

Architecture of MaQAS is depicted in this chapter. Indexing Module preprocesses all Malayalam documents and prepares a double level index. Question Analysis Module analyses the user query and understands the type of the expected answer. Then Answer Extraction Module searches through the index and determines the location of candidate sentences. These sentences are retrieved from the NE tagged documents and rank them. The Named Entity which is expected as the result is extracted from the top ranked sentence and returned to the user.
Understanding each and every word or sentence of the documents in the corpus is essential to find an exact answer. Malayalam being an agglutinative language most of the words in a Malayalam document are compound words. The meanings of compound words are not usually available in a dictionary or in a catalog. This fact shows how important a compound word splitter is!

Compound word is a word composed of two or more words either in the closed form, hyphenated form or in an open form [74]. Compounding is a common phenomenon in Malayalam, Bengali, Greek etc. Owing to this compounding or agglutinative nature 80-85% words in Malayalam documents are compound words. Hence a compound word splitter is essential to split these words into their components to deduce the word meaning.

IR and QA, two major tasks of NLU, require extensive knowledge of the compound words to have effective communication between computers and human. Compound word splitter is also an important preprocessing tool in NLP applications.
5.1 Malayalam Compound Word

In Malayalam language, a compound word might contain any number of
prefixes, postfixes, suffixes, verbs, nouns, dhyodhakam, pronouns, adjectives,
adverbs and qualifiers. Unlike English, Malayalam compound words exist in
the closed form which does not contain spaces or other word boundaries.

Examples

ആസൂത്രണൈവഭവേണടു കൂടിയ (aasUthraNavaibhavaTHOTukUTiya) (with planning capability)

This Malayalam word is a combination of 5 atoms of different categories.

noun + noun + suffix + verb + dhyodhakam

(The write-up is to be renewed)

verb + verb + suffix + dhyodhakam + Aux

Malayalam has a large number of compound words. All these are
obtained by suitably combining the 10 atom types mentioned above [66]. A
few examples are given below.

1) Verb + Verb = Verb

heard (heard) + study(study) = study by hearing

(kETTu + paTiKaNam = kETTu paTiKaNam)

2) Adverb + Verb = Verb
3) Noun + Noun = Noun

പന (palm tree) + കുരു (nut) = പനംരു (Seed of palm tree)

(Pana + kuru = panankuru)

4) Adjective + Noun +Verb + postfix = Verb

കറു + ആന + ഓടി + പാ = കറുയാനേയാടി (black elephant ran away)

(KaRuTHa + aana + OTi + pOyi = kaRuTHayaanayOTipoyi)

5.2 Methods for Compound Word Splitting

To split the compound word, different methods are available.

5.2.1 Most Probable Word Technique

Find all possible ways of breaking the compound word (C) into pieces and choose the one with highest probability [75].

C = അപകാട്ടിലാകുന്നും (ApakaTatthilaaKumennum)

(Will also be put into trouble)

Some possible ways of breaking this word include

1) അപകാടം കലം അക്ക അക്ക അക്ക അക്ക അക്ക

   (apa kaTam il aakka um enn um)

2) അപകാടം കലം അക്ക അക്ക അക്ക അക്ക

   (apakaTam il aakka um enn um)

3) അപകാടം കലം അക്ക അക്ക അക്ക അക്ക
If the components are taken as $S_1$ to $S_N$ and the corresponding probability as $Pr(S_1)$ to $Pr(S_N)$ then the overall probability of the string is $Pr(S_1...S_N) = Pr(S_1) \cdot Pr(S_2) \cdot \ldots \cdot Pr(S_N)$. Here the probabilities are estimated by counting the occurrences in a selected corpus. The second case has the highest probability as probability of പകടം (apakaTam) is greater compared to the probability of പ (apa). Also the probability of ആക (aaKa) is greater than that of ലാ (lak) in the selected corpus. This approach produces many more entries, unnecessarily increasing the search time.

5.2.2 N-gram Technique

The “N-gram” technique [76] is one approach to languages such as Japanese, Chinese and Korean which use no spaces between words. It requires less programming work on the back end and requires no dictionary or linguistic rules. N-gram segmentation works by chopping text into segments n-characters long, usually two to three characters. When “n” is two, a sliding window isolates strings of two-characters long from the input text. Using n-gram technique the above given word can be segmented into bigrams as follows: പകടം എക്കര ലാ പനാ കടിയം പകടം എക്കര ലാ പനാ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ മക്കൽ (apaka kata Taththa ththi ila laa aakka kku uma me manna nnu um). The number of “false words” created by this segmentation process is relatively high and also in this example number of genuine words is nil. This method gives many outputs but poor result.
5.2.3 Longest Match Technique

The basic idea is to search for the longest parts of the word (from left to right). On a given string, a longest string-matching is first performed from left hand side. Let the given compound word be $S$. Then find the longest string $X$ in the dictionary that is a prefix of $S$. Break this string off the beginning. Repeat this process until there is no such string in which case the remainder of the string is the last subcomponent [74].

Example: $\text{ImcWfp} \rightarrow \text{mhmsa\textbackslash n\textbackslash e\textbackslash p\textbackslash w (kaaraNangaLunTaavaamenkilum)}$

(though there may be reasons)

This word is split into sub-components as given below.

$\text{kaara} \\ \text{Nam} \\ \text{kaL} \\ \text{uNT} \rightarrow \text{aava} \\ \text{aam} \\ \text{engkil} \\ \text{um}$

(kaaraNam kaL uNT~ aava aam engkil um) where aava is not the correct sub-component.

5.2.4 Baseline Technique

A simple approach is to choose the splits with as few components as possible or selecting the split with the longest first component in case of a tie [77]. This approach works quite well giving an accuracy of 85% on the ambiguous compounds in our corpus.

Example: $\text{scampdnbn\textbackslash pÅbmfmWv (perumaRiyiTTLayaaLaaNa~)}$

(is a person who behaved)

Two possible splits of above word is given below

1. $\text{peruma aaRi itt uLLa aaL aaN~}$

( peruma aaRi itt uLLa aaL aaN~ )
2. There are many possible combinations for this compound word. If we consider the above two combinations with 6 components each, baseline method selects the second suggestion since it has the longest first component.

5.2.5 Finite State Transducer (FST)

FST uses morphological analysis, requires a dictionary and the intelligence to recognize features of the language: punctuation, actual words, word forms and affixes. For the non-weighted FST, a simple morphological parsing method is used with the grammar rules where atoms are being classified by their functionality.

The decomposition algorithm and simple FST methods try to find substrings of the given words. The process looks for all valid decompositions of the given word.

eg. சம்பவக் குழியைப் பாறை (nETiyeTukkukayumaavaNam) (it should be achieved)

The decompositions obtained are

1. எப்பிட்டு குண்டு கை ஊக எடு தோட்டம்

   (nETi eTu~ kk uka um aa aNam)

2. டோது குண்டு கை ஊக எடு தோட்டம்

   (nET iya eTu~ kuka um aavaNam)
3. To choose the correct interpretation some other mechanism is to be used.

5.2.6 Ad Hoc Rules

During the formation of compound words sometimes agamasandhi (augmentation) [66] must be used and there will be addition of consonants ‘ɔ’ (ya), ‘Alternate’ (ma), or ‘Va’ (va). But this will occur with certain word classes only. Same way if lopasandhi (elision) [66] is used certain consonants or vowels will be removed. Hence by checking the presence or absence of these letters, correct interpretation of the suggestion can be made.

e.g. പേരാവൻതൻ പനടിയിൽ കൂകിയാവ അനം (enniviTaNgaLilayEK) (to the places mentioned)

1. പേരാവൻ പനടിയിൽ കൂകിയാവ അനം

(enna iviTam kaL ila ae kk)

2. പേരാവൻ പനടിയിൽ കൂകിയാവ അനം

(enna iviTam kaL il ae kk)

Here the ambiguity is resolved by checking the presence of ‘ɔ’ (ya) after ‘Alternate’ (la). For the first combination, the compound word would be

പേരാവൻ പനടിയിൽ കൂകിയാവ അനം

(enna iviTam kaL ila ae kk = enniviTaNGaLilayEyK)

The sub-steps of forming the compound word are given below

a) പേരാവൻ + പനടി = പേരാവനപനടി (enna + iviTam = enniviTam)

b) പേരാവനപനടി + കൂകി = പേരാവനപനടി (enniviTam + kaL = enniviTangngaL)
Chapter 5

c) എന്നിവിടച്ചെളി + എല്ലാ = എന്നിവിടച്ചുപേരിമ

\[(\text{enniviTangngaL + ila = enniviTangngaLila})\]

d) എന്നിവിടച്ചുപേരിമ + അൽ = എന്നിവിടച്ചുപേരിമെല്ലാ

\[(\text{enniviTangngaLila + ae = enniviTangngaLilayE})\]

e) എന്നിവിടച്ചുപേരിമെല്ലാ + കെ = എന്നിവിടച്ചുപേരിമെല്ലാകെൽ

\[(\text{enniviTangngaLilayE + kk = enniviNaGaLilayEyK})\]

This compound word obtained is different from the original compound word since \((
 redeemed + 
 recorded = 
 recorded)\) \((
 ila + ae = ilayE)\)

The second group of elements gives

\[ 
	ext{ആന്ത് + ബോജ് + കെ + ലഠി + എല്ലാ + കെ = എന്നിവിടച്ചുപേരിമെല്ലാകെൽ} 
\]

\[(
\text{enna + iviTam+ kaL+ il + ae + kk = enniviNaGaLilEyK})\]  

since \((
 
 recorded + 
 recorded = 
 recorded)\) \((
 il + ae = ilE)\)

5.2.7 Hybrid Method

Since the above methods make errors on the interpretation of different compound words, various methods are combined to obtain higher accuracy than individual methods. The hybrid system developed combines the baseline technique with POS and Ad Hoc rules. This method has an accuracy of 92% on the ambiguous compounds.

ംണിച്ചുകരയുന്നവരും (thiricchukonTuvaraanumuLLa)

(to bring back also)

1. മണി ചെ ചെ കൊത്താറു വയ്ക്ക ഈെന പെടു (8)

\[(
	hirai i chch konTu~ var~ aan um uLLa)\]
In the previous example, 4 suggestions are given. (2) and (4) have less number of components. Hybrid method takes combinations (2) and (4) and checks the tagging of each component. ‘തിരി’ (thiri) can be a noun (candle) or a verb (turn) and the suffix ‘ച്ച’ (chch) is possible only with a verb. Same way there is ambiguity between വര (vara) and വര~ (var~). വര (vara) means line. വര~ means come. But suffix ‘ഞൻ’ (aan) can be used only with a verb. Hence the ambiguity is solved and (4) is selected as the correct interpretation.

5.2.8 Weighted FST

Weighted FSTs (WFST) are like ordinary FSTs with input and output symbols on arcs but they contain in addition a weight on every arc and every final state. These weights are combined during traversal of the automation to compute a weight for each path. To calculate weights various possibilities are tried.

- Number of segments
- Assigning probability to each grammar rule
- Component frequency
1) Compound segment with equal weight

In this method decomposition with less number of segments is selected as the correct morphological analysis [78]. To implement this technique equal weights are assigned to each segment.

With a segment weight 0.25 example given above gives overall weight as follows.

\[<0.25 \times 0.25 \times 0.25 \times 0.25 \times 0.25 \times 0.25 \times 0.25> = <0.000061025>\] (1)

\[<0.25 \times 0.25 \times 0.25 \times 0.25 \times 0.25 \times 0.25 \times 0.25> = <0.0002441>\] (2)

\[<0.25 \times 0.25 \times 0.25 \times 0.25 \times 0.25 \times 0.25 \times 0.25> = <0.000061025>\] (3)

2) Assigning probability to each grammar rule

FST uses grammar rules or morphological information to split the compound words. When many decompositions are possible, probability of grammar rule is considered to resolve the ambiguity [79].
3) Component Frequency

Given the count of words in a corpus, the splits with highest geometric mean frequency of its parts $P_i$ is chosen. The more frequent a word occurs in a training corpus the bigger the statistical basis to estimate translation probabilities [80].

Component frequency is used for the calculation of probability of occurrence of each word. During the analysis, (6) is selected as the correct interpretation as $\text{കാരണ}$ ($kaaraNam$) has a higher frequency than $\text{കാര}$ ($kara$) in (4), also $\text{ആ}$ ($aa$) in (6) has been referred to more times than $\text{ആവ}$ ($aava$) in (4) and (5).

Accuracy comparison of above mentioned methods are given in table 5.1.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most probable</td>
<td>70%</td>
</tr>
<tr>
<td>N-gram</td>
<td>25%</td>
</tr>
<tr>
<td>Longest Match</td>
<td>80%</td>
</tr>
<tr>
<td>Finite State Transducer</td>
<td>90%</td>
</tr>
<tr>
<td>Hybrid Method</td>
<td>92%</td>
</tr>
<tr>
<td>Weighted FST</td>
<td>98%</td>
</tr>
</tbody>
</table>

Table 5.1 Accuracy Comparison of Compound Word Splitting Methods
5.3 Compound Word Analyzer for Indian Languages

T. N. Vikram and Shalini R [81] developed a prototype of morphological analyzer for Kannada language based on Finite State Machine. This is just a prototype based on Finite State Machines and can simultaneously serve as a stemmer, Part-of-Speech tagger and spell checker. This analyzer has the capability to handle around 7000 distinct words from 500 distinct noun and verb stems. But it is far from being full-fledged morph analyzer as pronoun and adjective morphology have not been included in this work. Also it does not handle compound formation.

Shambhavi B. R and Dr. Ramakanth Kumar (2011) [82] developed a paradigm based morphological generator and analyzer for Kannada language using a trie based data structure. The disadvantage of trie is that it consumes more memory as each node can have at most ‘y’ children, where y is the alphabet count of the language. As a result it can handle up to maximum 3700 root words and around 88K inflected words.

MORPH- A network and process model for Kannada morphological analysis/generation was developed by K. Narayana Murthy (2001) [83] and the performance of the system is 60 to 70% on general texts.

Ramasamy Veerappan et.al [84] explains the working of MAG- a system for morphology analyzer and generator. Morphological analyzer was developed using FST. This project was developed as a part of MT system from English to Kannada. This model includes a lexicon, morphotactic, and orthographic rules. It handles only simple morphology.

Uma Maheshwar Rao G. and Parameshwari K. of CALTS, University of Hyderabad (2010) [85] attempted to develop a morphological analyzer and generators for South Dravidian languages.
Kiranmai.G et.al [86] presented a morphological analyzer for the classical Dravidian language Telugu using machine learning approach. The developed morphological analyzer is based on sequence labeling and training by kernel methods, it captures the non-linear relationships and various morphological features of Telugu language in a better and simpler way. This approach is more efficient than other morphological analyzers which were based on rules. In rule based approach every rule is depends on the previous rule. So if one rule fails, it will affect the entire rule that follows. Regarding the accuracy this system significantly achieves a very competitive accuracy of 94% and 97% in case of Telugu Verbs and nouns.

[87] describes the development of an open-source morphological analyzer for Bengali language using finite state technology. The morphological analyzer/generator was developed as a part of a new language pair for Apertium; hence the analyzer data conforms to Apertium's data format. The data for the analyzer/generator is contained in the Bengali monolingual dictionary. This morphological analyzer could also be used as a stemmer for any search engine for Bengali language. It could also double as a spell checker.

For Bengali, unsupervised methodology is used in developing a morphological analyzer system [88] and two-level morphology approach was used to handle Bengali compound words. This paper introduces a simple, yet highly effective algorithm for unsupervised morphological learning for Bengali, an Indo–Aryan language that is highly inflectional in nature. When evaluated on a set of 4,110 human-segmented Bengali words, our algorithm achieves an F-score of 83%, substantially outperforming Linguistica, one of the most widely-used unsupervised morphological parsers, by about 23%.

Rule based Morphological Analyzer were developed for Oriya and Sanskrit languages. [89] deals with the analysis and design of Oriya Morphological Analyzer (OMA). The major contents on which this OMA has
been built up with i) Pronoun Morphology, ii) Inflectional Morphology and iii) Derivational Morphology. The OMA system is designed according to the object oriented approach to increase its reusability, robustness and extensibility. It also provides sufficient interface for applications involved in Oriya Machine Translation (OMT), Word-Net for Oriya (OriNet), Oriya Spell Checker and Oriya Grammar Checker.

[90] describes a Sanskrit morphological analyzer that identifies and analyzes inflected noun-forms and verb-forms in any given sandhi-free text. This system checks and labels each word as three basic POS categories - subanta, tinanta, and avyaya. Thereafter, each subanta is sent for subanta processing based on an example database and a rule database. The verbs are examined based on a database of verb roots and forms as well by reverse morphology based on Paninian techniques.

Authors of [91] describe a morphological analyzer for Marathi language that is a paradigm based inflectional system combined with FSMs for modeling the morphotactics. This morphological analyzer used a lexicon and inflection rules for all paradigms. The accuracy obtained was as high as 97.18%. This analyzer could not handle Derivational Morphology and compound words.

In Tamil language, the first step towards the preparation of morphological analyzer for Tamil was initiated by Anusaraka group. Ganesan (2007) [92] developed morphological analyzer for Tamil to analyze CIIL corpus. Phonological and morphophonemic rules are taken into for building morphological analyzer for Tamil. Resource Centre for Indian Language Technological Solutions (RCILTS) -Tamil has prepared a morphological analyzer (Atcharam) for Tamil. Finite automata state-table has been adopted for developing this Tamil morphological analyzer [93].

Parameswari.K (2010) [94] developed a Tamil morphological analyzer and generator using APERTIUM tool kit. This attempt involves a practical
adoption of \textit{lttoolbox} for the modern standard written Tamil in order to develop an improvised open source morphological analyzer and generator. The tool uses the computational algorithm Finite State Transducers (FST) for one-pass analysis and generation, and the database is developed in the morphological model called word and paradigm.

Vijay Sundar Ram R et.al (2010) [95] was designed Tamil Morphological Analyzer using paradigm based approach and Finite State Automata, which works efficiently in recursive tasks and considers only the current state for having a transition. In this approach complex affixations are easily handled by FSA and in the FSA, the required orthographic changes are handled in every state. In this approach, they built a FSA using all possible suffixes, categorize the root word lexicon based on paradigm approach to optimize the number of orthographic rules and use morpho-syntax rules to get the correct analysis for the given word.

Akshar bharati et.al (2001) [96] developed an algorithm for unsupervised learning of morphological analysis and generation of inflectionally rich languages. This algorithm uses the frequency of occurrences of word forms in a raw corpus. They introduce the concept of “observable paradigm “by forming equivalence classes of feature-structures which are not obvious. Frequency of word forms for each equivalence class is collected from such data for known paradigms. This method only depends on the frequencies of the word forms in raw corpora and does not require any linguistic rules or tagger. The performance of this system is dependant on the size of the corpora.

M.Ganesan (2007, 2009) [92] explained about a rule-based system for the analysis and generation of Tamil corpora. Successive split method is used for splitting the Tamil words. Initially all the words are treated as stems. In the first pass these stems are split into new stems and suffixes based on the similarity of the characters. They are split at the position where the two words
differ. The right substring is stored in a suffix list and the left substring is kept as a stem. In the second pass, the same procedure is followed, and the suffixes are stored in a separate suffix list.

Uma Maheswar Rao (2004) [85] proposed a modular model that is based on a hybrid approach which combines the two basic primary concepts of analyzing word forms and Paradigm model. This architecture involves the identification of different layers among the affixes, which enter into concatenation to generate word forms.

Menon et.al (2009) [97] developed a Finite State Transducer (FST) based morphological analyzer and generator. They have used AT &T Finite State Machine to build this tool. The system is based on lexicon and orthographic rules from a two level morphological system.

Deepa. S.R et.al [98] presented an algorithm for Hindi compound word splitting. This algorithm has shown a correct split rate of about 85%. A trie like structure is used to store and efficiently match the words. Also few modifications are suggested to handle compound words.

The work presented in [99] is a morphological analyzer for Hindi language. This uses the SFST (Stuttgart Finite State Transducer) tool for generating the FST. A lexicon of root words is created. Rules are then added for generating inflectional and derivational words from these root words. The morph analyzer developed was used in a Part Of Speech Tagger based on Stanford POS Tagger. It was tested with about 4000 inflectional, derivational and compound words and gives approximately 97% correct results.

**5.4 Description of Malayalam Compound Word Splitter**

A compound word M is decomposed into its constituents M1 to Mi. To find each constituent, the longest match method is adopted. When one component, Mj, is separated, the remaining portion is sent to the Addition/
Deletion algorithm. The component $M_i$ is first searched in the lexicon and if it is not found, Transformation algorithm is applied to obtain various forms of $M_i$ and again searching is carried out. Even after transformation, if not found in the lexicon, process is repeated with next smaller substring of $M$. Working of Transformation and Addition Deletion algorithms are explained in Table 5.2.

**Methodology: Finite State Transducer (FST)**

Formally, a Finite State Transducer $T$ is a 6-tuple $(Q, \Sigma, \Gamma, I, F, \delta)$ such that:

- $Q$ is a finite set, the set of states;
- $\Sigma$ is a finite set, called the input alphabet;
- $\Gamma$ is a finite set, called the output alphabet;
- $I$ is a subset of $Q$, the set of initial states;
- $F$ is a subset of $Q$, the set of final states; and
- $\delta$ is the transition relation.

FST is a machine which accepts a string and translates it into another string. FST can also be used for generating and checking sequences [100].

A compound word is a sequence of Malayalam characters. To split this sequence into substrings FST can be used.

Compound word splitter uses an FST with the following definition.

In the 6-tuple, set of states $Q$ = \{A, B, C, D, E, F, G, H\}
Initial state $I$ = \{A\}
Final states $F$ = \{C, D, E, F, G, H\}
Input alphabet $\Sigma$ = \{compound words\}
Output alphabet $\Gamma$ = \{valid simple Malayalam words\}
Transition function $\delta$ = \{NOUN, VERB, ADJECTIVE,\ldots, SUFFIX\}
Table 5.2 Examples of Transformation and Addition/Deletion Algorithms

<table>
<thead>
<tr>
<th>Compound Word</th>
<th>Component to be transformed</th>
<th>Transformation</th>
<th>Output of Transformation Algorithm</th>
<th>Component to be modified</th>
<th>Addition/Deletion</th>
<th>Op of Add/Del Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. കൽനെയ്തോണി + കൃണം + കല്ലാലി</td>
<td>കല്ലാലി</td>
<td>ചെ</td>
<td>കല്ലാലി</td>
<td></td>
<td>Delete ചെ</td>
<td>Add ചെ</td>
</tr>
<tr>
<td>2. നടന്നുപറയാൻ + മെ വി</td>
<td>മെ</td>
<td>വി</td>
<td>മെ</td>
<td></td>
<td>Delete വി</td>
<td>Add വി</td>
</tr>
<tr>
<td>3. പിലായ് + പിലായ് + പിലായ്</td>
<td>പിലായ്</td>
<td>പിലായ്</td>
<td>പിലായ്</td>
<td></td>
<td>Delete പിലായ്</td>
<td>Add പിലായ്</td>
</tr>
<tr>
<td>4. പിലായ് + പിലായ് + പിലായ്</td>
<td>പിലായ്</td>
<td>പിലായ്</td>
<td>പിലായ്</td>
<td></td>
<td>Delete പിലായ്</td>
<td>Add പിലായ്</td>
</tr>
<tr>
<td>5. ലാണം = ലാണം + പിലായ്</td>
<td>ലാണം</td>
<td>No change</td>
<td>ലാണം</td>
<td></td>
<td>Delete പിലായ്</td>
<td>Add പിലായ്</td>
</tr>
<tr>
<td>6. ലാണം + പിലായ് + പിലായ്</td>
<td>ലാണം</td>
<td>No change</td>
<td>ലാണം</td>
<td></td>
<td>Delete പിലായ്</td>
<td>Add പിലായ്</td>
</tr>
<tr>
<td>7. ലാണം = ലാണം + പിലായ്</td>
<td>ലാണം</td>
<td>No change</td>
<td>ലാണം</td>
<td></td>
<td>Delete പിലായ്</td>
<td>Add പിലായ്</td>
</tr>
</tbody>
</table>
The machine starts at state A and repeats the following process till the word size becomes zero. At the current state, word type of the input word is matched against the labels on the outgoing arcs and if a match is found, cross that arc and move to the next state. The word whose match is found is separated from the compound word and the remaining portion is taken as the input word to the next state. If no match is found, longest match algorithm is used to obtain the next longest word and continue the same process. Likewise, all possible combinations are tried and still if no match is found go to the next state without making any changes to the input word.

Referring to Fig 5.1, at state A all prefixes present consecutively are separated from the incoming word W, and the remaining portion of W or new W is sent to state B. At this state, consecutive adverbs or adjectives or qualifiers are removed. Then automation enters into state C. If a verb or a noun or a pronoun is present in the word W, it is separated and either enters into state D (if the match is a verb) or E. At state D, search for postfix is carried out and at E
comparison is made for suffix/dhyodhakam. At state H if word size is zero (x=0) process stops or else goes to the initial state.

**Table 5.3 State Table for the FST in Fig 5.1**

<table>
<thead>
<tr>
<th>Present State</th>
<th>Prefix</th>
<th>Adj</th>
<th>Adv</th>
<th>Quaf</th>
<th>Noun</th>
<th>Verb</th>
<th>Postfix</th>
<th>P.Noun</th>
<th>Suffix</th>
<th>Dhyo.</th>
<th>Any Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>Error</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>Error</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>E</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>F</td>
<td>G</td>
<td>E</td>
</tr>
<tr>
<td>F</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>F</td>
<td>G</td>
<td>F</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>F</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>H</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>H</td>
</tr>
</tbody>
</table>

e.g. ആസൂത്രാനാവിഭഭവാധിനാഷ്ടം (aasUthraNavaibhavaTHOTukUTiya) (with planning capacity)

Working of compound word splitter for this input word is explained below.

**Step1**

Input word at state A is ആസൂത്രാനാവിഭഭവാധിനാഷ്ടം (aasUthraNavaibhavaTHOTukUTiya) and no match is found at this state against the list of PREFIX. Hence automation goes to state B.
Step 2

Same word is processed at state B. Lists of ADJECTIVES, ADVERBS and QUALIFIERS are compared with the compound words of longest to smallest size. Again no match is found hence goes to state C.

Step 3

Above word is checked against the lists of NOUN, VERB and PRONOUN. As mentioned before, various possibilities are compared and the word ആസൂത്രണം (aasUthraNam) is separated from this compound word ആസൂത്രണൈവഭവേഗാടുകൂടിയ (aasUthraNavaihavaTHOTukUTiya). When the longest word ആസൂത്രണ (aasUthraNa) is taken for comparison, different combinations are tried and an identical entry is located only for the word ആസൂത്രണം (aasUthraNam). Various alterations of words are obtained by the transformation algorithm whose working is explained in table 5.2. Automation continues at state E with the remaining word എവിഹവേഗാടുകൂടിയ (vaibhavaTHOTukUTiya).

Step 4

From state E process goes to state H. As the word size is not 0 at the present state next state is A as given in table 5.3.

Above process is repeated 3 times detaching ആസൂത്രണം (aasUthraNam) in the first cycle വേഗാടു (vaibhavam) and ഹി (Out) in the second cycle കൂടി (kUTi) and എ (a) in the third cycle.
The compound word splitter splits the word ആസൂത്രണത്തേംകൂടിയ (aasUthraNavaihbavaTHOTukUTiya) into five components as follows.

ആസൂത്രണം + നൽകേം + സു + കൂടി + അ (aasUthraNam + vaibhavam + Oat + kUTi+ a)
(noun + noun + suffix + verb + dhyodhakam)

The traversal path of the machine during the decomposition of the above compound word is A-B-C-D-E-H-A-B-C-E-F-H-A-B-C-D-E-G-H.

Fig 5.2 Output of Compound Word Splitter

Fig 5.2 shows the result of compound word splitter for the compound word സുരക്ഷിതമാക്കാനാവില്ല (surakshithamaakkaanaavillE). Output of splitter for this word is displayed as

ജലാവില്ല (sarakshitham aakka aan aa illa E)
Table 5.4 Examples of Compound Words and Components

<table>
<thead>
<tr>
<th>Malayalam</th>
<th>Sub components</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compound Words</strong></td>
<td><strong>Verb</strong></td>
</tr>
<tr>
<td>(pattika+il+u+leT+p+uN+u+aN+)</td>
<td>ḍhṛvī (p进去)</td>
</tr>
<tr>
<td>(inam+k+a+leT+u+k+h+i)</td>
<td>ḍhṛvī (p进去)</td>
</tr>
<tr>
<td>(thaTasam+t+ii+k+k+u+k+aN+)</td>
<td>ḍhṛvī (p进去)</td>
</tr>
<tr>
<td>(kalA+k+t+i+o+u+u+u+aN+)</td>
<td>ḍhṛvī (p进去)</td>
</tr>
<tr>
<td>(vaiNya+k=thu+v=saN+uN+l+k+tt+i)</td>
<td>ḍhṛvī (p进去)</td>
</tr>
</tbody>
</table>

*Note: The table continues with similar entries.*
A few examples of compound words and their constituents obtained by the compound word splitter are listed in Table 5.4.

5.5 Performance Evaluation

The FST for compound word splitter was tested for about 4000 words and 96% of the splits gave correct result. For testing the system, words are selected from different corpuses and are analysed. Then a list of compound words is prepared. The corpuses used are

- Malayala Manorama: compounds related to current affairs (Editorial page) (year 2008).
- Mathrubhumi: compounds from the weekly (year 2008).
- Vanitha fortnightly.

Even though the selected lists are independent, 90% of compound words are common in all the three lists.

5.6 Chapter Summary

Importance of compound word splitting in the field of QA and various methods of the same are comprehended and learned. FST method is comparatively superior to all other methods which gave an accuracy of 96%. Complexity of Malayalam compound words is well studied and a FST based compound word splitter is developed.
Part-of-Speech Tagging is an essential and important pre-processing step in many NLP systems. Tagged corpora play a significant role in Machine Translation, Information Retrieval, Data Mining and QA Systems. In this QA System, POS Tagging is needed for document pre-processing and index preparation. Each document in the corpus is tokenized and tagged with an appropriate POS tag. POS is one of the important features used in the NE Recognition and Classification.

POS Tagging, also called grammatical tagging, is a principal issue in Natural Language Processing. The purpose of this task is to assign part-of-speech or other lexical class markers to each and every word in a document. In English there are eight parts-of-speech such as noun, verb etc. But POS tags vary according to the language and application [100].

Since 100 B.C humans are aware that language consists of several distinct parts called POS. Those POS play a crucial role in many fields of linguistics. POS is based on both its definition and its context or relationship with adjacent and related words in a phrase, sentence or paragraph. Part-of-speech Tagging is harder because some words can represent more than one POS at different times [101]. The
significance of part-of-speech for language processing is the large amount of information they give about a word and its neighbour.

6.1 Related Work

The first major corpus of English for computer analysis was Brown corpus [100] which was tagged with POS markers over many years. In the mid 1980s researchers in Europe began to use Hidden Markov Model (HMM) [102] to disambiguate POS. In 1987 Steven J. DeRose [103] and Kenneth Ward Church [104] independently developed algorithms to solve this problem.

A lot of work has been done in Parts-of-Speech Tagging of western languages. These taggers mostly are implemented using stochastic or rule-based methods. In [105] author projects the use of Transformation-Based Learning (TBL) method to bootstrap the POS annotation results of English POS Tagger by exploiting the POS information of the corresponding Vietnamese words. The basic tagging step is achieved through the available POS Tagger (Brown) and the correction step is achieved through the Transformation-Based Learning method in which the information on the corresponding Vietnamese is used through available word alignment in the English Vietnamese parallel Corpus (EVC). A genetic algorithm for POS implementation is presented in [106] which correctly tag 93.8% of sentences using bigrams and word POS frequencies. This system is able to integrate statistical and rule-based approaches into one system. Eric Bril described a rule-based POS Tagger which automatically acquires rules and tags [101].

Rule-based techniques, and finite state machine morphological dictionary are some of the techniques used in ANUBHARATI [107], ANUVADHINI Machine Translation (MT) system for Bengali [108], and
Tamil spell checker [109] developed at Anna University, Chennai. In Tamil spell checker, morphological dictionary is internally represented by a set of FSTs that are automatically generated from a more general dictionary containing morphological syntactic information. A Punjabi spell checker [110] has been developed using Rule cum Dictionary-based method. It is a dictionary with search algorithms, which search string matching, fuzzy search, suffix stripping etc. Morphological stripping methods and paradigm-based FSTs are some other techniques used in Text Analyzers. In Indian languages Natural Language Processing tools are less as compared to English and other European languages. In Hindi a rule-based POS Tagger developed by IIT, Bombay, has been used in stemmer and morphological analyzer for Word Net project [111]. In Tamil, Anna university- K B Chandrasekhar (AU-KBC) research centre developed a morph analyzer and a POS Tagger [112]. But it is a rule-based system and its accuracy is less, and also showed word sense disambiguation problem for machine translation system [113].

Automatic Part-of-Speech Tagging is an area of Natural Language Processing where statistical techniques have been more successful than rule-based methods. Rule-based Taggers have many advantages over these taggers including vast reduction in stored information, the perspicuity of small set of meaningful rules, ease of finding and implementing improvements to the tagger and better probability from one tagset corpus genre or language to another. Stochastic approaches have often been preferred to rule-based approaches because of their robustness and their automatic training capabilities.

Smriti Singh et.al [114] have proposed a tagger for Hindi, that uses the affix information stored in a word and assigns a POS tag using no contextual information. By considering the previous and the next word in the verb group, it
correctly identifies the main verb and the auxiliaries. Lexicon lookup was used for identifying the other POS categories.

Hidden Markov Model based tagger for Hindi was proposed by Manish Shrivastava and Pushpak Bhattacharyya [115]. The authors attempted to utilize the morphological richness of the languages without resorting to complex and expensive analysis. The core idea of their approach was to explode the input in order to increase the length of the input and to reduce the number of unique types encountered during learning. This in turn increases the probability score of the correct choice while simultaneously decreasing the ambiguity of the choices at each stage.

Nidhi Mishra and Amit Mishra [116] proposed a Part-of-Speech Tagger for Hindi Corpus in 2011. In the proposed method, the system scans the Hindi corpus and then extracts the sentences and words from the given corpus. Also the system search the tag pattern from database and display the tag of each Hindi word like noun tag, adjective tag, number tag, verb tag etc.

Based on lexical sequence constraints, a POS tagger algorithm for Hindi was proposed by Pradipta Ranjan Ray (2003) [117]. The proposed algorithm acts as the first level of POS Tagger, using constraint propagation, based on ontological information, morphological analysis information and lexical rules. Even though the performance of the POS tagger has not been statistically tested due to lack of lexical resources, it covers a wide range of language phenomenon and accurately captures the four major local dependencies in Hindi.

Sivaji Bandyopadhyay et.al (2006) [118] came up with a rule-based chunker for Bengali which gave an accuracy of 81.64 %. The chunker has been developed using rule-based approach since adequate training data was not available. The list of suffixes has been prepared for handling unknown words.
They used 435 suffixes; many of them usually appear at the end of verb, noun and adjective words.

For Bengali, Sandipan et al., (2007) [119] have developed a corpus based semisupervised learning algorithm for POS tagging based on HMMs. Their system uses a small tagged corpus (500 sentences) and a large unannotated corpus along with a Bengali morphological analyzer. When tested on a corpus of 100 sentences (1003 words), their system obtained an accuracy of 95%.

Hammad Ali [120] developed a Baum-Welch trained HMM tagger for the Bangla language. Training is performed in order to learn the underlying HMM parameters. This HMM could then be used to perform tagging on a corpus and tested to obtain the accuracy figures. The complete work was divided into the following phases: 1) collect corpus and tagset, 2) search for implementation of Baum-Welch algorithm, 3) perform training and 4) test against gold standard for accuracy.

Debasri Chakrabarti [121] proposed a rule based Parts-of-Speech tagger for Bangla with layered tagging. There are four levels of Tagging which also handles the tagging of multi verb expressions. In the first level ambiguous basic category of a word is assigned. Disambiguation rules are applied in the second level with more detail morphological information. At the third level multi word verbs are tagged and the fourth or the final level is the level of local word grouping or chunking.

A rule based part-of-speech tagging approach was used for Punjabi [122]. This tagger uses handwritten linguistic rules to disambiguate the part-of-speech information, which is possible for a given word, based on the context
information. A tagset for use in this part-of-speech tagger has also been devised to incorporate all the grammatical properties that will be helpful in the later stages of grammar checking based on these tags.

Sreeganesh implemented a rule-based tagger for Telugu [123]. In the initial stage Telugu morphological analyzer analyses the input text. Then tagset is added. Around 524 morpho-syntactic rules do the disambiguation.

Avinesh et.al [124] proposed a Telugu tagger with a performance of 77.37%. In [125] three taggers are developed with accuracies 98.0%, 92.1%, and 87.8% respectively.

A SVM based POS tagger is explained in [126]. Multiclass SVM is used for classification. The tagset contained only 8 tags and overall accuracy obtained was 86.25% for passages which contains unknown words.

Approach presented in the paper [127] is a machine learning model. It uses supervised as well as unsupervised techniques. It uses a CRF to statistically tag the test corpus. The CRF is trained using features over a tagged and untagged data. Initially a rule based tagging code is run on the test data. This code used both machine learning and rule based features for tagging. It gave an accuracy of 86.43%. Then a CRF tool is used to test the data. It gave an accuracy of 89.90%.

A morphology driven Manipuri tagger is explained in [128]. This tagger was tested with 3784 sentences containing 10917 unique words. This tagger showed accuracy of 69%. Further this tagger is improved using CRF and SVM [129] and obtained accuracies of 72.04% and 74.38% respectively.
Antony P J and Soman KP [130] of Amrita University, Coimbatore proposed statistical approach to build a POS tagger for Kannada language using SVM. They have proposed a tagset consisting of 30 tags. The proposed POS tagger for Kannada language is based on supervised machine learning approach. The Part-of-Speech tagger for Kannada language was modeled using SVM kernel.

A rule-based POS tagger for Tamil was developed and tested by Dr.Arulmozhi P et.al [131]. This system gives only the major tags and the sub tags are overlooked during evaluation. A hybrid POS tagger for Tamil using HMM technique and a rule based system was also developed [132]. Parts of speech tagging scheme, tags a word in a sentence with its parts of speech. It is done in three stages: pre-editing, automatic tag assignment, and manual post-editing. In pre-editing, the corpus is converted to a suitable format to assign a POS tag to each word or word combination. Because of orthographic similarity one word may have several possible POS tags. After the initial assignment of possible POS, words are manually corrected to disambiguate words in texts.

Kathambam attaches parts of speech tags to the words of a given Tamil document. It uses heuristic rules based on Tamil linguistics for tagging and does not use either the dictionary or the morphological analyzer. It gives 80% efficiency for large documents, uses 12 heuristic rules and identifies the tags based on gender, tense and case markers. Standalone words are checked with the lists stored in the tagger. It uses ‘Fill in rule’ to tag ‘unknown words. It also uses bigram for identifying the unknown word using the previous word category.

Lakshmana Pandian S and Geetha T V (2009) [133] developed CRF Models for Tamil Part of Speech Tagging and Chunking. This method avoids a fundamental limitation of Maximum Entropy Markov models (MEMMs) and
other discriminative Markov models. The Language models are developed using CRF and designed based on morphological information of Tamil.

Selvam and Natarajan (2009) [134] have developed a rule based morphological analyzer and POS Tagger for Tamil. They improved the above systems using Projection and Induction techniques. Rule based morphological analyzer and POS tagger can be built from well defined morphological rules of Tamil. They applied alignment and projection techniques for projecting POS tags, and alignment, lemmatization and morphological induction techniques for inducing root words from English to Tamil. Categorical information and root words are obtained from POS projection and morphological induction respectively from English via alignment across sentence aligned corpora. They generated more than 600 POS tags for rule based morphological analysis and POS tagging.

6.2 Malayalam POS Tagging

POS Tagging is the process of identifying and labelling each word in a sentence with corresponding POS. Fixed word order languages like English have specific structure for a sentence. Therefore POS assignment in these languages can be done using the grammar rules. Malayalam is a free word order language; hence words can appear in any order in a sentence. But within a phrase, words are in a specific related order.

Since most of the words in a Malayalam document are compound words decomposition of these words into their constituents is extremely necessary for finalizing their POS tag. Sometimes more than one morphological analysis and hence more than one POS may occur for a single word. A correct resolution of this kind of ambiguity for each occurrence of the word is crucial in QA Systems. A large percentage of words also show ambiguity regarding lexical
category. Hence to determine the POS of a word in Malayalam language, grammar rules themselves are not sufficient but both word and contextual levels of information are necessary.

Currently available tagsets for other languages are only giving importance to the morphological and syntactical properties of the language while the tagset developed considers the semantic features of the language. In many other languages, they have used tagset derived from Penn Tree Bank [100] or Brown corpus for POS Tagging. But these tagsets are not sufficient for POS Tagging in Malayalam. MaQAS needed a POS tagset which gives importance to semantics in its development.

6.2.1 POS Tagset for Malayalam

A tagset with 52 tags is developed by manually tagging different documents from various newspapers and various fields. This tagset contains POS tags necessary for Information Retrieval and Question Answering tasks in Malayalam language. POS tagset developed for this work is given in Table 6.3. The method used to find the POS tags and a few tagging examples are given below.

Noun

In this system nouns without suffixes and nouns with gender or plural suffixes are considered as ‘NOUN’ whereas the Penn tagset, a standard tagset used for English, makes distinction between noun singular, noun plural, common nouns, proper nouns etc. Nouns which are acting as agents are given the tag, NOUN which has three subcategories proper noun, pronoun and noun.

Example 1: കുടിച്ചു (kuTTi paal kuTichchu) (child drank milk)
In this example, the words, both child and milk, are nouns without suffixes. But the noun, child, is the agent of the action and milk is the object of the action; therefore ‘NOUN’ tag is assigned to child and ‘OBJECT’ (OBJ) tag is assigned to milk.

Example 2. കുടിച്ചു പാല കെട്ടി പാല കെട്ടിച്ചു (kuTTikaL paal kuTichchu) (children drank milk)

In example 2 the word ‘children’ (noun with plural suffix) is also assigned the POS tag ‘NOUN’.

**Noun with Case Suffix**

When a noun occurs in a sentence with case suffix, POS tag of that noun is determined by the associated case. Here the case suffix changes the role of the word in the sentence. Case indicates a relation between noun and a verb [67]. Relation indicated by each case suffix is different. Hence separate POS tags are assigned to each unique case which are shown in Table 6.1.

**Examples**

- റാജ്യ ആമ്മ പിന്തുണ പോയി (raajuvinte amma pOyi) (Raju’s mother has gone)
- റാജി അമ്മയ തിന്നി (raaju ammaye aTichchu) (Raju slapped his mother)
- റാജ അമ്മോട്ട സാമ്മടി (raaju ammayOT samsaarichchu) (Raju talked to his mother)
- റാജി അമ്മക്ക് സാരി ബാംഗ് (raaju ammaykk saari vaangi) (Raju bought a saree for his mother)
- റാജി മതി മകൻ തിന്നി (raaju vaTiyaal makane aTichchu) (Raju beat his son with a stick)
- കമല അമ്മയുടെ സാരി ബാംഗ് (kamala ammayuTe saari vaangi) (Kamala got her mother’s saree)
- കമല ദാവത്തിൽ സ്വരാജീ ബാംഗ് (kamala daivaththil aaSravichchu) (Kamala depended on God)
Table 6.1 Example of Case Relations

<table>
<thead>
<tr>
<th>Case Relation</th>
<th>suffix</th>
<th>Example</th>
<th>POS Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominative</td>
<td>No suffix</td>
<td>അത് (amma)</td>
<td>NOUN</td>
</tr>
<tr>
<td>Accusative</td>
<td>ഓട്</td>
<td>അത്തക്ക് (ammaye)</td>
<td>ACC</td>
</tr>
<tr>
<td>Sociative</td>
<td>ക്</td>
<td>അത്തന് (amnayOT)</td>
<td>SOC</td>
</tr>
<tr>
<td>Dative</td>
<td>ന്</td>
<td>അത്തെ (ammaykk)</td>
<td>DAT</td>
</tr>
<tr>
<td>Instrumental</td>
<td>തെ</td>
<td>അത്തല്ലി (ammaayal)</td>
<td>INST</td>
</tr>
<tr>
<td>Possessive</td>
<td>ഉെട</td>
<td>അത്തെട ആട് (ammayuTe)</td>
<td>GEN</td>
</tr>
<tr>
<td>Locative</td>
<td>യ്ക്ക</td>
<td>അത്തെടി (ammyil)</td>
<td></td>
</tr>
</tbody>
</table>

All the underlined words in the above examples except the first are nouns but with case suffix. They are not agents of action but their case endings show the relationship of theirs to the agent and the main verbs.

Table 6.2 Examples of VERB Tags

<table>
<thead>
<tr>
<th>Example</th>
<th>Role of underlined word</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>രാജുവിന്റെ അമ്മയി</td>
<td>Verb</td>
<td>VERB</td>
</tr>
<tr>
<td>അമ്മയിട്ടിൽ വാണി</td>
<td>Adjectival Participle</td>
<td>AdjP</td>
</tr>
<tr>
<td>എന്നാണ് വാണി</td>
<td>Adverbial Participle</td>
<td>AdvP</td>
</tr>
</tbody>
</table>

Verbs

Finite verbs are tagged with VERB tag. Infinite verbs are divided into adjectival participle and adverbial participle. Adjectival participle relies on nouns and adverbial participle on verbs. All the three forms function distinctly [67]. Table 6.2 gives examples of VERB tags.

Examples

1. രാജുവിന്റെ അമ്മയി (raajuvinte amma pOyi) (Raju’s mother has gone)

   Here the word അമ്മയി is marked as finite verb and the corresponding POS tag is ‘VERB’.

2. വാണിയിട്ടിൽ (vaangiya pEna) (pen which was bought)
Adjectival participle വാരിയ is given the tag ‘AdjP’.

3. വരാപ്രാണി (varaan paRanju) (told to come)

The tag ‘AdvP’ is assigned to the Adverbial participle വാരിയ.

Table 6.3 POS Tags

<table>
<thead>
<tr>
<th>TAG</th>
<th>DESCRIPTION</th>
<th>TAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOUN</td>
<td>Noun</td>
<td>AdvP</td>
<td>Adverbial Participle</td>
</tr>
<tr>
<td>PN</td>
<td>Proper Noun</td>
<td>AdvT</td>
<td>Adverbial clause of Time</td>
</tr>
<tr>
<td>RN</td>
<td>Pronoun</td>
<td>AdvPl</td>
<td>Adverbial clause of Place</td>
</tr>
<tr>
<td>ACC</td>
<td>Accusative</td>
<td>AdvPr</td>
<td>Adverbial clause of Purpose</td>
</tr>
<tr>
<td>DAT</td>
<td>Dative</td>
<td>AdvR</td>
<td>Adverbial clause of Result</td>
</tr>
<tr>
<td>GEN</td>
<td>Genitive or Possessive</td>
<td>AdvSe</td>
<td>Adverbial clause of Reason</td>
</tr>
<tr>
<td>LOC</td>
<td>Locative</td>
<td>AdvCo</td>
<td>Adverbial clause of Comparison</td>
</tr>
<tr>
<td>SOC</td>
<td>Sociative</td>
<td>Adv-S</td>
<td>Adverbial clause of Supposition</td>
</tr>
<tr>
<td>INST</td>
<td>Instrumental</td>
<td>AdvC</td>
<td>Adverbial clause of Condition</td>
</tr>
<tr>
<td>OBJ</td>
<td>Object</td>
<td>AdjQi</td>
<td>Adj clause of Quality</td>
</tr>
<tr>
<td>RES</td>
<td>Reason</td>
<td>AdjQn</td>
<td>Adj clause of Quantity</td>
</tr>
<tr>
<td>PSP1</td>
<td>Postposition Type1</td>
<td>AdjD</td>
<td>Adj clause of Description</td>
</tr>
<tr>
<td>PSP2</td>
<td>Postposition Type2</td>
<td>AdjN</td>
<td>Adj clause of Number</td>
</tr>
<tr>
<td>PSP3</td>
<td>Postposition Type3</td>
<td>AdjI</td>
<td>Adj clause of Interrogation</td>
</tr>
<tr>
<td>PSP4</td>
<td>Postposition Type4</td>
<td>AdjE</td>
<td>Adj clause of Exclamation</td>
</tr>
<tr>
<td>PSP5</td>
<td>Postposition Type5</td>
<td>SourceP</td>
<td>Source-Place(Noun+PSP2)</td>
</tr>
<tr>
<td>PSP6</td>
<td>Postposition Type6</td>
<td>DestP</td>
<td>Destination-Place(Noun+PSP3)</td>
</tr>
<tr>
<td>PSP7</td>
<td>Postposition Type7</td>
<td>LikeP</td>
<td>Word Showing Similarity(Noun+PSP9)</td>
</tr>
<tr>
<td>PSP8</td>
<td>Postposition Type8</td>
<td>ListP</td>
<td>Word indicating a list(Noun+PSP11)</td>
</tr>
<tr>
<td>PSP9</td>
<td>Postposition Type9</td>
<td>TimeP</td>
<td>Word indicating Time(Noun+PSP8)</td>
</tr>
<tr>
<td>PSP10</td>
<td>Postposition Type10</td>
<td>ThruP</td>
<td>Word indicating a path or friendship(Noun+PSP5)</td>
</tr>
<tr>
<td>PSP11</td>
<td>Postposition Type11</td>
<td>SYM</td>
<td>Symbol</td>
</tr>
<tr>
<td>VERB</td>
<td>Verb</td>
<td>CN</td>
<td>Cardinal Number</td>
</tr>
<tr>
<td>AuxV</td>
<td>Auxiliary Verb</td>
<td>ON</td>
<td>Ordinal Number</td>
</tr>
<tr>
<td>AdjP</td>
<td>Adjectival Participle</td>
<td>INT1</td>
<td>Wh Words</td>
</tr>
<tr>
<td>Adv</td>
<td>Adverb</td>
<td>INT2</td>
<td>Yes/No Words</td>
</tr>
</tbody>
</table>
Postposition

This is a word used along with a noun or a pronoun to show how it is related to something else. Each postposition is assigned a different POS as they serve different roles. ‘PSP1’ to ‘PSP11’ are the tags assigned to different postpositions which are listed in table 6.4.

<table>
<thead>
<tr>
<th>Word</th>
<th>Tag</th>
<th>Word</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSP1</td>
<td>PSP2</td>
<td>PSP3</td>
<td>PSP4</td>
</tr>
</tbody>
</table>

Adjectives

An adjective is a word that adds to the meaning of the noun. It is working as a qualifier of the noun. There are different kinds of adjectives. Adjective indicates quality, number, quantity etc. The POS tag varies according to the type of the adjective. Examples are AdjQl, AdjN, AdjQn.

Adverbs

An adverb adds something to the meaning of a verb, adjective or another adverb. Adverb indicates time, condition, reason etc. POS tags corresponding to these are AdvT, AdvC, and AdvR.

Examples of Adjectives and Adverbs

Adjectives

- Adjective of Quality - AdjQ മുഴു
Adjective of Quantity - AdjQn ജന്യാം
Adjective of Number - AdjN സംഖ്യാം
Demonstrative Adjective - AdjD ഇതൊരു

Adverbs

- Adverbial clause of Time - AdvT സമയത്ത്
- Adverbial clause of Place - AdvPl സ്ഥലത്ത്
- Adverbial clause of Condition - AdvC ദയാവു
- Adverbial clause of Purpose - AdvPr ലക്ഷണം

Examples of other POS

- SourceP - സേവനമാണ്
- DestP - മാറ്റം
- LikeP - സാമ്യമാണ്
- ListP - താറാവു
- TimeP - സമയം
- ThruP - സമയസ്രോതസ്സ്

6.2.2 Part-of-Speech Tagger for Malayalam

Fig.6.1 shows the block diagram of the POS Tagger which was developed. Working of this tagger is as follows. The input document is divided into tokens. Then each token is sent to the Word Analyzer for the detailed analysis. Word Checker checks the word and determines whether it is a compound word or root word. If it is a root word, the Word Type Identifier
assigns suitable tags using the information from the lexicon. A view of lexicon, prepared for this work is given in Appendix C.
If the token is a compound word, the morphological details and the constituents of it are determined by the Compound word splitter. The Tag Marker assigns to each token, all possible POS tags with the help of information provided by the Word Analyzer and the Compound word splitter. Root Tag Detector assigns POS tags of the root words and the Word Tag Detector assigns POS tags of the compound words. If the tag is not unique or ambiguous, ambiguity is resolved by the Tag Disambiguator. Extended Conditional Random Field (ECRF) is the methodology used for the Tag Disambiguator. It uses contextual information to solve the ambiguity and eliminates all but one tag.

Tokenizer

Complete document can be assumed as a string of characters. Tokenization is the process of splitting a string of characters into lexical elements such as words and punctuations. This task is done by locating word boundaries.

Input to the Tokenizer block in Fig 6.1 is a document in Malayalam. During the tokenization process each sentence of the document is taken and splits into words or co-occurrence patterns. Multi part words are one of the main issues arise while tagging. These words are considered as single words to keep the semantic information intact for the purpose of efficient information extraction.

Word Analyzer

Word Analyzer checks each token to see whether it is present in the lexicon or not. Lexicon has all the root words along with its POS information. If it is present in the lexicon then it is a simple word and the word is labelled with POS details retrieved from the lexicon. Else the token is a compound word and it is labelled with <CW> tag.
Table 6.5 Output of Word Analyzer for Example 3

<table>
<thead>
<tr>
<th>Token</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>കമല (kamala)</td>
<td>&lt;Noun&gt;</td>
</tr>
<tr>
<td>അത്യുട (ammayuTe)</td>
<td>&lt;CW&gt;</td>
</tr>
<tr>
<td>സാരി (saari)</td>
<td>&lt;Noun&gt;</td>
</tr>
<tr>
<td>വാങ്ങി (vaangi)</td>
<td>&lt;CW&gt;</td>
</tr>
</tbody>
</table>

**Example 3** കമല അത്യുട സാരി വാങ്ങി (kamala ammayuTe saari vaangi)
(Kamala got her mother’s sari)

For this sentence, Word Analyzer produces the output shown in Table 6.5.

**Tag Marker**

Tag Marker uses the information provided by the Word Analyzer and marks the token with the most appropriate POS tag. It is designed with a Finite State Transducer as shown in Fig 6.2. Tokens marked with <CW> tags are sent to the Compound word splitter one by one and then receive the constituents of the words along with their POS information. Next the Tag Marker assigns suitable tags to each compound word based on the constituents. Sometimes there will be different valid decompositions possible for a compound word and in such cases the word will be marked with multiple tags.

For the FST in Fig 6.2 following elements form the tuple.

Initial state – A

Final states - {B, C,...,Z}

Input Alphabet - {NOUN, VERB..., DHYODHAKAM}

Output alphabet - {Any POS from the tagset}

Transition function - {any valid POS}
Example

For the word നഗരിൽക്ക് (nagaraththiEkk) (towards the city), Compound word splitter produces the following output.

നഗരം<NOUN> ഇല<C6> ഏക്ക<PSP4>

(nagaram<NOUN> il <C6> aek<PSP4>)

Then the FST for the Tag Marker takes <NOUN> <C6> <PSP4> as the input string and by traversing the path A-B-D-E produces the output ‘LOC’. i.e., the POS for the word നഗരിൽക്ക് (nagaraththiEkk) is ‘LOC’. Output of Tag Marker for a few examples are given in table 6.6.
Table 6.6 Output of Tag Marker

<table>
<thead>
<tr>
<th>Word</th>
<th>POS string</th>
<th>Traversal Path</th>
<th>POS tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>குட்டி (kutti) (child)</td>
<td>Noun</td>
<td>A-B</td>
<td>NOUN</td>
</tr>
<tr>
<td>செரியாக்குட்டி (cheRiyakutti) (small child)</td>
<td>Adj + Noun</td>
<td>A-L-B</td>
<td>NOUN</td>
</tr>
<tr>
<td>செரியாக்குட்டியுடைத்தெ (cheRiyakuttiyuTe) (of the small child)</td>
<td>Adj + Noun + Suffix-C5</td>
<td>A-L-B-N</td>
<td>GEN</td>
</tr>
<tr>
<td>செரியாக்குட்டியோடுத்துகுடுத்தெ (cheRiyakuttiyOTukUTe) (with the small child)</td>
<td>Adj + Noun + Suffix-C2 + PSP5</td>
<td>A-L-B-C-M</td>
<td>ThruP</td>
</tr>
</tbody>
</table>

Tag Disambiguator

Tag Marker assigns each input token a POS tag or multiple tags. Tokens with multiple tags are sent to the Disambiguator to solve the tag ambiguity which removes all tags except one. Output of Tag Disambiguator is a string of all tokens along with their POS tags.

Fig 6.3 Graphical structure of chain-structured CRFs

Methodology

Tag Disambiguator is implemented using High Order Conditional Random Field or Extended CRF. It is an undirected graphical model in which each vertex represents a random variable whose probability distribution is to be inferred and each edge represents a dependency between two random variables [135][136].
Considering Fig 6.3, \( X = \{X_1 \ldots X_N\} \) and \( Y = \{Y_1 \ldots Y_N\} \) are two sets of random fields. For the given input sequence \( X \), \( Y \) represents a hidden state variable and CRF’s define conditional probability distributions \( P(Y|X) \) over the input sequence. Sometimes the conditional dependency of each \( Y_i \) on \( X \) will be defined through a fixed set of feature functions (potential functions) of the form \( f(i, Y_{i-1}, Y_i, X) \). The model assigns each feature a numerical weight and combines them to determine the probability of a certain value for \( Y_i \). CRF’s can contain any number of feature functions and the feature function can inspect the entire input sequence \( X \) at any point during inference. CRF’s are extended into high order models by making each \( Y_i \) dependent on a fixed number of previous variables \( Y_{i-o} \ldots Y_{i-1} \).

POS Tagging can be modelled as a sequence labelling task where \( X = X_1 X_2 X_3 \ldots X_n \) represents an input sequence of words and \( Y = Y_1 Y_2 Y_3 \ldots Y_n \) represents corresponding POS label sequence. The general label sequence \( Y \) has the highest probability of occurrence for the word sequence \( X \) among all possible label sequences, that is, \( Y = \text{argmax} \{\Pr(Y|X)\} \). Referring Fig 6.3, a word \( X \) can be assigned any one of the labels \( Y_1 \) to \( Y_n \) and the relation between \( X \) and these labels are determined by the various feature functions. The relation which computes to the maximum probability is the best choice and hence corresponding label or tag is selected.

Main features for POS Tagging have been identified based on the word combination and word context. Following are the features used for POS Tagging in Malayalam.

- Constituents of current word: These determine the POS tag of the word as noun, verb etc.
• Context word features: Preceding (pw) and following words (nw) of the current word. pw1, pw2, pw3, nw1, nw2, nw3 are the features selected for this work.

• POS information: POS of previous words and successive words

• Digits or symbols present: If the word contains digits they are marked with ‘NUMBER’ POS (cardinal number (CN) or ordinal number (ON)).

• Lexicon feature: It contains Malayalam root words and their basic POS information such as noun, verb, adjective, adverb etc.

• Inflection lists: After analysing various classes of words inflection lists of nouns, verbs, and participles are prepared to improve the performance of the POS Tagger.

Example: വെള്ളം ഉടു. (avan veLLamunT uTuththu)

(He wore white Dhoti.)

Word Analyzer checks each word of this sentence and produces the following output.

avan <NOUN> veLLamunT <CW> uTuththu <CW>

Then the Tag Marker sends the last two words with the <CW> tag to the Compound word splitter which decomposes these words into its constituents as shown below.
The Tag Marker finalizes the POS tag of the word ഉടു (uTuththu) as ‘VERB’. The word വെള്ളംതു (veLLamunT) can be a verb or a noun as the POS of the compound word is based on the POS of the tail component of the compound word. This situation is solved by the Disambiguator. For the Disambiguator, input sentence represents X and corresponding POS represents Y. According to the principles of CRF, each POS Yi, is dependant on the corresponding word of the sentence X. But in Malayalam language, Yi is not only based on Xi but also on certain other features mentioned above.

**Probability Calculation**

In the sentence അവൻ  വെള്ളംതു ഉടു (avan veLLamunT uTuththu) the word വെള്ളംതു (veLLamunT) has two POS tags VERB and NOUN. For this sentence Tag Marker produces the following output.

\[
\text{അവൻ} < \text{NOUN}> \text{  വെള്ളംതു} < \text{VERB}/ \text{NOUN}> \text{ുടു} < \text{VERB}>
\]

\[
(\text{avan} < \text{NOUN}> \text{ veLLamunT} < \text{VERB/NOUN}> \text{uTuththu} < \text{VERB}>)
\]

Table 6.7 gives possible POS tags of successive and preceding words of VERB and that of a NOUN in question.
Table 6.7 Tagging Features

<table>
<thead>
<tr>
<th>VERB</th>
<th>NOUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>POS i-1 = adv</td>
<td>POS i-1 = adj</td>
</tr>
<tr>
<td>POS i-1 = noun</td>
<td>POS i-1 = adjP</td>
</tr>
<tr>
<td>POS i-1 = advP</td>
<td>POS i-1 = noun</td>
</tr>
<tr>
<td>POS i-1 = verb</td>
<td>POS i-1 = noun</td>
</tr>
<tr>
<td>POS i-1 =PSP</td>
<td>POS i+1 = verb</td>
</tr>
<tr>
<td>POS i+1 = AuxV</td>
<td>POS i+1 = adv</td>
</tr>
<tr>
<td>POS i+1 = verb</td>
<td>POS i+1 = advP</td>
</tr>
<tr>
<td>Probability of POSi-1 = 1/5</td>
<td>Probability of POSi+1 = 1/3</td>
</tr>
<tr>
<td>Probability of POSi+1 = ½</td>
<td>Probability of POSi+1 = 1/6</td>
</tr>
</tbody>
</table>

Considering the above example

\[ P(\text{VERB} \mid \text{veLLamunT}) = \text{sum of the probabilities of the features} \]

\[ = \text{probability of the preceding word} + \text{probability of the following word} \]

\[ = 1/5 + 0 = 1/5 = 0.2 \quad (1) \]

Similarly \[ P(\text{NOUN} \mid \text{veLLamunT}) = 1/3 + 1/6 = 1/2 = 0.5 \quad (2) \]

\[ P(Y \mid X) = \text{maximum of (1) and (2) and hence POS label NOUN is assigned to} \]

\[ \text{the word veLLamunT.} \]
When this same word occur in a sentence അവിെടെവിളമുെത്ത് (aviTe veLLamunT) Disambiguator calculates the conditional probability P (verb| veLLamunT) = 1/5 and P (noun| veLLamunT) = 0 and then finalizes the POS label as VERB. Above two examples of Tag Disambiguation are summarized in Table 6.8.

Table 6.8 Example of POS Tag Disambiguation

<table>
<thead>
<tr>
<th>Word</th>
<th>Sentence</th>
<th>Probability Calculation</th>
<th>Max Value</th>
<th>POS Tag</th>
</tr>
</thead>
</table>
| അവിെടെവിളമുെത്ത് (veLLamunT) | ശാപവെല്ലാമുെത്ത് (avan veLLamunTuTuTHu) | P (VERB| veLLamunT) = 1/5+ 0  
                                           | = 1/5 =0.2 |         |
|                       | അവിെടെവിളമുെത്ത് (aviTe veLLamunT)       | P (NOUN | veLLamunT) =  
                                           | 1/3+1/6=1/2=0.5 | 0.5     | NOUN    |
|                       |                                               | P (verb| veLLamunT) = 1/5 =0.2 |           | 0.2     | VERB    |
|                       |                                               | P (noun| veLLamunT) = 0        |           |         |         |

E.g. നമൂനയ വിവാഹം കഴിെത്ത്നഗരിൽെക്ക് മടീവണ്ണിലെ രാമ അച്ഛെനിെയാന്ന വനവാസിൽ പായി | (vivaaham kazhinj nagaththilEkk maTangivanna raaman achchhante niyOgaththaal vanavaasaththinu pOyi)

(Raman who returned to city after marriage, went to lead forest-life by the instruction of his father)

Table 6.9 lists the outputs of Word Analyzer, Compound word splitter and Tag Marker for sentence given above. Fig 6.4 shows the output of POS Tagger for the same sentence.
<table>
<thead>
<tr>
<th>Stage 1 Output of Word Analyzer</th>
<th>Stage 2 Output of Compound Word Splitter</th>
<th>Stage 3 Output of Tag Marker</th>
</tr>
</thead>
<tbody>
<tr>
<td>വിവാഹം&lt;NCW Noun&gt;</td>
<td>വിവാഹ&lt;NOUN&gt;</td>
<td></td>
</tr>
<tr>
<td>കഴി&lt;verb&gt; വാ&lt;suffix&gt;</td>
<td>കഴി&lt;AdvT&gt;</td>
<td></td>
</tr>
<tr>
<td>നഗരം&lt;Noun&gt; ഇ&lt;suffix&gt;</td>
<td>നഗരം&lt;LOC&gt;</td>
<td></td>
</tr>
<tr>
<td>മടൗിവാ&lt;AdjP&gt;</td>
<td>മടൗിവാ&lt;AdjP&gt;</td>
<td></td>
</tr>
<tr>
<td>അച്ഛ&lt;Noun&gt; സാ&lt;suffix&gt;</td>
<td>അച്ഛ&lt;GEN&gt;</td>
<td></td>
</tr>
<tr>
<td>നിയോഗാല&lt;RES&gt;</td>
<td>നിയോഗാൽ&lt;RES&gt;</td>
<td></td>
</tr>
<tr>
<td>വനവാസം&lt;DAT&gt;</td>
<td>വനവാസം&lt;DAT&gt;</td>
<td></td>
</tr>
<tr>
<td>പ്പായി&lt;VERB&gt;</td>
<td>പ്പായി&lt;VERB&gt;</td>
<td></td>
</tr>
<tr>
<td>(vivaaham&lt;NCW Noun&gt;)</td>
<td>(vivaaham&lt; NOUN&gt;)</td>
<td></td>
</tr>
<tr>
<td>(kazhinj &lt;CW&gt;)</td>
<td>(kazhinj &lt;AdvT&gt;)</td>
<td></td>
</tr>
<tr>
<td>(nagathhilEkk &lt;CW&gt;)</td>
<td>(nagathhilEkk &lt;LOC&gt;)</td>
<td></td>
</tr>
<tr>
<td>(maTangi &lt;verb&gt;vanna&lt;&lt;AdjP&gt;)</td>
<td>(maTangi &lt;verb&gt;vanna&lt;&lt;AdjP&gt;)</td>
<td></td>
</tr>
<tr>
<td>(raaman &lt;NCW Proper noun&gt;)</td>
<td>(raaman &lt;Noun&gt;)</td>
<td></td>
</tr>
<tr>
<td>(achchhante&lt;CW&gt;)</td>
<td>(achchhan&lt;Noun&gt;nte &lt;suffix&gt;)</td>
<td></td>
</tr>
<tr>
<td>(niyOgaththaal &lt;CW&gt;)</td>
<td>(niyOgam &lt;Noun&gt; aal&lt;suffix&gt;)</td>
<td></td>
</tr>
<tr>
<td>(vanavaasaththinu&lt;CW&gt;)</td>
<td>(vanavaasam&lt;Noun&gt; ni&lt;suffix&gt;)</td>
<td></td>
</tr>
<tr>
<td>(pOyi&lt;CW&gt;)</td>
<td>(pO&lt;verb&gt;I &lt;suffix&gt;)</td>
<td></td>
</tr>
</tbody>
</table>
6.3 Results and Discussion

The performance evaluation of the POS Tagger is carried out using the standardized evaluation techniques like precision and recall, where precision is defined as the ratio of correct number of token tag pair sequence in the output to total number of token tag pair that appears in the output and recall is the ratio of correct number of token tag pair sequence to the number of correct token tag pair that is possible [137].

POS Tagger results were analysed using a confusion matrix or contingency table. A contingency table is a simple performance analysis tool typically used in classification tasks and IR scenarios.
A 2*2 contingency table is shown in Table 6.10. The entries in the table have the following meaning in the context of POS labelling.

True Positive – number of correct predictions that an instance is +ve
False Positive – number of incorrect predictions that an instance is +ve
False Negative – number of incorrect predictions that an instance is -ve
True Negative – number of correct predictions that an instance is –ve

The recall or true positive rate is defined as the ratio of the number of true positives to the elements actually belonging to the positive class.

Recall = True Positive / (True Positive + False Negative)

The precision or positive predictive value is defined as the ratio of the number of true positives to the elements labeled as belonging to the positive class.

Precision = True Positive / (True Positive + False Positive)

Documents from Malayalam dailies Malayala Manorama, Mathrubhumi, Karshaka Sree, and few text books pertaining to five different fields were selected as the test corpus. These documents were texts with simple, compound and complex sentences and 95% of these words were compound words. Totally 2352 sentences were tested and obtained an average precision of about 92% and
recall of 94%. Some words were not correctly tagged since all features were not included in the system. Performance evaluation details are listed in Appendix D.

Table 6.11 Overall Performance of the POS Tagger

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Actual</th>
<th>Pos</th>
<th>Neg</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>10947</td>
<td>951</td>
<td>11898</td>
<td></td>
</tr>
<tr>
<td>Neg</td>
<td>698</td>
<td>53</td>
<td>751</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>11645</td>
<td>1004</td>
<td>12649</td>
<td></td>
</tr>
</tbody>
</table>

Average precision and recall can be obtained from the contingency table shown in Table 6.11.

\[
\text{TP} = 10947 \quad \text{FP} = 951 \quad \text{FN} = 698 \quad \text{TN} = 53
\]

Average Precision = \( \frac{10947}{(10947 + 951)} = 92.0 \% \)

Average Recall = \( \frac{10947}{(10947 + 698)} = 94.0 \% \)

- This approach is accurate and efficient for a language like Malayalam since it considers language level and word level features.

- Malayalam language has no specific structure for a sentence. Hence it is very difficult to assign POS tag for a word; many times a word will have multiple tags. This problem is solved in this system by taking into account the contextual information.

- A POS tagset is developed which is unique in its nature as it reflects morphological, syntactical and semantic features of the language.
6.4 Chapter Summary

POS Tagging is the first and one of the important stages of document preprocessing. POS tags necessary for Malayalam document tagging and their method of development is described in this chapter. A study of, POS Taggers available in other languages are done and is also given in this chapter. A detailed discussion of Malayalam POS Tagger design, development and implementation is conducted.
In this work, phrase chunking is the second phase of document preprocessing. Chunker identifies and divides sentences into syntactically correlated word groups. An Artificial Immunity System (AIS)-based phrase chunker is discussed which identifies and labels each phrase chunk with suitable phrase tag. Phrase tag is one of the important language dependant feature used for Named Entity Recognition and Classification.

QA is the task of automatically answering a question given in natural language. In order to find the answer, the question is analysed, and the type of expected answers is determined. Then the answer retrieval module retrieves either a document, passage, or a phrase as the answer to the query. Questions such as ‘when’, ‘where’, ‘what’, ‘why’ etc. mostly return an answer that contains a noun, a noun phrase or a prepositional phrase. Hence identification of phrases is an essential preprocessing step in QA Systems.

Phrase chunks can be identified and separated either by chunking or by way of full parsing. Abney [138] has proposed text chunking as a useful step for full parsing since it lays foundation for further levels of analysis. Abney describes chunking as a natural phenomenon in the following words.

“(when I read) (a sentence) (I read it) (a chunk) (at a time)”
Full parsing [139] is the syntactic analysis of a text, made up of tokens to determine the grammatical structure with respect to a given formal grammar. By this analysis it will be clear how words are combined to get phrases and phrases into sentences. Full parsing is an extremely difficult process for morphologically rich language like Malayalam. Also it is expensive and is not very robust. But chunking is more robust, efficient, faster, and sufficient for many applications.

Chunking is a shallow parsing technique also called light parsing for extracting non-overlapping segments from a stream of data [140]. A typical chunk consists of a single context word surrounded by a constellation of function words and each chunk contains a head word. Some words in a sentence may not be grouped into a chunk. A phrase chunker identifies phrases like verb phrase, noun phrase etc. in a sentence but does not either specify their internal structure or their role in the main sentence.

7.1 Related Work

The Artificial Immunity System Principle is explored in intelligent robotics system [141]. Authors of [141] have implemented a behaviour arbitration mechanism for the robots to choose the best option when an abnormal situation arises. Nasser Omer Sahel Ba-Karait et al. [142] explains the use of negative selection algorithm for the classification of Electroencephalography (EEG) signals and proved that this method reveals very promising performance. The general purpose algorithm based on the clonal selection and affinity maturation process in an adaptive immunity system is presented in [143] which is capable of solving complex engineering tasks like multimodal and combinatorial optimization. Clonal selection algorithms are applied for edge detection problems
in pattern recognition and computer vision [144]. The principle of hypermutation and receptor editing are exploited for the above purpose. Akshat Kumar and Shivashankar B Nair described the working of English Grammar checking system using adaptive immunity system [145].

In Hindi [146], Tamil [147], Telugu [148], Bengali [149] languages several works are reported but they are mainly using statistical approaches like Conditional Random Field (CRF), Hidden Markov Model (HMM) and Maximum Entropy Markov Models (MEMM). In Tamil a few taggers are developed employing Support Vector Machines (SVM) [150] and they obtained a tagging accuracy of 95.82%.

Many researchers have used various machine learning methods and their combinations for chunking. Noun phrase chunking is an important and useful task in many natural language applications and it is studied well in languages such as English [151] and French [140]. But for a less privileged language like Malayalam, Corpus-based NLP tasks are at deadlock due to the unavailability of lexicons and taggers.

[152] describe experiences in building an HMM based Part-Of-Speech (POS) tagger and statistical chunker for 3 Indian languages-Bengali, Hindi and Telugu. For chunking, the training data is used to extract chunk pattern templates defined as a sequence of POS tags. These templates, in conjunction with the POS tag of the word following the chunk, are used to decide chunk boundaries in the unannotated text. A dynamic programming algorithm is used to find the best possible chunk sequence. The chunk accuracies obtained are 67.52 for Bengali, 69.98 for Hindi and 68.32 for Telugu.
Pattabhi R.K Rao et.al [153] presents a Transformational-Based Learning (TBL) approach for text chunking. In this technique of chunking, a single base rule (or a few base rules) is provided to the system, and the other rules are learned by system itself during the training phase for reorganization of the chunks. This system is designed to work with three of the Indian languages namely Hindi, Bengali and Telugu.

In [154] authors describe a rule-based chunker that has been developed and tested on the Bengali development test set and demonstrated 85.88% accuracy. This chunker has then been tested on the Hindi and Telugu development sets and demonstrated 73.88% and 55.96% accuracies. The chunking system has demonstrated 80.63%, 71.65% and 53.15% accuracies with the unannotated Bengali, Hindi and Telegu test sets.

Paper [155] presents the building of POS Tagger and Chunk Tagger using Decision Forests and also focuses on the investigation towards exploring different methods for Parts-Of-Speech Tagging of Indian languages using sub-words as units. For chunking this system used 2-tag scheme. Features used for chunking are 2-level context of POS Tags namely present, previous, previous-previous, next and next-next word POS Tags. The two models POS Tagger and Chunk Tagger were tested with 3 different Indian languages Hindi, Bengali, Telugu and achieved the accuracies as 69.92%, 70.99%, 74.74% and 69.35%, 60.08%, 77.20% respectively.

A computational framework for chunking based on valency theory and feature structures has been described in [156]. This paper also draws an analogy of chunk formation in free order languages with the bonding of atoms,
radicals or molecules to form complex chemical structures. A chunker has been implemented for Bengali using this approach with considerably good accuracy.

A rule based model [157] is developed using 21 linguistic rules for automatic VP chunking. A 100,000 word Urdu corpus is manually tagged with VP chunk tags. The corpus is then used to develop a hybrid approach using HMM based statistical chunking and correction rules. The technique is enhanced by changing chunking direction and merging chunk and POS tags. The automatically chunked data is compared with manually tagged held-out data to identify and analyze the errors. Based on the analysis, correction rules are extracted to address the errors. By applying these rules after statistical tagging, further improvement is achieved in chunking accuracy. The results of all experiments are reported with maximum overall accuracy of 98.44% achieved using hybrid approach with extended tagset.

In this work phrase chunker is implemented using a novel approach based on Artificial Immunity System. AIS is a new branch of AI used for intelligent problem solving technique in optimization and scheduling using the theory of human immunity system. AIS algorithms are more efficient than the classical heuristic scheduling algorithms. Also they are more successful than Genetic algorithms [158]. There are three algorithms widely applied: clonal selection algorithms [143], immune network algorithms [159] and negative selection algorithms [142]. AIS theory is not yet used for the development of phrase chunker. It is easy and possible to make any modifications to an AIS-based system as this system works on the principle of generation and deletion of self/non-self patterns.
7.2 Malayalam Phrase Chunker

Phrase Chunker which is the second stage of document preprocessing module takes the output of POS Tagger and labels it with phrase tags. Clause identifier in Fig 7.1 identifies and separates the clauses from the sentences using a rule base. Then the phrase separator separates the phrases from each of these clauses and phrase tagger attaches appropriate phrase tags to them.

7.2.1 Clause Identifier

Malayalam sentences are mostly complex or compound sentences containing multiple clauses. Hence clause identification is an essential step of phrase chunking. Clause identifier identifies and separates clauses from the sentences using a rule base and gives the output to the phrase separator.
To obtain the clauses it is necessary to find out where a clause begins and where it ends. The POS tags assigned to each and every token in the sentence is used to determine these boundaries. Normally main clause ends with a verb and a subordinate clause ends with an infinite verb. This module identifies clauses with the help of handcrafted linguistic rules. A clause may end either with a verb, Auxiliary verb, Adjectival participle or an Adverbial participle.

In the rules given below, N, N+1, N+2, N+3 are the positions of the current word and the three words following the current word. Examples of the rules are:

- Rule 1: IF N+1 adjp N+2 Noun then separate clause
- Rule 2: IF N+1 adjp N+2 adjp N+3 Noun then separate clause
- Rule 3: IF N+1 advp then separate clause

7.2.2 Phrase Separator

Each clause obtained in the previous step is taken one by one and sent to the phrase separator. All the phrases corresponding to each clause is identified and separated. Then they are labelled with phrase tags.

Methodology

Phrase Separator (PS) is implemented using the principles of human immune system. There are two levels of defence mechanism in human body. First level of defence called innate immune system is available since birth and is regulated by white blood cells. Innate immune system defends the external organisms by making barriers such as mucus, low PH, saliva etc. Second level of defence is called adaptive immune system. This defence is pathogen specific and is controlled by Lymphocytes consisting of T-cells and B-cells. When a
specific pathogen enters the human body the T-cells with the specific receptors recognizes and tries to destroy them. In case, the number of cells to attack the invading pathogen is insufficient then they are multiplied by clonal selection and attack the pathogen [143].

In a phrase separator a POS tagged clause is the input. Using the phrase grammar the POS tag string can be analysed to identify the phrase chunks. In an AIS-based phrase chunker the POS tag string can be assumed as a pathogen. A pathogen can be identified by the corresponding T-cells. Since the number of valid phrase chunks identified for Malayalam language is thirty two, there are 32 different T-cells to identify the phrase chunks. T-cells which recognize the valid phrase chunks are called self cells and others are called non self cells [160]. POS patterns for phrase identification are given in Appendix E. Once T-cells are created, the system is ready to work like an Artificial Immune System. i.e. the T-cells are mature enough to detect the presence of incoming phrase chunks (pathogens). If the chunks are detected, they are separated (thrown out) from the incoming clause.

![Fig 7.2 Working of Phrase Separator](image)

Working of Phrase Separator is described in Fig 7.2. The PS module accepts a stream of words with the corresponding POS tags as input and group sub-sequences of words and tags that most likely form a phrase. Here, the phrase identifier groups every sequence and POS tag string separator separates
the POS tag string from the word-tag sequence. Every subgroup of tags is compared to every pattern in the self set. If there is no match found then the tested pattern is of non self and a process is activated to reject the incoming phrase. If the tested pattern matches any self pattern then “accept” process is activated. This action is continued until all the clauses are split into phrases.

Table 7.1 Phrase Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Descriptions</th>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>Noun phrase</td>
<td>AdvpC</td>
<td>Adverbial clause of Condition</td>
</tr>
<tr>
<td>VP</td>
<td>Verb Phase</td>
<td>AdvpRe</td>
<td>Adverbial clause of Reason</td>
</tr>
<tr>
<td>NP-Acc</td>
<td>Noun phrase-Accusative</td>
<td>AdvpS</td>
<td>Adverbial clause of Supposition</td>
</tr>
<tr>
<td>NP-Dat</td>
<td>Noun phrase-Dative</td>
<td>AdvpCo</td>
<td>Adverbial clause of Comparison</td>
</tr>
<tr>
<td>NP-Gen</td>
<td>Noun phrase-Genitive</td>
<td>AdvpQn</td>
<td>Adverbial clause of Comparison</td>
</tr>
<tr>
<td>NP-Loc</td>
<td>Noun phrase-Locative</td>
<td>AdvpQn</td>
<td>Adverbial clause of Quantity</td>
</tr>
<tr>
<td>NP-Soc</td>
<td>Noun phrase-Sociative</td>
<td>AdvpN</td>
<td>Adverbial phrase of Number</td>
</tr>
<tr>
<td>NP-Obj</td>
<td>Noun phrase-Object</td>
<td>AdvpE</td>
<td>Adjectival phrase of Emphasis</td>
</tr>
<tr>
<td>NP-Inst</td>
<td>Noun phrase-Instrumental</td>
<td>AdjpI</td>
<td>Adjectival phrase of Imperative</td>
</tr>
<tr>
<td>NP-Res</td>
<td>Noun phrase-Reason</td>
<td>AdjpD</td>
<td>Adjectival phrase of Description</td>
</tr>
<tr>
<td>AdjNP</td>
<td>Adjectival Participle of phrase</td>
<td>ThruPP</td>
<td>Noun Phrase-Path</td>
</tr>
<tr>
<td>AdVP</td>
<td>Adverbial Participle of phrase</td>
<td>ListPP</td>
<td>Noun Phrase-Group</td>
</tr>
<tr>
<td>AdvpT</td>
<td>Adverbial clause of Time</td>
<td>LikePP</td>
<td>Noun Phrase-Similarity</td>
</tr>
<tr>
<td>AdvpPr</td>
<td>Adverbial clause of Purpose</td>
<td>SourcePP</td>
<td>Noun Phrase-Source</td>
</tr>
<tr>
<td>AdvpR</td>
<td>Adverbial clause of Result</td>
<td>DestPP</td>
<td>Noun Phrase-Destination</td>
</tr>
<tr>
<td>AdvpP</td>
<td>Adverbial clause of Place</td>
<td>TimePP</td>
<td>Noun Phrase-Time</td>
</tr>
</tbody>
</table>

7.2.3 Phrase Tagger

This module assigns suitable phrase tags to the output of Phrase Separator. Phrase tags are listed in Table 7.1.

Examples

1. നി നാണ്യി പഠിക്കു ഏതെന്നാണ് മറ്റുവില്ലാകാം.

   (nI nannaayi paThichchaal ninakk fast klaas labhikkum)

(If you study well you will get first class.)
POS Tagger produces the following output for the above sentence.


This POS tagged sentence is divided into two clauses as follows.

Clause 1: 

Clause 2: 

Corresponding to Example 1 two clauses are identified and they are given as input to phrase separator. From each of those clauses, phrases are separated and labelled with phrase tags.

Input to phrase separator

 Clause 1: 

 Clause 2: 

Steps of processing clause 1

Step 1: form the POS tag string <RN><Adv><AdvC> from clause 1

Step 2: this POS pattern is checked with all entries in the self set. Since this POS tag string forms an invalid pattern no match is found and hence this phrase is rejected.

Step 3: Next the pattern <RN><Adv> is checked which is also rejected.
Step 4: Pattern <RN> is sent to the self set, match is found. As <RN> forms a valid noun phrase it is recognized by the self set and “accept” process is activated, i.e. നി (nI) <RN> is a noun phrase and this word is identified as <NP>.

Step 5: <Adv><AdvC> pattern is then sent to the test, since it is a valid phrase, chunker marks നായി പത്തിച്ച അതിനെ (nannaayi paThichaal) with phrase tag <AdvpC>.

Similarly, corresponding to clause 2, three phrases are obtained which are shown below.

ninakk <NP-Dat> ഫസ്യ സ് <NP-Obj> ലഭിഷ്മ <VP>

Phrase chunker output for the above example is listed in table 7.2.

<table>
<thead>
<tr>
<th>Clauses</th>
<th>POS Tag</th>
<th>Detected String</th>
<th>Phrase Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>നിനക്ക് &lt;DAT&gt; ഫസ്യ സ് &lt;NP-Obj&gt; ലഭിഷ്മ &lt;VP&gt;</td>
<td>DAT OBJ VERB</td>
<td>DAT OBJ VERB</td>
<td>NP AdvpC NP-Dat NP-Obj VP</td>
</tr>
</tbody>
</table>

2. (Grand father instructed all of them to get up next day early in the morning)

POS Tagger produces following output for the above sentence.

PitEnn ellaavarum veLuppinezhunnElkkaNamenn appUppan aajnjaapichchu

(pitEnn ellaavarum veLuppinezhunnElkkaNamenn appUppan aajnjaapichchu)

(Grand father instructed all of them to get up next day early in the morning)
Then the clause identifier separates the clauses as shown below.

Clause 1:

(PitEnn ellaavarum veLuppinezhunnElkkaNamenn)

Clause 2:

(appUppan aajnjaapichchu)

For the above two clauses phrase chunker produces the output as shown in Table 7.3.

<table>
<thead>
<tr>
<th>Clauses</th>
<th>POS string</th>
<th>Detected String</th>
<th>Phrase Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) േപി titre എംവുംം വുള്ളിെനഴു േ ക്കണെമ യ്യ്</td>
<td>Adv RN VERB</td>
<td>Adv VERB RN NP</td>
<td>VP NP</td>
</tr>
<tr>
<td>2) അറൂറ് ആംപിച്ചു</td>
<td>NOUN VERB</td>
<td>NOUN VERB NP VP</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.3 Phrase Chunker output for Example 2

Fig 7.3 Screen shot of Phrase Chunker
Fig 7.3 describes the output of Phrase Chunker for the sentence

3. വിവാഹം കഴിഞ്ഞ് നഗരത്തിലെ മട്ടിവം രാമൻ അച്ഛന്റെ നിയോഗത്താല്‍ വനവാസം പോയി

(vivaaham kazhinj nagaththilEkk maTangivanna raaman achchhante niyOgaththaal vanavaasaththinu pOyi)

(Raman who returned to city after marriage, went to forest according to the decision of his father)

The phrases identified are listed below in table 7.4

<table>
<thead>
<tr>
<th>Phrases</th>
<th>Phrase Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)വിവാഹം കഴിഞ്ഞ് (vivaaham kazhinj)</td>
<td>AdvT</td>
</tr>
<tr>
<td>2)വിവാഹം കഴിഞ്ഞ് നഗരത്തിലെ മട്ടിവം രാമൻ (vivaaham kazhinj nagaththilEkk maTangivanna raaman)</td>
<td>AdjNP</td>
</tr>
<tr>
<td>3)അച്ഛന്റെ നിയോഗത്താല്‍ (achchhante niyOgaththaal)</td>
<td>AdvC</td>
</tr>
<tr>
<td>4)അച്ഛന്റെ നിയോഗത്താല്‍ വനവാസം പോയി (achchhante niyOgaththaal vanavaasaththinu pOyi)</td>
<td>VP</td>
</tr>
</tbody>
</table>

7.3 Performance Evaluation

The performance of the phrase chunker was evaluated using the standardized techniques like precision, recall, and F-score where precision is defined as a ratio of number of correct chunks to the number of chunks in the output and recall is the ratio of number of correct chunks to the number of chunks in the test data.
Table 7.5 Performance Evaluation of the AIS-based Phrase Chunker

<table>
<thead>
<tr>
<th>Chunk</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>93.5%</td>
<td>92.6%</td>
<td>93.0%</td>
</tr>
<tr>
<td>VP</td>
<td>96.7%</td>
<td>97.1%</td>
<td>96.9%</td>
</tr>
<tr>
<td>NP-Obj</td>
<td>92.0%</td>
<td>92.2%</td>
<td>92.1%</td>
</tr>
<tr>
<td>NP-Dat</td>
<td>96.4%</td>
<td>97.0%</td>
<td>96.7%</td>
</tr>
<tr>
<td>NP-Acc</td>
<td>94.3%</td>
<td>95.2%</td>
<td>94.7%</td>
</tr>
<tr>
<td>NP-Gen</td>
<td>91.0%</td>
<td>90.5%</td>
<td>90.7%</td>
</tr>
<tr>
<td>NP-Loc</td>
<td>87.9%</td>
<td>88.3%</td>
<td>88.1%</td>
</tr>
<tr>
<td>NP-Soc</td>
<td>95.0%</td>
<td>95.0%</td>
<td>95.0%</td>
</tr>
<tr>
<td>NP-Inst</td>
<td>92.8%</td>
<td>91.4%</td>
<td>92.1%</td>
</tr>
</tbody>
</table>

F-score = $2 \times \text{recall} \times \text{precision} / \text{recall} + \text{precision}$ [161]. Precision, recall and F-score obtained for certain types of chunks are shown in table 7.5. Complete list is given in appendix F.

Documents related to five different fields were selected as the test corpus. All types of sentences were tested. Average precision is 91.3% and recall 90.6%. Out of 5647 phrase chunks tested, 4671 chunks gave the results correctly. 447 failure cases are identified where phrases were not in proper order and also overlapping phrases were present. To separate the phrases correctly, the word order within the phrase has to be corrected.

Table 7.6 Overall Performance of the Phrase Chunker

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pos</td>
</tr>
<tr>
<td>Pos</td>
<td>4671</td>
</tr>
<tr>
<td>Neg</td>
<td>482</td>
</tr>
<tr>
<td>Total</td>
<td>5153</td>
</tr>
</tbody>
</table>

Average precision and recall can be obtained from the contingency table shown in table 7.6.
TP = 4671   FP = 447   FN = 482   TN = 47

Average Precision = 4671 / (4671+447) = 91.3 %

Average Recall     = 4671 / (4671+482) = 90.6 %

This phrase chunker is less complex and has high speed due to the reduced number of computations. Without much difficulty, system can be updated just by updating the self/non self cell patterns.

7.4 Chapter Summary

Phrase chunking and its importance in a QA System are discussed. Various approaches to chunking, and methods adopted in different languages are described. A thorough investigation of Artificial Immune Systems is done and its application for phrase chunking is explained. A phrase tagset with 32 tags is also developed.

.......................... END ................................
Named Entity Recognition and Classification is an important preprocessing step in all NLU applications. Named Entities are words or word sequences which usually cannot be found in common dictionaries and yet encapsulate important information that can be useful for the semantic interpretation of texts. Hence they are helpful in understanding natural language documents. Support Vector Machine-based NE Tagger for Malayalam language is discussed in this chapter.

A QA System becomes most effective when it understands the question given by the user and returns the appropriate answer from the collection of documents in the corpus. Answer selection is another important aspect that requires understanding of NL documents. Both of these issues demand the identification of semantic information which can be decided by the Named Entities present in the question or texts in the corpus.

Named Entities are words or word sequences which usually cannot be found in common dictionaries and yet encapsulate important information that can be useful for the semantic interpretation of texts. The term “named entity”
now widely used in natural language processing was first introduced in 1995 by the sixth Message Understanding Conference (MUC-6) [162]. At that time MUC was focusing on the information retrieval tasks, where structured information of company and defence related activities were extracted from unstructured text such as newspaper articles. Then people noticed that references to the entities such as names including person names, location names, organization names and numerical expressions including date, time, amount etc. were essential to recognize information units for accurate information retrieval. Identifying these entities is one of the important tasks of IE and was called Named Entity Recognition and Classification. The NE task that was first introduced as a part of the MUC 6 (MUC 1995) evaluation exercise was continued in MUC 7(MUC 1998) [163]. This formulation of NE task defined 7 types of NE: PERSON, ORGANISATION, LOCATION, DATE, TIME, MONEY and PERCENTAGE.

NEs are theoretically identified and classified using phonological, morphological, semantic, and syntactic properties of linguistic forms that act as the targets of linguistic rules and operations. Two kinds of features are commonly used – internal and external. Internal features are provided from within the sequence of words that constitute the entity and external features are those that can be obtained by the context in which entities appear [164].

8.1 Related Work

Various techniques available for solving NER problems are statistical machine learning techniques, rule-based systems and hybrid approaches. Machine learning methods are using either supervised learning or unsupervised learning techniques. Statistical methods require large amount of manually annotated training data. A few commonly used statistical methods are Hidden
Markov Model, Maximum Entropy Markov Model, and Conditional Random Field. Sequence labelling problem can be solved very efficiently with the help of HMM. The conditional probabilistic characteristics of CRF and MEMM are very useful for the development of Named Entity Recognition (NER) systems. MEMM is having a label biasing problem. But all the machine learning techniques require large relevant corpuses which is unavailable in Malayalam. Machine learning methods are cost effective and do not need much of language expertise. In [165] authors describe an NER system using CRF. This system uses different contextual information of the words along with both language independent and language dependant features. GuoDong Zhou and Jian Su [166] described a HMM based on the mutual information independence assumption where they claimed that their system reached ‘near human’ performance. NER system based on MEMM is presented in [167].

Grammar-based techniques are used for creating NER systems that obtain better precision but at the cost of lower recall and months of work by experienced computational linguistics [168]. Rule-based approaches lack the ability of coping with the problems of robustness and portability. Each new source of text requires significant tweaking of rules to maintain optional performance and the maintenance cost is quite high. Rule-based systems perform the best especially for specialized applications. [169] introduces a rule-based system that used handcrafted rules and this approach gave them better performance than the CRF method.

Hybrid methods either use combinations of different machine learning methods or combinations of rule-based and machine learning methods. In [170] authors present an NE recognition tool for Portuguese. It has two components—rule-based components for recognition of number expressions and hybrid
component for names. Lot of work has been reported in the field of NER for English and European languages.

Research indicates that NER systems developed for one domain do not typically perform well on another domain.

Asif Ekbal et.al [171] reports the development of a Named Entity Recognition (NER) system for Bengali using the statistical Conditional Random Fields (CRFs). The system makes use of the different contextual information of the words along with the variety of features that are helpful in predicting the various named entity classes. A portion of the partially NE tagged Bengali news corpus, developed from the archive of a leading Bengali newspaper available in the web, and has been used to develop the system. The training set consists of 150K words and has been manually annotated with a NE tagset of seventeen tags. Experimental results of average Recall, Precision and F-Score values of 93.8%, 87.8% and 90.7%, respectively.

Bengali NER system in [172] used SVM approach for its development. Training set contained 130,000 words with 16 NE tags using BIE model for PERSON, LOC, and ORG entities. This model includes gazetteers for names of persons, locations, organization, and miscellaneous. The evaluation reported a goo F-Measure of 91.8%.

Hasan et.al (2009) [173] presented a learning-based Named EntityRecognizer for Bengali that donot rely on manually constructed gazetteers in which they developed two architectures for the NER system. The corpus consisting of 77942 words is tagged with one of 26 tags in the tagset defined by IIT Hyderabad where they used CRF++ to train the POS tagging model. Evaluation results shows that the recognizer achieved an improvement of 7.5% in F-measure over a baseline recognizer.
B. B. Chaudhuri and S. Bhattacharya [174] have proposed a three-stage approach of named-entity detection. The stages are based on the use of Named-Entity (NE) dictionary, rules for named-entity and left-right co-occurrence statistics. Corpus of Anandabazar Patrika has been used from the year 2001-2004. Experimental results have shown the average recall, precision and f-measure to be 85.50%, 94.24% and 89.51%.

Vijayakrishna and Sobha L [175] developed a domain focused Tamil NER for tourism using CRF. The tag set contained 106 tags. Morphological analysis, POS tagging, NP Chunking and NE annotation are done manually on the corpus. This corpus is divided into training data (80%) and test data (20%). CRF model is then trained with the training data. A total of 4059 NEs are tested and obtained an overall F-Measure of 80.44%.

S Lakshmana Pandian [176] describes a hybrid 3-stage NER system for Tamil. There are 3 phases namely shallow parsing, shallow semantic parsing, and statistical processing. Statistical processing uses E-M algorithm which takes inputs from both shallow parsing and semantic parsing stages. Average F-Measure values obtained is 72.72%. This system uses both linguistic and statistical methods.

[177] describes NER system for Hindi using CRF approach. The training set has been manually annotated with a NE tag set of 12 tags. The performance of the system has shown improvements by using the Part-of-Speech information of the current and surrounding words, name list, location name list, organization list, person prefix gazetteers list etc. It has been observed that using prefix and suffix feature helped a lot in improving the results. This system has achieved Precision, Recall and F-score of 72.78%, 65.82% and 70.45% respectively. CRF++ toolkit was used for training and testing data.
Saha et. al [178] describes the development of Hindi NER using MEMM approach. The training data consisted about 234K words collected from the newspaper “Dainik Jagaran”. It was evaluated against a blind test corpus of 25K words having four classes –PERSON, ORGANISATION, LOCATION, and DATE. This system is able to achieve an f-value of 81.52%, using a hybrid set of features including traditional NER features augmented with gazetteer lists and extracted context patterns.

Amit Goyal describes [179] a NER system for Hindi language using CRF approach which also identifies nested entities. This method was evaluated on test set 1 and test set 2 and attains a maximal F1 measure around 49.2 and nested F1 measure around 50.1 for test-set 1; maximal F1 measure around 44.97 and nested F1 measure 43.70 around for test-set 2 and F-measure of 58.85% on development set.

Authors of [180] describes the application of Conditional Random Fields (CRFs) with feature induction to a Hindi named entity recognition task. They discover relevant features by providing a large array of lexical test and using feature induction to construct the features that increases the conditional likelihood. Combination of Gaussian prior and early-stopping based on the results of 10-fold cross validation is used to reduce over fitting.

Gupta and Arora (2009) [181] describes the observation made from the experiment conducted on CRF model for developing Hindi NER. It shows some features which makes the development of NER system complex. It also describes the different approaches for NER. The data used for the training of the model was taken from Tourism domain and it is manually tagged in IOB format.
Biswas et.al [182] presented a hybrid system for Oriya NER that applies both ME and HMM and some handcrafted rules to recognize NEs. Firstly the ME model is used to identify the named entities from the corpus and then this tagged corpus is regarded as training data for HMM which is used for final tagging. The annotated data used in the system is in IOB format. The F-Measure obtained was between 75% and 90%.

[183] explains a Telugu NER system by using MEMM approach. The corpus was collected from various newspapers. The system makes use of the different contextual information of the words and gazetteer list was also prepared manually or semi-automatically from the corpus and came out with an F-Measure of 72.07% for PERSON, 60.76%, 68.40%, and 45.28% for ORG, LOC, and others respectively.

A CRF-based NER system is explained in [184]. This describes the working of a noun tagger. This system was trained with a manually tagged data of 13425 words and tested on a data set of 6223 words. F-Measure obtained in various experiments is between 80% and 97%. This work is only for single word NEs. No POS Tagger or Chunker information is used for this work.

In [185] authors present the experiments conducted as a part of NER for South and South East Asian Languages (NERSSEAL) Competition with various feature combinations for Telugu NER. It is also observed that the prefix and suffix information helps a lot in finding the class of the token. The best performing model gave an Fβ=1 measure of 44.91.

Authors of [186] describe a handcrafted rule-based NER for Assamese. A corpus of about 50000 words of Assamese online Pratidin article was first manually tagged. This tagged corpus is then analyzed to form the rules.
8.2 Difficulties in Finding Named Entities in Malayalam Language

NER in Malayalam language is a difficult task since it is not case sensitive as English and other European languages. Owing to its agglutinative nature most of the words are compound words and hence NE of these words cannot be determined directly. The world of NE is “open” since new Person, Location, and Organization names are coming up regularly. It is impossible to add all these NEs to a dictionary and it is a time consuming task. Another problem is that the same word can be grouped into several NE types.

8.3 Methodology- Support Vector Machines

Support Vector Machine proposed by Vapnik is a set of machine learning algorithms based on statistical methods. It is known as one of the best supervised learning algorithms and has been successfully applied to natural language tasks such as text categorization [187], phrase chunking [188], POS Tagging [189], data classification [190] etc. This algorithm analyses data and recognizes patterns used for statistical classification and regression analysis [191].

Assigning a Named Entity label to a word in the sentence can be taken as a classification problem. SVM is normally used for solving binary classification problems but can be extended to solve multiclass problems such as Named Entity Classification (NEC). Linguistic features are mapped to SVM feature vectors and setting these features to 1 if it exists or else 0 otherwise. These feature vectors are used to decide whether a word is an NE or not, and the specific class of NE to which the word belongs.
Multiclass SVM

Multiclass SVMs are usually implemented by combining several two-class SVMs. Popular methods for doing this are: one-versus-all method using Winner-Takes-All (WTA) strategy, one-versus-one method implemented by Max-Wins Voting (MWV), Directed Acyclic Graph-SVM (DAG-SVM), and error-correcting codes [192].

For a given multiclass problem, M will denote the number of classes and \( \omega_i, i = 1 \ldots M \) will denote the M classes. For binary classification the two classes are referred as positive and negative. A binary classifier will be assumed to produce an output function that gives relatively large values for examples from the positive class and relatively small values for examples belonging to the negative class.

WTA SVM

WTA SVM constructs M binary classifiers. The \( i^{th} \) classifier output function \( \rho_i \) is trained, taking the examples from \( \omega_i \) as positive and the examples from all other classes as negative. For a new example \( x \), WTA SVM strategy assigns it to the class with the largest value of \( \rho_i \).

MWV SVM

This method constructs one binary classifier for every pair of distinct classes and so, all together \( M(M-1)/2 \) binary classifiers are constructed. The binary classifier \( C_{ij} \) is trained in taking the examples from \( \omega_i \) as positive and the examples from \( \omega_j \) as negative. For a new example \( x \), if classifier \( C_{ij} \) says \( x \) is in class \( \omega_i \), then the vote for class \( \omega_i \) is added by one. Otherwise, the vote for class \( \omega_j \) is increased by one. After each of the \( M(M-1)/2 \) binary classifiers makes its vote, MWV strategy assigns \( x \) to the class with the largest number of votes.
For example, in Fig 8.1 there are three classes A, B, and C. Using MWV strategy, \( 3 \times (3-1)/2 = 3 \) binary classifiers are created namely \( C_{AB}, C_{BC}, \) and \( C_{CA} \). \( C_{AB} \) classifies word in the test data as of class A or class B by increasing the count corresponding to A or B. \( C_{BC} \) and \( C_{CA} \) repeats the same process and increases the count corresponding A or B or C. Repeating this strategy, each word in the test data is assigned a class label either as A or B or C.

**8.4 Malayalam NE Tagger**

This tagger takes a document marked with POS and phrase tags, and produces an NE tagged document. Block diagram of Malayalam NE Tagger is given in Fig 8.2. There are mainly four modules, NE marker, NE identifier, NE classifier, and NE Disambiguator. NE Marker determines whether a token is a Named Entity or not. NE identifier categorizes an entity as sole-entity, constituent-entity, or dependant-entity. If a particular token is an entity, then it is classified as one of the 26 predefined categories. Sometimes it might not be possible to tag a token with a single NE tag, and then NE Disambiguator solves the issue using the Gazetteer lists and assigns the token an NE tag.
8.4.1 NE Marker

The function of this module is to classify the incoming tokens into an NE or NOT-AN-NE (NAN). For this purpose, it checks the POS tag of each token in the input document. Words which have POS other than ‘VERB’, ‘AuxV’, and postpositions are decided as named entities and label them with ‘NE’ tag while all other tokens are assigned a ‘NAN’ tag.

8.4.2 NE Identifier

Once an ‘NE’ tag is assigned to a token, next step is to determine its type whether it is sole entity, constituent entity, or dependant entity. A word which can
act as an entity irrespective of its surroundings is called a sole entity. A word which cannot stand as a sole NE can be combined with other words to form an NE. Such words are constituent entities. Owing to the presence of certain surrounding words, a word can become an entity which is called dependant entity.

### Table 8.1 Named Entity Tagset

<table>
<thead>
<tr>
<th>NE Tagset</th>
<th>MEDIUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISEASE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>MEDICINE</td>
<td>REST</td>
</tr>
<tr>
<td>TREATMENT</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>ORGANISATION</td>
<td>SYMPTOM</td>
</tr>
<tr>
<td>PERSON</td>
<td>TIME</td>
</tr>
<tr>
<td>LOCATION</td>
<td>DATE</td>
</tr>
<tr>
<td>VIRUS</td>
<td>COLOR</td>
</tr>
<tr>
<td>PRECAUTION</td>
<td>TEST</td>
</tr>
<tr>
<td>FOOD</td>
<td>OBJECT</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>JOURNAL</td>
</tr>
<tr>
<td>REASON</td>
<td>COUNTRY</td>
</tr>
<tr>
<td>ELEMENT</td>
<td>AGENT</td>
</tr>
<tr>
<td>AGE</td>
<td></td>
</tr>
</tbody>
</table>

### 8.4.3 NE Classifier

The NE Classifier determines the NE classes of all entity tokens in the document. An NE tagset is designed for this work and it contains 26 NE tags which are listed in Table 8.1.

To classify an entity into one of these 26 tags, SVM is used. During the training phase, features are extracted from the training data and classifiers are designed for each named entity. For classification, features are extracted from the current document and are encoded as vectors. If the entity is a sole entity
internal or external features are not required for classification since the entity itself determines its NE. In the case of a constituent entity, internal features (internal to the word) are used for the classification while a dependant entity uses external features or surrounding features. The features used for NER and classification are given below.

For a constituent entity internal features are extracted to form a feature vector. Dependant entity uses a sliding window and feature extractor to obtain the external features. Once the feature vector of the input token is ready, WTA strategy of Multiclass SVM can be used to determine the NE label.

**Features used for Named Entity Identification and Classification**

In this system clues present as inner word (internal feature) and context word (external features) are used for NE identification and classification. Some of these features are Language Independent Features while others are Language Dependant Features [193].

**Language Independent Feature**

*Word Prefix/Suffix*

Prefix/Suffix information of a word and its surrounding words are useful for highly inflected language like Malayalam. Word suffix information is useful to identify the named entities. Variable length suffixes can be matched against the list of suffixes for different classes of NEs. A list of linguistic suffixes (verbs, adjectives, adverbs, nouns) is prepared which helps to recognize ‘Not a Named Entity’ cases. Certain suffixes are helpful in detecting Named Entities such as PERSON, LOCATION etc.
Chapter-8

Digit Information

This gives word level orthographic information. If a word contains digits or special symbols, corresponding feature is set to 1; otherwise 0. Three features are considered depending upon the presence of digits and/or the number of tokens, combination of digits and symbols. These features are helpful in recognizing DATE expressions, TIME expressions, AGE expressions etc.

Length of the Word

If the length of the word is greater than two, the feature ‘Length’ is set to 1; otherwise it is set to 0. Named Entities, since they are nouns/noun phrases or open class entities they are rarely shorter words.

Word position

There are two features corresponding to word position. The feature ‘First Word’ is set to 1 if the current word is the first word of the sentence; or else set to 0 and the feature ‘Last Word ‘is set to 1 if the given word is the last word of the sentence.

Frequent Word List

According to Luhn, words with very high frequency and very low frequency are not sense carrying agents, they are either rare words or closed class words.

Surrounding Words

Previous and subsequent words of a particular word are used as features. This feature is multivalued. Different window sizes were used for different domains in different experiments.

NE Information

The NE tags of the previous words are used as a dynamic feature.
**Language Dependant Feature**

*Parts of Speech Information*

POS of the current word and the surrounding words are important to recognize NEs. POS also indicates whether the word is a ‘stand alone word’ or part of a phrase.

*Phrase Chunk Information*

Certain NEs are noun phrases which appear either directly or as a part of other phrases such as postpositional phrases.

**8.4.4 NE Disambiguator**

Gazetteer lists are created for the names of diseases, persons and locations. In case NE Classifier cannot decide the final NE tag of the token, decision is made by this module.

**8.5 Performance Evaluation**

Its performance is evaluated using standardized techniques precision, recall and F-score where precision is defined as a ratio of number of correct NE tags in the output to the number of NE tags in the output and recall is the ratio of number of correct NE tags in the output to the number of NE tags in the test data. \( F_{score} = \frac{2 \times recall \times precision}{recall + precision} \) [194]. Documents in the corpus were selected from the medical domain. Then 4000 sentences were randomly selected for training and 700 sentences as test set. Precision, recall, and F-score obtained for certain NE tags are shown in Table 8.2. Highest recall is 96.21% for the entity LOCATION and highest precision for the entity JOURNAL. Highest F-Score is obtained for the named entity JOURNAL.
### Table 8.2 NER Performance by Named Entity Type

<table>
<thead>
<tr>
<th>Named Entity</th>
<th>Recall</th>
<th>Precision</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERSON</td>
<td>95.32</td>
<td>94.28</td>
<td>94.78</td>
</tr>
<tr>
<td>ORGANISATION</td>
<td>89.25</td>
<td>88.46</td>
<td>88.85</td>
</tr>
<tr>
<td>LOCATION</td>
<td>96.21</td>
<td>95.30</td>
<td>95.75</td>
</tr>
<tr>
<td>DISEASE</td>
<td>88.41</td>
<td>89.53</td>
<td>88.97</td>
</tr>
<tr>
<td>SYMPTOM</td>
<td>86.38</td>
<td>84.14</td>
<td>85.25</td>
</tr>
<tr>
<td>REASON</td>
<td>82.18</td>
<td>83.60</td>
<td>82.88</td>
</tr>
<tr>
<td>AGE</td>
<td>87.64</td>
<td>89.43</td>
<td>88.52</td>
</tr>
<tr>
<td>TIME</td>
<td>90.40</td>
<td>92.52</td>
<td>91.44</td>
</tr>
<tr>
<td>DATE</td>
<td>92.67</td>
<td>91.46</td>
<td>92.05</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>91.20</td>
<td>90.16</td>
<td>90.68</td>
</tr>
<tr>
<td>VIRUS</td>
<td>95.47</td>
<td>94.50</td>
<td>94.98</td>
</tr>
<tr>
<td>JOURNAL</td>
<td>96.00</td>
<td>97.23</td>
<td>96.61</td>
</tr>
<tr>
<td>MEDICINE</td>
<td>92.87</td>
<td>94.75</td>
<td>93.80</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>74.49</td>
<td>76.28</td>
<td>75.37</td>
</tr>
<tr>
<td>PRECAUTION</td>
<td>78.34</td>
<td>75.65</td>
<td>76.97</td>
</tr>
<tr>
<td>TREATMENT</td>
<td>88.00</td>
<td>86.50</td>
<td>87.24</td>
</tr>
<tr>
<td>FOOD</td>
<td>87.04</td>
<td>92.56</td>
<td>89.72</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>87.00</td>
<td>91.03</td>
<td>88.97</td>
</tr>
<tr>
<td>ELEMENT</td>
<td>88.24</td>
<td>89.21</td>
<td>88.77</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>84.52</td>
<td>85.31</td>
<td>84.91</td>
</tr>
<tr>
<td>REST</td>
<td>87.64</td>
<td>91.46</td>
<td>89.50</td>
</tr>
<tr>
<td>COLOR</td>
<td>79.28</td>
<td>82.71</td>
<td>80.96</td>
</tr>
<tr>
<td>TEST</td>
<td>81.65</td>
<td>82.27</td>
<td>81.46</td>
</tr>
<tr>
<td>OBJECT</td>
<td>78.43</td>
<td>81.10</td>
<td>79.74</td>
</tr>
<tr>
<td>COUNTRY</td>
<td>79.36</td>
<td>85.42</td>
<td>82.28</td>
</tr>
<tr>
<td>AGENT</td>
<td>80.14</td>
<td>81.16</td>
<td>80.65</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>86.90</td>
<td>87.90</td>
<td>87.70</td>
</tr>
</tbody>
</table>

### Table 8.3 Overall Performance of the Named Entity Tagger

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Actual</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pos</td>
<td>Neg</td>
</tr>
<tr>
<td>Pos</td>
<td>743</td>
<td>102</td>
</tr>
<tr>
<td>Neg</td>
<td>113</td>
<td>53</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>856</td>
<td>155</td>
</tr>
</tbody>
</table>
The contingency table given above shows the overall performance of Named Entity Tagger.

\[
\begin{align*}
TP &= 743 \quad FN= 113 \quad FP = 102 \quad TN= 166 \\
\text{Average Precision} &= \frac{743}{743+102} = 87.9 \% \\
\text{Average Recall} &= \frac{743}{743+113} = 86.9 \%
\end{align*}
\]

Table 8.4 gives a few examples of Named Entities identified and corresponding NE tags assigned. Underlined word/phrase/sentence is the Named Entity identified.

8.6 Chapter Summary

NER plays an important role in the design of an accurate answer retrieval and efficient MT system. The objective of NER is to categorize all NE words in a document into predefined classes like PERSON, LOCATION, DISEASE etc.

Twenty six Named Entity tags were identified for a closed domain QA System in the medical domain. An SVM based NER system is designed and its details are discussed in this chapter. Results show that Multiclass SVM is a promising method for Malayalam NE classification and recognition. Approach presented here requires linguistic preprocessing of the document text such as morphological analysis, POS Tagging and phrase chunking.
## Table 8.4 NE Tag Examples

<table>
<thead>
<tr>
<th>NE TAG</th>
<th>Examples and the Named Entity</th>
</tr>
</thead>
</table>
| TREATMENT   | പുരിന്നാണ്‌ വെളുത്ത കാരണം കാണിക ശീലത്തിൽ അപൂർവ്വമായ വിഭാഗങ്ങൾ അര പി വെളിയിൽ ഒന്നർതന്നെയാണ്‌ പ്രത്യേകിച്ചു അവകാശം പിൻവലിച്ചു കൊല്ലിയായിരിക്കുന്നു
   (pURNNa viSramavum uppum kozhuppum kuRanja glukkOs kUTuthalLLa aahaaravum aaN~ ii rOgam kaThinamallaaththavaRakkualLLar mukhyachikilsaa nirddESangaL) |
| SYMPTOM     | ആര്‍ കാരണങ്ങളൊക്കെ അവിശ്വാസിക്കുന്ന വിഭാഗങ്ങളാണ്‌ സൂക്ഷിച്ചു.
   (chhaRddi aaakkaanam varvRerichchil ennI rOgalakSHAaNangaL unTaakaam) |
| MEDICINE    | നേരും മാത്രമാണ്‌ മുക്ത്‌ മേള (intaRpheRON enna marunn) |
| REST        | മുമ്പ് പിന്നിലേക്ക് നീക്കുന്ന പോലെ കാട്ടി എത്താണ്‌ പ്രാണത്തിലുള്ളു. |
| (manjappiththam thIRchchappeTuththiyaal mUnn naal~ aazhcha viSRamameTukkaNam) |
| AGE         | (chhaRddi aaakkaanam varvRerichchil ennI rOgalakSHAaNangaL unTaakaam) |
| ELEMENT     | കാരണത്തിൽ കാരണം മാത്രമാണ്‌ അവിശ്വാസിക്കുന്ന വിഭാഗം അപൂർവ്വമായ കാരണങ്ങൾ ഉണ്ടായിരിക്കുന്നു
   (rakthaththil belRUbin athikarikkunnathu mUlamaaN~ SarIraththin~ manjaniRam unTaakkunnath~) |
| DEFN        | നേരും മാത്രമാണ്‌ മുക്ത്‌ മേള (intaRpheRON enna marunn) |
| TEST        | അകത്തെന്നെ പോലെ പോലെ പ്രത്യേകിച്ചു അവിശ്വസിച്ചായിരിക്കുന്നു
   (echch~ bi es~ e Ti) |
In this chapter various performance evaluation techniques that can be used for QA Systems are discussed. The experimental details and various methods of MaQAS evaluation are also described.

In Information Retrieval systems, user query request is inherently vague and the retrieved documents are not exact answers and hence they have to be ranked according to their relevance to the query. Such relevance ranking introduces a component which plays central role in IR that determines how precise the answer set is. This type of evaluation is referred to as retrieval performance evaluation [195]. Such an evaluation is based on a test reference collection and on an evaluation measure. The test reference collection consists of a collection of documents, a set of sample information requests, and a set of relevant documents for each sample information request. Given a retrieval strategy S, the evaluation measure quantifies the similarity between the set of documents retrieved by S and the set of relevant documents.

When considering the performance evaluation of IR systems, we should first consider the retrieval task that is to be evaluated. For instance, the retrieval task could simply consist of a query processed in batch mode or of a whole
interactive session. Batch and interactive query tasks are quite distinct processes and thus their evaluations are also distinct. Besides the nature of the query request one has to consider the environment where the evaluation will take place and the type of interface used. Evaluation of experiment performed in a laboratory might be quite distinct from evaluation of experiments carried out in real life situation.

9.1 General Methods of Evaluation

From previous chapters it is evident that the QA Systems are not just made up of a single component but a series of components working together to achieve the final goal. Though the final goal of a QA System is to obtain a correct answer to the question asked, each of the individual components within the system has their own goals which eventually lead to the final goal. Thus the performances of the individual components are likely to influence the entire QA task.

**Black-box Evaluation Approach** – Here the performance of the system is considered as whole without caring much about the performance of individual components [196]. Thus the final answer from the system is compared with the question asked to evaluate the QA System. A correct answer is what is expected from a QA System but there could be more than one correct answer for a given question. Thus the QA System needs to find out all possible final answers.

**Glass-box Evaluation Approach** – Here each of the individual components of QA Systems are evaluated with appropriate methods particular to that component [196]. The goal is to have optimal performances for each individual component which will ultimately lead to a better overall performance of the QA System.

**Interactive Setting** - Evaluation Scenario (setting) determines the appropriate evaluation metrics. An example task for interactive setting is finding out the
answers for a questionnaire. The performance of the system is measured in terms of the average number of questions attempted, the average number of questions answered correctly, and the time taken to fill out the questionnaire.

**Non-Interactive Setting** - In a batch (or non-interactive) setting, the ability of the system is measured by finding the relevant documents corresponding to a query and ranking them accordingly [197].

### 9.2 Evaluation Metrics

To accurately measure the performance of a QA System we need matrices that can provide good indication on how the system would perform in real world scenario. In QA it is important to retrieve documents that contain the answer or part of the answer that will satisfy the user question. There are a number of evaluation measures that can be used to compare the performance of the various retrieval techniques. Each measure highlights a different aspect and use of several measures to describe the performance of a system is more revealing than using a single measure. On the other hand, when comparing systems, it is often more convenient to use a single measure and the choice depends on the purpose of the retrieval system and the context in which it is used.

**Precision and Recall**

These are the most commonly used indicators to measure IR retrieval quality. Precision of the system is the fraction of retrieved material that is actually relevant. Recall is defined as the fraction of relevant material actually retrieved in answer to a search request [198]. Precision can be seen as a measure of exactness or quality whereas recall is a measure of completeness or quantity. High recall means that a system returned most of the relevant results. High precision means
that the system returned more relevant results than irrelevant. Precision and recall
are depicted in fig 9.1.

Consider a sample information request I. Given that \( |R_r| \) be the number of relevant documents retrieved, \( |R_j| \) be the number of irrelevant documents retrieved and \( |N_r| \) be the number of relevant documents not retrieved, then the precision and recall are given by the following equations.

\[
\text{Precision} = \frac{|R_r|}{|R_r| + |R_j|} = \frac{A \cap B}{B}
\]
\[
\text{Recall} = \frac{|R_r|}{|R_r| + |N_r|} = \frac{A \cap B}{A}
\]

Fig 9.1 Set Diagram Showing Elements of Precision and Recall

Precision and recall can also be defined by a contingency table. This table is normally used in classification tasks. A typical contingency table used in IR systems is shown below.

Table 9.1 Contingency Table

<table>
<thead>
<tr>
<th>Total Documents(D)</th>
<th>Relevant (A)</th>
<th>Non-Relevant (A')</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieved (Predicted +ve) (B)</td>
<td>TP = A \cap B \quad \text{(true +ves)}</td>
<td>FP = A' \cap B \quad \text{(false +ves)}</td>
</tr>
<tr>
<td>Not Retrieved (Predicted -ve) (B')</td>
<td>FN = A \cap B' \quad \text{(false -ves)}</td>
<td>TN = A' \cap B' \quad \text{(true -ves)}</td>
</tr>
</tbody>
</table>

TP = Number of true positives
FP = Number of false positives
TN = Number of true negatives
FN = Number of false negatives
True positives are documents deemed relevant by both the human expert and the information retrieval system. False positives are returned by the IR system, but were reckoned irrelevant to the query by the human expert. False negatives are documents relevant to the query which are not found by the system. True negatives are not returned by the system and are considered irrelevant by the human expert.

\[
\text{Accuracy} = \frac{TP+TN}{(TP+FP+TN+FN)}
\]

\[
\text{Precision} = \frac{TP}{(TP+FP)}
\]

\[
\text{Recall} = \frac{TP}{(TP+FN)}
\]

**F-Measure**

F-Measure or F-Score can be defined as harmonic mean of precision and recall. It is a measure of system’s accuracy. It considers both the precision \(P\) and recall \(R\) to compute the score.

\[
\text{F-Measure} \ F = \frac{2\times P\times R}{P + R}
\]

F-Measure reaches its best value at 1 and worst score at 0. It is 0 when no relevant documents have been retrieved and is 1 when all ranked documents are relevant. The harmonic mean \(F\) assumes a high value only when both recall and precision are high [69].

**9.3 MaQAS –Implementation**

MaQAS accepts questions in simple sentences, analyses them, and returns answers in a single word, phrase, or sentence. The platform used for the implementation of MaQAS is given below.
Platform Used

An Intel Core2 Duo CPU 2.00 GHz with 2GB RAM and 150GB Hard disk was used. Operating System was Windows XP.

Database: My SQL Version 4.1.14

Language: J2SDK 1.4.2

Testing Environment

The corpus was prepared by collecting documents from medical books, journals, and health magazines like “Arogya” (health). Test questions were gathered from various users around us. This system was tested with hundreds of different types of questions. A sample document and a set of sample questions are given below.

Sample Document

Platform Used

Database: My SQL Version 4.1.14

Language: J2SDK 1.4.2

Testing Environment

The corpus was prepared by collecting documents from medical books, journals, and health magazines like “Arogya” (health). Test questions were gathered from various users around us. This system was tested with hundreds of different types of questions. A sample document and a set of sample questions are given below.

Sample Document
Transliterated version of above document is given below.

*manushyaSrareerathhil udaraththinte va lathubhaagathth~ vaariyellukaLkk~thottu thaazheyaaN~ karaL sthithicheyyuth~. pradhaanamaayum heppataitis~e vaiRasukaLaaN~ paTarnupiTikkunna manjinjapiththaththinu kaaraNam. pradhaanamaayum malinajalaththilooTe pakarunna heppataitis~e avikasitha raajyangngaLilaaN~ kooTuthalaayi kaaNappeTunnath~ Ethu praayaththiluLLavareyum heppataitis~e manjinjappiththam baadhikkaamengkilum kuTTikaLilaaN~ kooTuthal rOgasaddhyatha pakarchchavyaadhiyaaya manjinjapiththam varshaththilellaakkaalavum paTar nnupiTikkaamengkilum mazhakkaalaththaaNu rOgam kooTuthal vyaapakamaakunnath~. malinajalam upayOgichch~ kazhukiya pazhavarggangngaL, pachchakkaRikaL iva upayOgikkunnathilooTeyum rOgam pakaraam.*

**Sample Questions**

a. 农业大学 (enthaaN~ manjinjappiththam?) (what is Jauntice?)

b. 农业大学 农业大学 农业大学 (manjinjappiththam ennaal enthaaN~?) (what is meant by Jauntice?)

c. 农业大学 农业大学 农业大学 (engngane chikithsikkaNam?) (how to treat?)

d. 农业大学 农业大学 (enthaaN~ marunn~?) (what is the medicine?)

e. 农业大学 农业大学 农业大学 农业大学 (enthu chikithsayaaN~ manjinjappiththaththinuLLath~?) (what is the treatment for Jauntice?)

An example of a document available in the corpus and types of questions used for testing the system are listed in Appendix G and H respectively.
MaQAS performance was evaluated employing various measures and its details are vividly portrayed below.

9.4 Performance Evaluation of MaQAS

Evaluation criteria do a major job in the completeness of performance analysis. The question set was first divided into ten different categories based on the question type and then into 26 categories based on the expected NE type. These questions were then fed into the QAS one by one and the retrieved answers were analysed. MaQAS was evaluated using the measures precision, recall, and F-Measure. Table 9.2 shows the test results of MaQAS for a particular run. It gives the following information.

<table>
<thead>
<tr>
<th>Total number of questions asked</th>
<th>= 200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of answers present in the corpus</td>
<td>= 171 (147+24)</td>
</tr>
<tr>
<td>Number of questions correctly answered</td>
<td>= 147</td>
</tr>
<tr>
<td>Number of questions wrongly answered</td>
<td>= 19</td>
</tr>
<tr>
<td>Number of answers present in the corpus but not retrieved</td>
<td>= 24</td>
</tr>
<tr>
<td>Answers which were not relevant</td>
<td>= 10</td>
</tr>
</tbody>
</table>

Table 9.2 Contingency Table Showing MaQAS output

<table>
<thead>
<tr>
<th>Number of Answers</th>
<th>Relevant</th>
<th>Not-relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieved</td>
<td>147</td>
<td>19</td>
</tr>
<tr>
<td>Not retrieved</td>
<td>24</td>
<td>10</td>
</tr>
</tbody>
</table>

True Positives (TP) = 147
False Positives (FP) = 19
True Negatives (TN) = 10
False Negatives (FN) = 24
Performance Evaluation

Precision (P) = TP / (TP+FP) = 147 / (147+19) = 88.5%
Recall (R) = TP / (TP+FN) = 147 / (147+24) = 85.9%
F-Measure = 2*P*R / (P+R) = 2*88.5*85.9 / (88.5+85.9) = 87.2%

9.5 Analysis and Discussion of Results

Performance of MaQAS was evaluated by measuring its ability to retrieve all and only relevant information. MaQAS performance is strongly dependant on NE Tagging and correct processing of the queries. The system achieved an overall precision of 88.5% and 85.9% of recall. There is a tradeoff between precision (P) and recall (R). Higher the value of P lower will be the value of recall and vice versa. The F-Measure is the harmonic mean of P and R.

Referring to Table 9.2 it is clear that for 200 questions asked, 166 answers were retrieved out of which 147 answers were relevant to the query and 19 were non-relevant. Even though 24 other relevant answers existed in the corpus they were not extracted because some of the question patterns were not recognized properly or semantics was not sufficient enough to identify the required NE. In this work, retrieval scheme is purely based on named entities. Hence all the sentences with the candidate entities are retrieved. But all these entities might not be the answers to the question.

Table 9.3 Performance of MaQAS

<table>
<thead>
<tr>
<th>Questions</th>
<th>Relevant Answer</th>
<th>Retrieved Answers</th>
<th>Correct Output</th>
<th>Recall</th>
<th>Precision</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>173</td>
<td>166</td>
<td>147</td>
<td>85.9</td>
<td>88.5</td>
<td>87.2</td>
</tr>
</tbody>
</table>
The overall performance of MaQAS is given in Table 9.3. Precision of 88.5% shows that the answers retrieved are correct answers and only very few non-relevant answers are retrieved. Percentage of recall is less than precision and is 85.9%. This can certainly be improved by providing a WordNet facility. While raising the questions no restriction is kept to avoid any bias that may affect the system performance. Question patterns were prepared just considering the question keyword. No considerations were given to other words which are present in the question as different users might use different combinations of words or phrases. It is not possible to consider all word combinations that can occur in a query and also is not an efficient practice. If a WordNet is present, it can be used during the identification of secondary keywords of question analysis phase which will certainly increase the recall value.

Table 9.4 describes the performance of MaQAS based on question type. Factoid and certain non-factoid questions were only considered in this work. Yes/No questions are not considered in the design of MaQAS and hence still it remains as a research topic. Poor performances of other question types are mainly due to the disability of correct identifications of NE from the question analysis phase. Handling of ‘HOW’ and ‘WHY’ type questions are the most difficult because they mostly require answers spreading over more than one sentence or paragraph. These questions sometimes require deep semantic processing of the sentences and identification of more keywords to detect the presence of explanations, intentions, justifications etc.
Table 9.4 Performance According to Question Type

<table>
<thead>
<tr>
<th>Question type</th>
<th>No. of questions</th>
<th>Retrieved Answers</th>
<th>Correct Answers</th>
<th>Wrong Answers</th>
<th>Relevant Not-Retrieved</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>75</td>
<td>100</td>
<td>85.7</td>
</tr>
<tr>
<td>Where</td>
<td>11</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>83.3</td>
<td>71.4</td>
<td>76.9</td>
</tr>
<tr>
<td>What</td>
<td>48</td>
<td>44</td>
<td>42</td>
<td>2</td>
<td>2</td>
<td>95.5</td>
<td>95.5</td>
<td>95.5</td>
</tr>
<tr>
<td>Why</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>75.0</td>
<td>60.0</td>
<td>66.7</td>
</tr>
<tr>
<td>Which</td>
<td>53</td>
<td>49</td>
<td>46</td>
<td>3</td>
<td>4</td>
<td>93.9</td>
<td>92.0</td>
<td>92.0</td>
</tr>
<tr>
<td>When</td>
<td>14</td>
<td>12</td>
<td>10</td>
<td>2</td>
<td>2</td>
<td>83.3</td>
<td>83.3</td>
<td>83.3</td>
</tr>
<tr>
<td>How</td>
<td>26</td>
<td>23</td>
<td>18</td>
<td>5</td>
<td>5</td>
<td>78.3</td>
<td>78.3</td>
<td>78.3</td>
</tr>
<tr>
<td>How much</td>
<td>9</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>100</td>
<td>80.0</td>
<td>88.9</td>
</tr>
<tr>
<td>Others</td>
<td>18</td>
<td>15</td>
<td>13</td>
<td>2</td>
<td>4</td>
<td>86.7</td>
<td>76.5</td>
<td>81.3</td>
</tr>
<tr>
<td>Total</td>
<td>200</td>
<td>166</td>
<td>147</td>
<td>19</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Performance based on expected answer type is listed in Table 9.5. No answer is retrieved for ‘ORGANISATION’ type questions. This is because no organization NE is present in the index. Though this entity is not present in the documents presently tagged, it might be required in future. Named Entity ‘ORGANISATION’ is considered in the tag list since it is noticed during the question analysis phase that many user questions had this. At the same time 100% accuracy is obtained for ‘REST’ type questions. The failures in NE Tagging, question analysis phase and unavailability of appropriate answers in the corpus are the reasons behind the reduced performance rate in other question types.
Table 9.5 Performance According to Answer Type

<table>
<thead>
<tr>
<th>Answer Type</th>
<th>No. of questions</th>
<th>Retrieved Answers</th>
<th>Correct Answer</th>
<th>Wrong Answer</th>
<th>Relevant Not-Retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISEASE</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MEDICINE</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TREATMENT</td>
<td>12</td>
<td>10</td>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORGANISATION</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PERSON</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>LOCATION</td>
<td>11</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>VIRUS</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PRECAUTION</td>
<td>14</td>
<td>12</td>
<td>11</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>FOOD</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>REASON</td>
<td>13</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>ELEMENT</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AGE</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>10</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>REST</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SYMPTOM</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TIME</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>COLOR</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TEST</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OBJECT</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>JOURNAL</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>COUNTRY</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>AGENT</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>DATE</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 9.6 Performance of Different Runs

<table>
<thead>
<tr>
<th>Run ID</th>
<th>Total Questions</th>
<th>Retrieved Answers</th>
<th>Correct Answers</th>
<th>Wrong Answers</th>
<th>Relevant</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>112</td>
<td>98</td>
<td>90</td>
<td>8</td>
<td>108</td>
<td>91.8</td>
<td>83.3</td>
<td>80.3</td>
</tr>
<tr>
<td>Run 2</td>
<td>97</td>
<td>92</td>
<td>82</td>
<td>10</td>
<td>95</td>
<td>89.1</td>
<td>86.3</td>
<td>84.5</td>
</tr>
<tr>
<td>Run 3</td>
<td>63</td>
<td>57</td>
<td>45</td>
<td>12</td>
<td>60</td>
<td>78.9</td>
<td>75.0</td>
<td>71.4</td>
</tr>
<tr>
<td>Run 4</td>
<td>150</td>
<td>141</td>
<td>121</td>
<td>20</td>
<td>148</td>
<td>85.1</td>
<td>81.7</td>
<td>80.7</td>
</tr>
<tr>
<td>Run 5</td>
<td>170</td>
<td>122</td>
<td>104</td>
<td>18</td>
<td>134</td>
<td>85.2</td>
<td>77.8</td>
<td>61.2</td>
</tr>
<tr>
<td>Run 6</td>
<td>45</td>
<td>39</td>
<td>38</td>
<td>1</td>
<td>40</td>
<td>97.4</td>
<td>95.0</td>
<td>84.4</td>
</tr>
<tr>
<td>Run 7</td>
<td>200</td>
<td>166</td>
<td>147</td>
<td>19</td>
<td>171</td>
<td>88.5</td>
<td>85.9</td>
<td>78.5</td>
</tr>
</tbody>
</table>

Questions were collected from seven users of different walks of life and the respective results are shown in Table 9.6.

9.6 Chapter Summary

Performance evaluation of MaQAS is discussed in this chapter. General measures used to evaluate the performance of QA Systems are described. Then the analysis and discussion of experiments conducted to test the system MaQAS is presented. The experimental results show that the Named Entity Based approach is a successful method for QAS. In addition to keyword matching, use of the deep analysis of the text helps to improve the performance of the system. Double level index not only reduced the run time overhead but also increased the speed of answer retrieval.
Main contributions of this research work are highlighted in this chapter. Some open issues in the design of Malayalam QAS are discussed and the ways in which they could be addressed in future work are proposed.

This research work is about the development of a QAS for Malayalam language. MaQAS extracts answers suitable to the type of question using Named Entity techniques. Question Answering Systems available in other languages are mostly document or passage retrieval systems or sometimes extract answers just by keyword or pattern matching techniques. In this system, question is analysed and type of the expected answer is identified. Then double level indexing scheme is used to extract possible candidate sentences from the preprocessed documents. These sentences are ranked and top scored sentence is used as answer candidate from which the Named Entity that matches the question type is separated.

In this approach, various Natural Language Processing tools are used for the analysis of both the question and the documents. The most important part of this system is the document preprocessor which performs morphological analysis, syntactic analysis and semantic analysis. These stages tag each word of the document with POS, phrase, and NE tags. Next important stage is a
question classifier which uses pattern matching and rules that have been formulated by observing a bank of sample questions. This system classifies the input queries by the question type and the type of answer. From the question, main keywords and secondary keywords or phrases are extracted to retrieve the answer from the preprocessed documents.

This system performs deep linguistic processing of the documents and prepares NE based index. During runtime, no overhead is present for document processing. MaQAS when tested with Factoid, Definition, List, and Descriptive questions gave a recall of 85.9% and precision of 88.5%.

10.1. Contributions

A major contribution of this work is the design and development of an innovative QA System for Malayalam language which focus on efficiency and robustness. The most relevant contributions of MaQAS are highlighted below.

- Question Answering Systems require understanding of both questions and documents. This needs morphological, syntactical and semantic analysis of natural language sentences. Since no NLP tools were available for Malayalam language, tools like Compound Word Splitter, POS Tagger, Phrase Chunker and Named Entity Tagger were developed.

- Special POS and Phrase Tagsets are designed.

- Most of the QA Systems perform document retrieval and sequential search of the query over the entire document collection. This system adopted a double level indexing scheme which facilitates the direct retrieval of candidate sentences.
Traditional systems consider only proper nouns as NEs while MaQAS define every role carrying agent as a NE.

No need to restructure the question entered by the user while majority of the systems translate the input questions to a Boolean query or a SQL statement.

10.2. Future Work

QAS developed in this work is restricted to handle questions from the domain of lifestyle and infectious diseases. This can be extended to any area (open domain QA System) making it beneficial to the society at large.

This research work can be extended to include Anaphora Resolution techniques since it presents a big challenge to QAS. Anaphora is a natural language expression used, to refer an entity that has been previously introduced in the document or in the question. The pronouns and definite noun phrases found during the pattern matching process of question analysis phase have to be linked to the correct antecedent in order to identify the exact question target. This will be helpful in extracting complete and meaningful answers. Identification of the “references” is essential to understand and process the documents in the corpus. Accuracy of NE identification and index preparation is solely dependant on proper understanding of the documents. Hence Reference Resolution has an important role in the index preparation stage.

Another direction in which this work can be extended is to include the facility of Word Sense Disambiguation (WSD). Word and their meanings provide appropriate bits necessary to construct the NE-based document representation and for key word extraction. A word can have multiple senses and hence, it is necessary to discover which sense of the word is used in the
documents and in the query. Various relations exist between words and their senses. Some of them are Homonymy, Polysemy, Synonymy, and Hyponymy. Homonymy and Polysemy issues reduce the precision of a QAS by leading a system to return answers irrelevant to the user’s information need. Synonymy and Hyponymy reduce the recall by causing the retrieval system without identifying the relevant answers. WordNet and WSD techniques can solve such word sense issues.
REFERENCES


References


References


References


[29] David N. Chin, “Knowledge Structures in UC, the UNIX Consultant”, National Science Foundation


[34] E.M. Voorhees, Buckland, Proceedings of 13th Text Retrieval Conference, TREC 2004


References


References


[49] Carlos Amaral, Helena Figueira, Andre Martins Afonso Mendes, Pedro Mendes, Claudia Pinto, “Priberam’s Question Answering System for Portuguese”, 1999


References


[67] Dr.C.K.Chandrasekharan Nair, “Adisthana Vyakaranam”, Kerala Bhasha Institute, Thiruvananthapuram, 1997


[70] Marcin Skowron, Kenji Araki, “Effectiveness of Combined Features for Machine Learning Based Question Classification”, Journal of Natural Language Processing 12(6), pp 63-83, 2005


[78] Anne schiller, “German compound word analysis with wfsc”, Proceedings of the Fifth International Workshop of Finite State Methods in Natural Language Processing, Helsinki, 2005


[94] Parameshwari K, “An Implementation of APERTIUM Morphological Analyzer and Generator for Tamil”, Language in India
References


References


References

References


References


References


A) Publications in International Refereed Journals


B) Papers in International Conferences


C) Papers in National Conferences

APPENDIX A

STOP WORD LIST

( saadhaarana gathiyil )
( vare )
( ee )
( iLam )
( cheRiya )
( ann~ )
( athiniTe )
( thane )
( pTTenn~ )
( iththaram )
( aviTe )
( athupOle )
( ithinte )
( ethrayO )
( dhaaralLam )
( ithOTe )
( athinekkuRich~ )
( thamasiyaathe )
( alpamaaya )
( angngEyatam )
( oru )
( ini )
( pala )
( kramENa )
( prasathutha )
( thuTarnn~ )
( ennaal )
( Ethaum )
( upayOgichchu )
( poorNNamaayi )
( kazhivathum )
( thalpalamaayi )
( paripoorNNamaayO )
( pinneeT~ )
( enniva )
( theere )
( kooTTamaayi )
( muthalulLLa )
( mat~ )
( ithiniTe )
( chilappOL )
( vaLare )
( iviTe )
( oTTTERe )
( athinte )
( uNTaakunn )
( ennivayile )
( pradhaana )
( munpEthanne )
( paramaavadhi )
( sahaayikkum )
( aavaSyam )
( paridhivare )
( ithallaathe )
( maathram )
( athE )
( kooTuthalaayi )
( aaya )
( vaLare )
( orikkalum )
( nimishanggaLkk~ )
( palappOzhum )
( vEnTavidham )
( athhavaa )
( paridhivare )
( bhaagikamaayO )
<table>
<thead>
<tr>
<th>Appendixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>പരസ്പരം (parasparam)</td>
</tr>
<tr>
<td>നിശ്ശേഷം (niSSEsham)</td>
</tr>
<tr>
<td>മൊത്തമായ്ക്ക് (moththamaayO)</td>
</tr>
<tr>
<td>വ്യാക്തമായി (vyakthamaayi)</td>
</tr>
<tr>
<td>സ്രദ്ധകോത്താണ്ഡ് (SradhakoTuthth~)</td>
</tr>
<tr>
<td>ഗോതമായ (ghaTanaaparamaayi)</td>
</tr>
</tbody>
</table>
## APPENDIX B

**Malayalam ISCII-Unicode Mapping Table**

<table>
<thead>
<tr>
<th>Hindi</th>
<th>English</th>
<th>ISCII</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>अ</td>
<td>a</td>
<td>0021</td>
<td>0001</td>
</tr>
<tr>
<td>आ</td>
<td>a</td>
<td>0022</td>
<td>0002</td>
</tr>
<tr>
<td>इ</td>
<td>i</td>
<td>0023</td>
<td>0003</td>
</tr>
<tr>
<td>ई</td>
<td>e</td>
<td>0024</td>
<td>0005</td>
</tr>
<tr>
<td>उ</td>
<td>u</td>
<td>0025</td>
<td>0007</td>
</tr>
<tr>
<td>ऊ</td>
<td>u</td>
<td>0026</td>
<td>0008</td>
</tr>
<tr>
<td>ऋ</td>
<td>r</td>
<td>0027</td>
<td>0009</td>
</tr>
<tr>
<td>ऌ</td>
<td>r</td>
<td>0028</td>
<td>000A</td>
</tr>
<tr>
<td>ए</td>
<td>e</td>
<td>0029</td>
<td>000D</td>
</tr>
<tr>
<td>ऐ</td>
<td>e</td>
<td>002A</td>
<td>000E</td>
</tr>
<tr>
<td>ओ</td>
<td>o</td>
<td>002B</td>
<td>000F</td>
</tr>
<tr>
<td>औ</td>
<td>o</td>
<td>002C</td>
<td>0010</td>
</tr>
</tbody>
</table>

... (Table continues with more entries)
APPENDIX C

A View of Lexicon used in MaQAS

<table>
<thead>
<tr>
<th>Type field</th>
<th>Mean 1</th>
<th>Mean 2</th>
<th>Mean 3</th>
<th>Mean 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun/verb/Adj/Adv/PSP/Suffix</td>
<td>SL/PL/T1/T2/T3</td>
<td>M/F/N/C</td>
<td>C1-C6 / R1-R2</td>
<td>0 or 1(Human or non-human)</td>
</tr>
<tr>
<td>SL</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>PL</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>T1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>T2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>T3</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>M</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>F</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>C1-C6</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>R1-R2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Type field: - Noun/verb/Adj/Adv/PSP/Suffix
Mean 1: - SL/PL/T1/T2/T3
Mean 2: - M/F/N/C
Mean 3: - C1-C6 / R1-R2
Mean 4: - 0 or 1(Human or non-human)

SL: - Singular
PL: - Plural
T1: - Past tense
T2: - Present Tense
T3: - Future Tense
M: - Masculine Gender
F: - Feminine Gender
C: - Common Gender
N: - Neuter Gender
C1-C6: - Case suffixes
R1-R2: - Active/Passive voice
## APPENDIX D

### Performance of POS Tagger

<table>
<thead>
<tr>
<th>POS Tag</th>
<th>Precision (%)</th>
<th>Recall (%)</th>
<th>F-Score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOUN</td>
<td>87.3</td>
<td>94.6</td>
<td>90.8</td>
</tr>
<tr>
<td>PN</td>
<td>98.0</td>
<td>97.1</td>
<td>97.6</td>
</tr>
<tr>
<td>RN</td>
<td>90.2</td>
<td>95.2</td>
<td>92.6</td>
</tr>
<tr>
<td>ACC</td>
<td>95.6</td>
<td>97.0</td>
<td>96.3</td>
</tr>
<tr>
<td>DAT</td>
<td>92.3</td>
<td>96.2</td>
<td>94.2</td>
</tr>
<tr>
<td>GEN</td>
<td>91.0</td>
<td>95.5</td>
<td>93.2</td>
</tr>
<tr>
<td>LOC</td>
<td>90.5</td>
<td>94.3</td>
<td>92.4</td>
</tr>
<tr>
<td>SOC</td>
<td>89.1</td>
<td>95.0</td>
<td>92.0</td>
</tr>
<tr>
<td>INST</td>
<td>95.5</td>
<td>91.4</td>
<td>93.4</td>
</tr>
<tr>
<td>OBJ</td>
<td>93.3</td>
<td>89.7</td>
<td>91.5</td>
</tr>
<tr>
<td>RES</td>
<td>90.6</td>
<td>88.5</td>
<td>89.5</td>
</tr>
<tr>
<td>PSP1</td>
<td>94.0</td>
<td>95.8</td>
<td>94.9</td>
</tr>
<tr>
<td>PSP2</td>
<td>95.6</td>
<td>93.2</td>
<td>94.4</td>
</tr>
<tr>
<td>PSP3</td>
<td>94.1</td>
<td>94.0</td>
<td>94.1</td>
</tr>
<tr>
<td>PSP4</td>
<td>95.3</td>
<td>93.1</td>
<td>94.2</td>
</tr>
<tr>
<td>PSP5</td>
<td>96.4</td>
<td>95.4</td>
<td>95.9</td>
</tr>
<tr>
<td>PSP6</td>
<td>92.3</td>
<td>97.9</td>
<td>95.0</td>
</tr>
<tr>
<td>PSP7</td>
<td>94.5</td>
<td>92.1</td>
<td>93.3</td>
</tr>
<tr>
<td>PSP8</td>
<td>96.6</td>
<td>93.6</td>
<td>95.1</td>
</tr>
<tr>
<td>PSP9</td>
<td>95.2</td>
<td>92.3</td>
<td>93.7</td>
</tr>
<tr>
<td>PSP10</td>
<td>94.4</td>
<td>95.2</td>
<td>94.8</td>
</tr>
<tr>
<td>PSP11</td>
<td>94.0</td>
<td>91.9</td>
<td>92.9</td>
</tr>
<tr>
<td>VERB</td>
<td>86.5</td>
<td>94.2</td>
<td>90.2</td>
</tr>
<tr>
<td>AdjP</td>
<td>84.9</td>
<td>91.4</td>
<td>88.0</td>
</tr>
<tr>
<td>Adv</td>
<td>83.7</td>
<td>89.6</td>
<td>86.6</td>
</tr>
<tr>
<td>AdvP</td>
<td>92.3</td>
<td>96.7</td>
<td>94.5</td>
</tr>
<tr>
<td>AuxV</td>
<td>89.2</td>
<td>94.3</td>
<td>91.7</td>
</tr>
<tr>
<td>AdvT</td>
<td>89.7</td>
<td>95.3</td>
<td>92.4</td>
</tr>
<tr>
<td>AdvP1</td>
<td>91.4</td>
<td>97.3</td>
<td>94.3</td>
</tr>
<tr>
<td>AdvPr</td>
<td>94.1</td>
<td>92.9</td>
<td>93.5</td>
</tr>
<tr>
<td>AdvR</td>
<td>90.5</td>
<td>95.5</td>
<td>92.9</td>
</tr>
<tr>
<td>AdvSe</td>
<td>91.9</td>
<td>93.0</td>
<td>92.5</td>
</tr>
<tr>
<td>AdvCo</td>
<td>92.4</td>
<td>95.1</td>
<td>93.7</td>
</tr>
<tr>
<td>AdvS</td>
<td>93.2</td>
<td>96.2</td>
<td>94.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>AdvC</td>
<td>90.3</td>
<td>95.7</td>
<td>92.9</td>
</tr>
<tr>
<td>AdjQl</td>
<td>94.6</td>
<td>93.6</td>
<td>94.1</td>
</tr>
<tr>
<td>AdjQn</td>
<td>90.5</td>
<td>92.2</td>
<td>91.3</td>
</tr>
<tr>
<td>AdjD</td>
<td>89.6</td>
<td>90.7</td>
<td>90.2</td>
</tr>
<tr>
<td>Adjn</td>
<td>90.3</td>
<td>96.0</td>
<td>93.1</td>
</tr>
<tr>
<td>AdjI</td>
<td>91.5</td>
<td>92.0</td>
<td>91.8</td>
</tr>
<tr>
<td>AdjE</td>
<td>88.4</td>
<td>90.4</td>
<td>89.4</td>
</tr>
<tr>
<td>SourceP</td>
<td>95.6</td>
<td>93.4</td>
<td>94.5</td>
</tr>
<tr>
<td>DestP</td>
<td>92.5</td>
<td>96.7</td>
<td>94.6</td>
</tr>
<tr>
<td>LikeP</td>
<td>87.4</td>
<td>93.0</td>
<td>90.1</td>
</tr>
<tr>
<td>ListP</td>
<td>88.7</td>
<td>91.5</td>
<td>90.1</td>
</tr>
<tr>
<td>TimeP</td>
<td>90.5</td>
<td>93.0</td>
<td>91.7</td>
</tr>
<tr>
<td>ThruP</td>
<td>93.1</td>
<td>92.5</td>
<td>92.8</td>
</tr>
<tr>
<td>Sym</td>
<td>96.6</td>
<td>94.7</td>
<td>95.6</td>
</tr>
<tr>
<td>CN</td>
<td>94.3</td>
<td>96.2</td>
<td>95.2</td>
</tr>
<tr>
<td>ON</td>
<td>87.4</td>
<td>93.6</td>
<td>90.4</td>
</tr>
<tr>
<td>INT1</td>
<td>96.0</td>
<td>97.4</td>
<td>96.7</td>
</tr>
<tr>
<td>INT2</td>
<td>93.4</td>
<td>94.3</td>
<td>93.8</td>
</tr>
<tr>
<td>Average Value</td>
<td>92.0</td>
<td>94.0</td>
<td>93.0</td>
</tr>
</tbody>
</table>
# APPENDIX E

Patterns for Phrase Identification

<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Phrase</th>
<th>POS Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NP</td>
<td>NOUN/PN/RN</td>
</tr>
<tr>
<td>2</td>
<td>VP</td>
<td>VERB/AuxV</td>
</tr>
<tr>
<td>3</td>
<td>NP-Acc</td>
<td>ACC</td>
</tr>
<tr>
<td>4</td>
<td>NP-Dat</td>
<td>DAT</td>
</tr>
<tr>
<td>5</td>
<td>NP-Gen</td>
<td>GEN</td>
</tr>
<tr>
<td>7</td>
<td>NP-Soc</td>
<td>SOC</td>
</tr>
<tr>
<td>8</td>
<td>NP-Obj</td>
<td>OBJ/ACC PSP6</td>
</tr>
<tr>
<td>9</td>
<td>NP-Inst</td>
<td>INST/GEN INST/NOUN PSP1/PN PSP1/RN PSP1</td>
</tr>
<tr>
<td>10</td>
<td>NP-Res</td>
<td>RES/NOUN PSP7/PN PSP7/RN PSP7</td>
</tr>
<tr>
<td>11</td>
<td>AdjNP</td>
<td>AdjP NOUN/AdjP PN/AdjP RN</td>
</tr>
<tr>
<td>12</td>
<td>AdVP</td>
<td>AdvP / Adv AdvP</td>
</tr>
<tr>
<td>13</td>
<td>AdvpT</td>
<td>AdvT / Adv AdvT</td>
</tr>
<tr>
<td>14</td>
<td>AdvpPr</td>
<td>AdvPr / Adv AdvPr</td>
</tr>
<tr>
<td>15</td>
<td>AdvpR</td>
<td>AdvR / Adv AdvR</td>
</tr>
<tr>
<td>17</td>
<td>AdvpC</td>
<td>AdvC / Adv AdvC</td>
</tr>
<tr>
<td>18</td>
<td>AdvpRe</td>
<td>AdvRe / Adv AdvRe</td>
</tr>
<tr>
<td>19</td>
<td>AdvpS</td>
<td>AdvS / Adv AdvS</td>
</tr>
<tr>
<td>20</td>
<td>AdvpCo</td>
<td>AdvCo / Adv AdvCo</td>
</tr>
<tr>
<td>21</td>
<td>AdjpQl</td>
<td>AdjQl / Adj AdvQl</td>
</tr>
<tr>
<td>22</td>
<td>AdjpQn</td>
<td>AdjQn / Adj AdjQn</td>
</tr>
<tr>
<td>23</td>
<td>AdjpN</td>
<td>AdjN / Adj AdjN</td>
</tr>
<tr>
<td>24</td>
<td>AdjpE</td>
<td>AdjE / Adj AdjE</td>
</tr>
<tr>
<td>25</td>
<td>AdjpI</td>
<td>AdjI / Adj AdjI</td>
</tr>
<tr>
<td>26</td>
<td>AdjpD</td>
<td>AdjD / Adj AdjD</td>
</tr>
<tr>
<td>27</td>
<td>SourcePP</td>
<td>NOUN PSP2/PN PSP 2/RN PSP 2/SourceP</td>
</tr>
<tr>
<td>28</td>
<td>ListPP</td>
<td>NOUN PSP11 NOUN PSP3/NOUN PSP2 DestP</td>
</tr>
<tr>
<td>29</td>
<td>LikePP</td>
<td>ACC PSP 9</td>
</tr>
<tr>
<td>30</td>
<td>ThruPP</td>
<td>LOC PSP5</td>
</tr>
<tr>
<td>31</td>
<td>DestPP</td>
<td>NOUN PSP3/PN PSP3/GEN PSP10/DestP</td>
</tr>
<tr>
<td>32</td>
<td>TimePP</td>
<td>RN PSP8/AdjN SourceP PSP8</td>
</tr>
</tbody>
</table>
### APPENDIX F

Performance Evaluation of the AIS-based Phrase chunker

<table>
<thead>
<tr>
<th>Chunk</th>
<th>Precision (%)</th>
<th>Recall (%)</th>
<th>F-Score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>93.5</td>
<td>92.6</td>
<td>93.0</td>
</tr>
<tr>
<td>VP</td>
<td>96.7</td>
<td>97.1</td>
<td>96.9</td>
</tr>
<tr>
<td>NP-Obj</td>
<td>92.0</td>
<td>92.2</td>
<td>92.1</td>
</tr>
<tr>
<td>NP-Dat</td>
<td>96.4</td>
<td>97.0</td>
<td>96.7</td>
</tr>
<tr>
<td>NP-Acc</td>
<td>94.3</td>
<td>95.2</td>
<td>94.7</td>
</tr>
<tr>
<td>NP-Gen</td>
<td>91.0</td>
<td>90.5</td>
<td>90.7</td>
</tr>
<tr>
<td>NP-Loc</td>
<td>87.9</td>
<td>88.3</td>
<td>88.1</td>
</tr>
<tr>
<td>NP-Soc</td>
<td>95.0</td>
<td>95.0</td>
<td>95.0</td>
</tr>
<tr>
<td>NP-Inst</td>
<td>92.8</td>
<td>91.4</td>
<td>92.1</td>
</tr>
<tr>
<td>NP-Res</td>
<td>90.7</td>
<td>89.7</td>
<td>90.2</td>
</tr>
<tr>
<td>AdjNP</td>
<td>89.7</td>
<td>88.5</td>
<td>89.1</td>
</tr>
<tr>
<td>AdvP</td>
<td>93.5</td>
<td>91.8</td>
<td>92.6</td>
</tr>
<tr>
<td>AdvPr</td>
<td>86.3</td>
<td>85.2</td>
<td>85.7</td>
</tr>
<tr>
<td>AdvP</td>
<td>93.4</td>
<td>92.0</td>
<td>92.7</td>
</tr>
<tr>
<td>AdvP</td>
<td>94.6</td>
<td>93.1</td>
<td>93.8</td>
</tr>
<tr>
<td>AdvP</td>
<td>88.3</td>
<td>85.4</td>
<td>86.8</td>
</tr>
<tr>
<td>AdvS</td>
<td>88.0</td>
<td>87.9</td>
<td>88.0</td>
</tr>
<tr>
<td>AdvpT</td>
<td>87.2</td>
<td>86.1</td>
<td>86.6</td>
</tr>
<tr>
<td>AdvpRe</td>
<td>92.8</td>
<td>93.6</td>
<td>93.2</td>
</tr>
<tr>
<td>AdvpCo</td>
<td>91.3</td>
<td>92.3</td>
<td>91.7</td>
</tr>
<tr>
<td>AdjpQn</td>
<td>91.4</td>
<td>89.5</td>
<td>90.4</td>
</tr>
<tr>
<td>AdjpQn</td>
<td>89.2</td>
<td>88.9</td>
<td>89.0</td>
</tr>
<tr>
<td>AdjpN</td>
<td>95.1</td>
<td>94.2</td>
<td>94.6</td>
</tr>
<tr>
<td>AdjpE</td>
<td>92.3</td>
<td>91.4</td>
<td>91.8</td>
</tr>
<tr>
<td>AdjpD</td>
<td>87.9</td>
<td>86.6</td>
<td>87.2</td>
</tr>
<tr>
<td>AdjpD</td>
<td>87.4</td>
<td>86.0</td>
<td>86.7</td>
</tr>
<tr>
<td>ListPP</td>
<td>88.6</td>
<td>87.3</td>
<td>87.9</td>
</tr>
<tr>
<td>ThruPP</td>
<td>85.5</td>
<td>86.7</td>
<td>86.0</td>
</tr>
<tr>
<td>LikePP</td>
<td>87.8</td>
<td>88.1</td>
<td>87.9</td>
</tr>
<tr>
<td>TimePP</td>
<td>94.1</td>
<td>94.7</td>
<td>94.4</td>
</tr>
<tr>
<td>SourcePP</td>
<td>92.7</td>
<td>91.1</td>
<td>91.9</td>
</tr>
<tr>
<td>DestPP</td>
<td>93.2</td>
<td>89.8</td>
<td>91.5</td>
</tr>
<tr>
<td>Average Value</td>
<td>91.3</td>
<td>90.6</td>
<td>90.9</td>
</tr>
</tbody>
</table>
APPENDIX G

A Sample Malayalam Document
Transliterated document

kOLaRa vibriyO ennaRiyappeTunna aNukkaLaaN~ kOLaRayuNTaakkunnath~. cheRukuTalineyaanN~ ee aNookkaL baadhikkunnath~. ee aNukkaL athiSakthamaaya vishamayam uNTaakkunnu.cheRukuTalile kOSangngaLe aakramikkunnathinte phalamaayi Sar eeraththinaaVaSyamaaya sODiyaththinte aagiraNam thaTassappeTukayum jalavum kLOraiD~ am_Savum nashTappeTukayum cheyyunnu.ee rOgathamthinte aNukkaL Sar eeraththileththi maNikkkuRuL kkuLLil rOgam prathyakshappeTunnu.

rOgiyuTe visar_iyavasthukkaLiL ninnu praaNiL vazhi bhakshaNaththilum veLLaththilumeththunna aNukkaLaaN~ rOgamuNTaakkunnath~.maleemasamaaya chutupaattukaLiL theeNnigaappaa kyunna kuTum_bangngaLile kuTTikaLilum muthir_nnavarilum ee asukham vaLare peTTenn~ paTar nnu piTkkaaRuNT~.

vaLare peTTennuNTaakkunna vayaRiLakkavum chhar_ddiyumaaN~ ithinte lakshaNangngaL. vayaRuvEdana, moothrathaTassam, masihukaLile koLu ththippiTikkal ennivayumuNTaakum.thuTakkaththil malaththin~ manjnijaniRam kaaNunnu. chhar_ddiyuTe kooTeththanne vayaRiLakkavum kaaNunnu. peennIT~ malavum chhar_ddiyum arikkaaTipOleyaayiththeerunnu.

chhar_ddiyuTe kooTe bail kaaNukayilla. chhar_ddichchum athisarichchum kramENa Sar eeraththile jalaam_Sam nashTappeTunnathu koNT~ rOgi apakaTunilayil eththeaRaNT~.theNuthth~ varaNTa tholi, kuzhinjniu thaaNa kaNNukaL, uNangngivaraNTa naakk~, chuNTukaL, pani enniva apakaTakaramaya lakshaNangngaLaaN~. rakthamar_ddam kuRannj~ paLS~ kiTTaathirikkuka, kithapp~ ennee lakshaNangngaLum apakaTakarangngaLaaN~.

kOLaRayethuTarnn~ kiDniyuTe pravar_ththanam nilachch~ yuReemiya, rakthacham_kramaNam nilaykkuka, graan green, nyumONiya vayar, gaal_bLaaDar, kaNNukaL, parOTTiDi~ granthhi ennivayile aNubaadha, Sakthamaaya pani enniva uNTaakaam.

o aar es~ laayaniyum karikkin veLLam, panjchasaa arayum uppu mChEr_ththa veLLam ennivayellaum upayOgikkaNaM. khararoopaththiluLLa bhakshaNaM ozhivaakki laayankkaL thanne kathikkunnathaaN~ aadyadivasangngaLil uchitham. maleemasamaaya chutupaattukaLiL_ninnu rOgiye maatippaar_ppikkaNam. poor_NaviSramam aavaSyamaaN~
APPENDIX H

LIST OF SAMPLE QUESTIONS

(маньунььаиттам энгганиетаAN~ уNTакуTҮ~ ?)
(еппOзhaаN~ маньунььаиттам пакруTҮ~ ?)
(энтааN~ маньунььаиттамthaneиттине каараNаNаNаL ?)
(эну marуunnаN~ маньунььаиттамthaneиттин~ upayOгиккуTҮ~ ?)
(энтоккеmааN~ prатhividхиL ?)
(чикитхсareeithikaL Ethоккe ?)
(эну praayаккааеN~ маньунььаиттам baadhиккуTҮ~ ?)
(ETHу vaiRасaaN~ маньунььаиттам уNTаккуTҮ~ ?)
(ETHу divасаm koNT~ маньунььаиттham бhEdамаakуm ?)
(ингниетеN~ эе rоgam пакруTҮ~ ?)
(эну TestаN~ маньунььаиттham thirичччаRиNан upayOгиккуTҮ~ ?)
(энтхааaаm kazhikкааm ?)
Appendices

- കുടിയു മരകൊടുകൊള്ളാം പിന്നീട് ?
- enthu mungkaruthalaaN~ vENTath~ ?
- മനോഹരമായ അനുഭവസഭ ?
- enthaaN~ lakshaNangngal ?
- കുടി പ്രയോട്ടാവാട് പ്രബുജം ?
- Ethu samayaththaaN~ asukham pakaruth~ ?
- മന്ദട്ടിനീ മനോഹരമായ മണയം ?
- manjinjappithham ennaal enthaaN~ ?
- അഥവാ മന്ദട്ടിനീകളും ബാലരുമായ കാര്യമില്ല ?
- aaraaN~ manjinjappiththaththinte marunn~ kaNTuppiTichchath~ ?
- പ്രതിരോധകോട്ടിന്റെയും നൂറ്റാണ്ടുവരെ ?
- prathirOdhakkuththiveypuNTO ?
- സെന്റ്റ്രിക്കടകട് ദൃശ്യൂം തികു സെങ്കും ?
- engnigneyaaN~ ee rOgam pakaruth~ ?
- കുടി അത്യംസന്തോഷവും ദൃശ്യൂം തികു സെങ്കും ?
- ethu avayavaththeyaaN~ ee rOgam baadhikkuth~ ?
- കുടി സെന്റ്റ്രിക്കടകട് ദൃശ്യൂം തികു സെങ്കും ?
- ethu kampaniyaaN~ ithin~ marunn~ uNTaakkuth~ ?
- കുടി അതിജീവിക്കുന്ന ദൃശ്യൂം തികു സെങ്കും ?
- ethu jeeviyaaN~ ee rOgam pakar_ththuth~ ?
- അഥവാ മന്ദട്ടിനീ ദൃശ്യൂം തികു സെങ്കും ?
- aaraaN~ manjinjappithham kaNTuppiTichchath~ ?
- കുടിയും മന്ദട്ടിനീകളും കൊള്ളാം ?
- enthaaN~ manjinjani Raththinu kaaraNam ?
- മന്ദട്ടിനീകളും സെന്റ്രിക്കടകട് വെള്ള ?
- manjinjappiththaththinte ananthaphalangngal Eva ?
- അഥവാ മന്ദട്ടിനീ സെന്റ്രിക്കടകട് വെള്ള ?
- engnigneyaaN~ manjinjappithham thirichchaRiyuka ?
Appendices


datum jomakalocere vino dreeco kathavimal?  
Ethu raajyakkaraaN~ ee rogam kaNTupiTichath~?  

kutubopasna? kugartipamn~ kuthapakasna?  
eppOzhaaN~ kuthhivyeyp~ eTukkENTath~?  

kadaloru dreeco ketukara?  
karaLinte upayOgam enthaaN~?  

kutubopasna? kadal kathavimal?  
eviTeyaaN~ karaL sthithicheyyuth~?  

akshi kathavimal? katal pakkamath?  
Ethu vazhiyaaN~ ithu pakaruth~?  

kutubopasna? kathavimal kojypukam?  
aahaaraththil enthokke SraddhikkaNam?  

akshi kathavimal? kathavimal kathavimal?  
Ethu kampaniyyaaN~ manjinjappithaththinte marunn~ uNTaakkuth~ kathavimal kojypukam?  
manjinjappitham thirichchaRiyaanuLLa Test~ EthaaN~  

akshi kathavimal? kathavimal pakkamath?  
Ethu samayaththaaN~ manjinjappithham pakaruth~
Appendixes

Appendix I

Screen shots showing Output of MaQAS

[Images of screen shots showing the output of MaQAS]