

**DESIGN AND DEVELOPMENT OF AN
ADAPTABLE FRAME-BASED SYSTEM
FOR DRAVIDIAN LANGUAGE
PROCESSING**

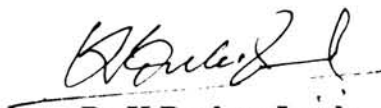
A THESIS SUBMITTED BY
SUMAM MARY IDICULA
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
UNDER THE FACULTY OF TECHNOLOGY

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE
KOCHI – 682 022

1999

C E R T I F I C A T E

This is to certify that the thesis entitled "**DESIGN AND DEVELOPMENT OF AN ADAPTABLE FRAME-BASED SYSTEM FOR DRAVIDIAN LANGUAGE PROCESSING**" is a report of the original work carried out by **Ms. SUMAM MARY IDICULA** under my supervision in the Department of Computer Science, Cochin University of Science and Technology. The results enclosed in this thesis or part of it have not been presented for any other degree.



Dr.K.Poullose Jacob
(Supervising Teacher)
Professor

Cochin – 682 022
5th April, 1999.

Department of Computer Science
Cochin University of Science and Technology

Contents

Acknowledgement		i
Abstract		ii
Chapter 1	Introduction	
1.1	Background	1
1.2	Motivation and Scope	1
1.3	Outline of the Work	3
Chapter 2	Natural Language Processing - A Review	
2.1	Introduction	7
2.2	Natural Language Analysis Techniques	10
2.2.1	Pattern Matching	10
2.2.2	Keyword Analysis	11
2.2.3	Syntactically Driven Parsing	12
2.2.3.1	Context-free Grammars	13
2.2.3.2	Transformational Grammars	14
2.2.3.3	Augmented Transition Network	15
2.2.3.4	Semantic Grammar	18
2.2.3.5	Lexical Functional Grammar	20
2.2.3.6	Government & Binding	21
2.2.4	Case Grammar	21
2.2.5	Conceptual Dependency	23
2.2.6	Expectation-Driven Parsing	24
2.2.7	Word-Expert Parsing	28
2.2.8	Integrated Partial Parser	29
2.2.9	Connectionism	30

2.3	The Indian Context	35
Chapter 3	System Architecture	
3.1	Introduction	39
3.2	Dravidian Languages	40
	3.2.1 The Common Characteristics of Dravidian Languages	40
3.3	Design Methodology	43
3.4	Ambiguity Resolution	49
3.5	System Architecture	50
	3.5.1 Morphological Analyzer	50
	3.5.2 Local Word Grouper	52
	3.5.3 Parser	53
3.6	Case study – 1 (Machine Translation)	74
3.7	Case study – 2 (A NLI for RDBMS)	79
	3.7.1 Meaning Extraction	83
	3.7.2 SQL Generator	85
	3.7.3 Ellipses & Anaphoric References	87
	3.7.3.1 Ellipses	87
	3.7.3.1.1 Surface level Ellipses	87
	3.7.3.1.2 Deep Level Ellipses	90
	3.7.3.2 Anaphoric References	90
	3.7.4 Processing of Null Responses from Database	92
	3.7.4.1 E-relation	93
	3.7.4.2 EG-relation	93
	3.7.4.3 EXC-relation	94
	3.7.4.4 V-relation	94
Chapter 4	Implementation	
4.1	Software Design Methodology	97
4.2	Classes of the System	98

4.2.1	Meaning Representation System	98
4.2.2	Machine Translation System	104
4.2.3	Natural Language Interface System	104
4.3	Platform Used	111
Chapter 5	Performance Evaluation of the Model	
5.1	Introduction	112
5.2	Performance of the NLI system for Information Retrieval	114
5.3	Performance of the Machine Translation System	125
Chapter 6	Conclusion & Future Work	133
Appendix	1	136
Appendix	2	137
Appendix	3	138
Appendix	4	139
References		140
Published work	of the Author	149

DESIGN AND DEVELOPMENT OF AN ADAPTABLE FRAME-BASED SYSTEM FOR DRAVIDIAN LANGUAGE PROCESSING

Abstract

The goal of natural language processing (NLP) is to build computational models of natural language for its analysis and generation. These computational models provide a better insight into how humans communicate using natural language and also help in the building of intelligent computer systems such as machine translation systems, man-machine interfaces, text analysis, understanding systems etc.

This work is aimed at building an adaptable frame-based system for processing Dravidian languages. There are about 17 languages in this family and they are spoken by the people of South India. These languages are free word order languages. Since most of the existing computational grammars are positional grammars, they are not suitable for the analysis and understanding of free word order languages. For the development of the prototype, Malayalam (A typical language of the Dravidian family and the native language of Kerala) language is considered in detail. But the frames developed for its internal meaning representation are easily adaptable for other languages coming in this group. This is because their grammar rules are almost the same and their vocabularies are closely related. They also exhibit structural homogeneity.

Karaka relations are one of the most important features of Indian languages. They are the semantico-syntactic relations between the verbs and other related constituents in a sentence. The karaka relations and surface case endings are

analyzed for meaning extraction. This approach is comparable with the broad class of case based grammars.

The efficiency of this approach is put into test in two applications. One is machine translation and the other is a natural language interface for information retrieval from databases. In the first case, simple/complex sentences in any Dravidian language could be converted to English. Here the source is free word order language while the target is a fixed word order one. Also the source is a verb-ending one while the target is a verb-central one. Since English is a fixed word order language, regular patterns could be identified in its structure. The source language sentence is mapped into one of these patterns which is most apt. Ambiguous translations due to word sense disambiguity are resolved by semantic tags attached to words.

In the second application a natural language interface (NLI) for information retrieval from databases is tried. As it is known, many of the shortcomings of the database languages could be overcome by putting an interface between the user's native language and the database language. Since nowadays, relational database management systems are de facto standards and SQL or SQL like languages are commonly used, the internal meaning representation is mapped to SQL commands. For a NLI to be useful, it must accept anaphora, ellipsis and other means of abbreviating utterances. It must be also capable of handling user misconceptions and producing quality responses. Methods to handle these are designed and implemented in a prototype NLI to databases. Entity-Relationship model is used to capture the structure of the database.

Object-oriented design methodology is used for the development of the prototype. In summary, this work makes the following contributions. It gives an elegant account of the relation between vibakthi and karaka roles in

Dravidian languages. This mapping is elegant and compact. The same basic thing also explains simple and complex sentences in these languages. This suggests that the solution is not just ad hoc but has a deeper underlying unity. This methodology could be extended to other free word order languages. Since the frames designed for meaning representation are general, they are adaptable to other languages coming in this group and to other applications.

Chapter 1

Introduction

1.1 Background

Human beings speak and understand natural language. But computers understand cryptic, artificial languages, which common people find difficult to understand and use. With the help of AI techniques and cognitive and linguistic theories, computers are gradually learnt to communicate in natural language. Developing a natural language understanding system that could be used in any kind of application still remains a dream.

Computers require a great deal of precision in communication. Some of the characteristics of natural language that seem to cause no problems for people but which create great difficulties for computers are ambiguity, imprecision, incompleteness and inaccuracy. Because of these basic problems of NLP, the central task in NLP is the translation of the potentially ambiguous natural language input into an unambiguous internal representation. There is no commonly agreed standard for internal representation and different types are found useful for different purposes. Translation of an utterance into an unambiguous internal representation requires inference based on potentially unbounded set of real-world knowledge.

1.2 Motivation and Scope

Automated Natural language understanding systems have several potential applications. They include natural language front-ends to databases and expert systems, machine translation, computer aided instruction systems etc. Many

computer systems offer a range of tools for data extraction, statistical analysis and graphical output. Since these tools have been developed independently, the user has to express identical commands to the different software packages in different ways. A natural language front end to all the different packages would enable the user to be more comfortable, without need to be mindful of the specifics of each such package. Thus users are interested in natural language as a standard man-machine interface. The computer software best suited to benefit from a natural language front end is the database. Databases hold huge quantities of data. There are several artificial languages for manipulating this data. But their usage needs knowledge about the database model, database structure, language syntax etc.

Many of the shortcomings of the database languages could be overcome by putting an intelligent interpreter between the user's native language and the database language. This method has several advantages. The interpreter can eliminate the necessity for the user to conform to an artificial syntax. It relieves the user from knowing about the details of the database model and data structure, data definition languages and data manipulation languages. The interpreter enables to understand incomplete or slightly erroneous queries, elliptical requests, anaphoric references etc. It can also recognize the logical inconsistencies in a query and warn the user. Thus an intelligent interpreter bridges the gap between the user and the database.

Several natural language front ends have been developed as a result of rigorous works done in this field of artificial intelligence. But majority of them uses English as the natural language. In an Indian context (India is a multilingual country and fifteen languages are included in the eighth schedule of the Indian Constitution) where only five percent of the population can boast

of education up to matriculation level and much less of them can work through English, their use is limited.

Machine translation helps in melting away language barriers. The world becomes culturally and intellectually united. However this still remains as the dream of people working in the fascinating research area of machine translation. The important problems that come in the way of machine translation are word sense selection, ambiguity in the sentence structure, pronoun reference and identification of tense and modality. These problems point to an important inference: A sentence must be “understood” before it can be translated.

1.3 Outline of the Work

This work is aimed at the development of an unambiguous understanding system for Dravidian languages. The meaning is extracted from the written text and is stored in a frame like structure. This structure is a generalized one so that it could be easily adapted to the potential applications of NLU like machine translation and man-machine interface. For the development of the prototype, Malayalam language is considered in detail. But the frame developed for its internal meaning representation is easily adaptable for other languages coming in this group. This is because their grammar rules are almost the same and their vocabularies are closely related.

Karaka relations are one of the most important features of Indian languages. They explain the relationship between the verbs and other related constituents in a sentence. They themselves do not impart any meaning, but tell how the nouns and other parts of speech are related to the verbs present in the

sentence. In this work the karaka relations are analyzed for sentence comprehension.

The system mainly consists of a morphological analyzer, local word grouper, a parser for the source language and a sentence generator for the target language. Simple and complex sentences are tried to comprehend. The first stage of sentence understanding is morphological analysis. For each word in the input sentence a lexicon is looked up and associated grammatical information is retrieved. It includes the parts-of-speech, gender, number, person, vibakthi form, tense, mood etc. The second stage is the building of an internal meaning representation structure. The knowledge representation technique selected is frames. For filling the various slots of this frame, expectation-driven parsing is used. The verb is first spotted. Since the languages under study are verb-ending, parsing starts from right most end. About 80 verbs belonging to the important verb-classes of movement, perception, emotions, vocation, transaction etc are considered. A frame corresponding to each of these verbs is stored. Frames belonging to same group have similar structure. After verb spotting the frame corresponding to that verb is loaded and the various slots are filled by finding the karakas involved in the sentence. The vibakthi forms are different for different karaka relations. For each verb depending upon its tense, mood and voice, the vibakthi endings differ for karaka relations. Hence a Vibakthi-karaka mapper has been developed.

In the case of complex sentences which contain more than one phrase, the individual phrases are first located. Here also verbs are used for demarcating the phrases. Then the meaning representation of component phrases are formed and they are added up to get the complete representation.

The efficiency of this approach is put into test in two applications. One is machine translation and the other one is a natural language interface for information retrieval from databases. In the first case simple and complex sentences (containing one principal clause and one subordinate clause) in a Dravidian language are converted to English. Here the source is free word order language while the target is a fixed word order one. Also the source is a verb-ending one while the target is a verb central one. Since English is a fixed word order language, regular patterns could be identified in its structure. Fifteen such patterns are used in the study and the source language sentence is mapped into one of these patterns which is found to be most apt one. Ambiguous translations due to word sense disambiguaty is resolved by semantic tags attached to words. Semantic tags are keywords that denote the real world usage of a word. This idea is closely related to the concept of reference in linguistics. In addition to this semantic tags a set of sense disambiguating rules are also used for the sense disambiguation of verbs and modifiers. This method of word sense disambiguation is simple though the effort required is high.

In the second application an NLI for information retrieval from databases is tried. The internal meaning representation is mapped to SQL commands. The prototype of the NLI developed, answers written Malayalam questions by generating SQL commands, which are executed by RDBMS. Complex queries are answered and in the cases of questions that can not be answered, co-operative messages are generated. The system is user friendly and can easily be configured for new database domains using the built-in domain editor. The built in domain editor helps the user to describe the entity types of the world to which the database refers. The system was tested with three database domains. A parliament election database ,a university academic database and a library database. The first one had 5 tables, the second had 7 tables and the

third had 5 tables. The size of the smallest table was 3 rows and that of the largest table was 1500. Each query was converted to a single SQL statement and the DBMS was left out to find the answer to the query, utilizing its own specialized optimization techniques. Thus the full power of the RDBMS was available during question answering.

The user misconception is an important cause of null responses. There can be extensional misconceptions and intentional misconceptions. For generating quality responses during the occurrence of null values, in addition to the relations in the database scheme, Event relations, Event-Graph relations, Exception relation and View relations are added to the knowledge base.

The natural language input becomes ambiguous due to anaphora and ellipsis. A question is called elliptical if one or more of its constituents are omitted. Ellipses could be of two types - Surface level ellipsis and deep level ellipsis. Surface level ellipsis is handled by recognizing input as elliptical and getting in some antecedents and constructing the complete sentence. Deep level ellipsis is handled by the use of domain's knowledge and user interaction. Anaphora refers to the general pronouns and definite noun phrases present in the query. Anaphoric references are resolved by two filter tests. Gender-number-person test and tests using the knowledge about the structure of the domain.

Chapter 2

Natural Language Processing – A Review

2.1 Introduction

Natural language processing is a technology with the ambitious goal of making communication with computers as easy as it is with people. Natural language understanding and natural language generation are the two components of natural language processing. The phrase NLP generally refers to language that is typed, printed or displayed rather than spoken [1]. Understanding spoken language is the focus of speech recognition and speech understanding.

A major division arises within NLP - general NLP and applied NLP. General NLP could be thought of as a way of tackling cognitive psychology from a computer science viewpoint. The goal is to make models of human languages usage and to make them computationally effective. Examples are general story understanding systems developed by Charniak [2] Schank [3] and Carbonell [4] and dialogue modeling systems developed by Cohen and Perrault [5], Grosz [6], Sidner [7] and others. These works have showed that general NLP requires a tremendous amount of real-world knowledge. Because of the difficulty in handling the amount of knowledge required for these tasks, the systems constructed in this area tend to be pilot systems that demonstrate the feasibility of concept or approach, but do not contain enough knowledge base to make them work on more than a handful of carefully selected passages or dialogues.

On the other hand applied NLP allows people to communicate with machines through natural language. It is less important in applied NLP whether the machine understands its natural language input in a cognitively plausible way than whether it responds to the input in a way helpful to the user and in accordance with the desires expressed in the input. Examples are database interfaces developed by Hendrix [8], Grosz [9], Kaplan [10] and others, and interfaces to expert systems as in the work of Brown and Burten [11] and J.G. Carbonell et al [12]. These systems should be capable of detecting and resolving errors and misunderstandings by the user.

For computers natural languages are ambiguous, imprecise, incomplete and inaccurate. Some of the factors that contribute to the ambiguity of natural language are multiple word meaning (eg. *The man went to the bank to get some cash. The man went to the bank and jumped in*), syntactic ambiguity (eg. *I hit the man with the hammer*) and unclear antecedents (eg. *John hit Bill because he sympathized with Mary*). People often express concepts with vague and inexact terminology. Consider the sentences *I have been waiting in the doctor's office for a long time, The crops died because it hadn't rained in a long time.* Without conceptual familiarity, a computer would not be able to differentiate between the two different lengths of time represented by the same phrase.

People often do not say all what they mean. Their expectation of likely events in a particular situation enables them to understand information that was not included in the text. To be able to comprehend incomplete information, a computer must possess the same kind of situational expectations. Human beings are capable of understanding inaccuracies like spelling errors, transposed words, ungrammatical constructions, incorrect syntax, incomplete sentence, improper punctuation etc. A computer designed to understand

natural language must be able to understand inaccurate uses of languages at least as well as a person. One way to resolve linguistic ambiguity is by understanding an idea in context. The problems of imprecision could be avoided by identifying it with situations that are familiar. The problems of incompleteness, could be overcome by experience or expectations in certain situations.

Because of the several basic problems of NLP given in earlier paragraph, the central task in NLP is the translation of the potentially ambiguous natural language input into an unambiguous internal representation. There is no commonly agreed standard for internal representation and different types are useful for different purposes. In general NLP, translation of an utterance into an unambiguous internal representation requires inference based on potentially unbounded set of real-world knowledge. Consider for instance: *John went out to a restaurant last night. He ordered steak. When he paid for it, he noticed that he was running out of money.* To answer question like *what did Johan pay for? Did John eat the steak ?*

information on restaurants, ordering, eating and other real world topics are required. Knowledge representation techniques have not yet developed to the stage where they can handle to an acceptable level of efficiency the larger quantities of such knowledge required to do a complete job of understanding a large variety of topics. Current general NLP systems are demonstration systems that operate with a very small amount of carefully selected knowledge specifically designed to enable the processing of a small set of example inputs.

Applied NLP systems tackle this problem by taking advantage of the characteristics of the highly limited domains in which they operate. Since the domain is restricted the amount of knowledge that must be represented and the

number of interfaces that must be made could be reduced to a manageable level. The current state of art of Applied NLP is natural language interfaces capable of handling a limited task in a limited domain. Each task and domain that are tackled require careful pre-analysis so that the required inference can be pre-encoded in the system, thus making it difficult to transfer successful natural language interfaces from one task to another. In the language craft (Carnegie-group Inc.) approach this difficulty is reduced by providing a development environment and grammar interpreter which drastically shorten the development of new domain specific interfaces.

2.2 Natural Language Analysis Techniques

A brief overview of the techniques commonly used for the analysis of natural language sentences is given next.

2.2.1 Pattern Matching

In pattern matching approach, the interpretations are obtained by matching patterns of words against input utterances. Associated with each pattern is an interpretation, so that the derived interpretation is the one attached to the pattern that matched. Pattern matching systems are also called template systems. ELIZA system of Weizenbaum [4] is an example of a Pattern matching system. The carefully selected task of ELIZA was to simulate a psychologist as he interviewed a patient. ELIZA did not construct an internal representation of its input as such, but directly went from the input to its reply. The input was matched by a small set of single-level patterns, each of which was associated with several replies. For example if the user typed *you are X*, ELIZA could respond *What makes you think I am X*, where *X* is an adjective.

Several AI programs used templates. SIR (Semantic Information Retrieval) by Bertram Raphael is an example. It could do enough logical reasoning to answer question such as *Every person has two hands. Every hand has ten fingers. Joe is a person. How many fingers does Joe have?*. Another was Bobrow's STUDENT which could solve high school level mathematical problems stated in English. A worthy feature of STUDENT was that it used recursive templates. Norvig re-implemented it in Common LISP [13].

To make more complete analysis of the input using the same techniques would require far too many patterns. Many of these patterns would contain common sub elements since they refer to same objects or had the same concepts managed with slightly different syntax. In order to resolve these problems, hierarchical pattern-matching methods have been developed in which some pattern matches only part of the input and replace that part by some canonical result. Other higher level patterns can then match on these canonical elements in a similar way, until a top level pattern is able to match the canonicalized input as a whole according to the standard pattern matching paradigm. The best-known hierarchical pattern matching is the PARRY systems of Colby [14]. Like ELIZA, this program operates in a psychological domain but models a paranoid patient rather than a psychologist. Pattern matching is a quick way to extract useful information from natural language input, adequate for many practical user-interface applications. HAL, the English language command interface for the Lotus 1-2-3 spread sheet programs is a typical template system.

2.2.2 Keyword analysis

An alternative to template matching is keyword analysis. Instead of matching the whole sentence to a template, a keyword systems looks for specific words

in the sentence and responds to each word in a specific way. One of the first key word systems was that of Blum developed in 1966. Blum wrote a program to accept sentences such as *Copy 1 file from U1 to U2, binary, 556 bpi*, and convert them into commands for a program that managed magnetic tapes. Though he took ELIZA as his model, Blum ended up using a quite different technique. He distinguished these kinds of key words *requests* (list, copy, backspace etc.) *qualifiers* that further described the request, such as binary and *quantities* such as 5 files or 556 bpi. His program simply collected all the requests, qualifiers and quantifiers in the input and put them together into a properly formed command, paying very little attention to the word order and completely ignoring unrecognized words. Keyword systems have had a long and successful history. Unlike template systems, they are not thrown off by slight variations of wording. Unrecognized words are simply skipped. These prominent keyword systems are AI Corp's Intellect, Symantec's Q&A, and QP&QC system developed by Wallace [15]. All of them are data base query systems. Key word systems work well for database querying because each important concept associated with the database has a distinctive name.

2.2.3 Syntactically Driven Parsing

Syntax deals with the rules for combining words into well-formed phrases, clauses and sentences. In syntactically driven parsing the interpretation of larger groups of words are built up out of the interpretations of their syntactic constituent words or phrases. In this sense it is just opposite of pattern matching, in which the emphasis is on interpretation of the input as a whole. In this method syntactic analysis is done completely first and then the internal representation or interpretation is built.

Syntactic analysis is obtained by application of a grammar that determines which sentences are legal in the language being parsed. The method of applying the grammar to the input is called parsing. The important grammar representation formalisms are listed below.

2.2.3.1 Context-free Grammar

It has been one of the most useful techniques in natural language analysis. In context-free grammar, symbol on the left side of a rewrite rule may be replaced by the symbols on the right side, regardless of the context in which the left side symbol appears. It has the advantage that all sentences structure derivations can be represented as a tree and several good practical parsing algorithms do exist. The context-free grammar consists of rewrite rules of the following forms.

S → NP VP
NP → DET N | DET ADJ N
VP → V NP
DET → the
ADJ → red | big
N → Car | child
V → hit | jumped

This grammar could generate the sentence *The car hit the child*. The parse tree for the sentence is as shown in fig 2.1 Although it is a relatively natural grammar, it is unable to capture all the sentence constructions found in English. Context-free nature of the grammar does not allow agreements such as the one required in English between subject and object.

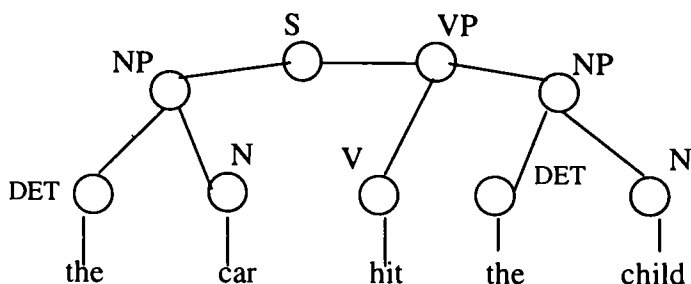


Fig 2.1 A parse for “the car hit the child”

Grammar that allowed passive sentences, required completely different set of rules to handle active sentences, even though both of them have the same internal representation. This leads to exponential growth in the number of the grammar rules. Gazdar (18) has tackled these problems to some extent by adding augmentation to handle situations that do not fit basic grammar.

2.2.3.2 Transformational Grammar

Linguists tackled the problems specific to context free grammars, in particular Chomsky through transformational grammar. A transformational grammar consists of a dictionary, a phrase structure grammar and set of transformations. In analyzing sentences, using a phrase structure grammar, first a parse tree is produced. This is called the surface structure. The transformation rules are then applied to the parse tree to transform it into a canonical form called the deep or underlying structure. As the same thing can be stated in several different ways, there may be many surface structures that translate into a common deep structure. Although the regularities of natural language are accounted much better by transformational grammar than context free grammar, its computational effectiveness is not very good. It enables to produce a sentence starting from the symbol S. Running the model

in the reverse direction is highly non-deterministic. Hence parsers based on transformational grammar have not played a major role in NLP.

2.2.3.3 Augmented Transition Network

As a response to the problems of transformational grammar, Bobrow and Fraser [19] proposed and Woods [20] subsequently developed a method of expressing a syntactic grammar that was computationally tractable and could capture linguistic generalizations in a concise way than transformational grammar. The formalism Wood developed was known as an augmented transition network (ATN). It consisted of a recurrent transition network augmented by a set of registers that could be used to save intermediate results or global state. An example of ATN is shown in fig 2.2 This network can recognize simple sentences of active, passive, declarative and interrogative types with just a subject, verb, and direct object. The symbols attached to the arc show what constituent must be recognized to traverse the arc. AUX is an auxiliary verb (like 'is' or 'have'). NP is a noun phrase, which is defined by another network of the same formalism as this arc. V is a verb and *by* is the word 'by'. The numbers on the arcs serve as indices to the table 2.1, which lists the tests that must be true to traverse the arcs and the action that must be performed as the arc is traversed.

In this LISP –like notation, the asterisk refers to the constituent just parsed and SETR sets a register, whose name is specified by its first argument, to the value of its second argument.

Test	Actions
1. T	(SETR V*) (SETR TYPE' QUESTION)
2. T	(SETR SUBJ*) (SETR TYPE' DECLARATIVE)
3. (agrees* V)	(SETR SUBJ*)
4 (agrees SUBJ*)	(SETR V*)
5. (AND (GETF PPRT) (= V 'BE)	(SETR OBJ SUBJ) (SETR V*) (SETR AGFLAG T) (SETR SUBJ 'SOMEONE)
6. (TRANS V)	(SETR OBJ*)
7. AGFLAG	(SETR AGFLAG FALSE)
8. T	(SETR SUBJ*)

Table 2.1. Tests and Actions for the ATN in Fig 2.2

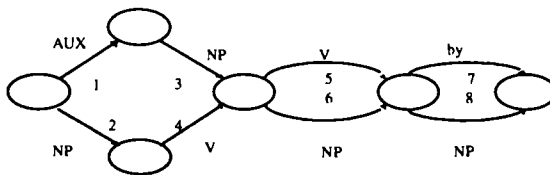


Fig 2.2 An example of ATN

Very large ATN grammars of several hundred nodes that capture large subjects of English have been developed. However, ATNs also have several disadvantages.

Complexity and Non-modularity

As the coverage of an ATN increases, the structural complexity also increases. Modification or augmentation of an existing ATN would cause unforeseen side effects. For instance, suppose a new outgoing arc is added to a node with a large number of incoming arcs to handle an additional type of phrase which is a valid continuation of the parse represented by one of the incoming arcs, it could lead to spurious and incorrect parses when the node is reached via a different incoming arc. Fan-out and fan-in factors of 10 or 20 are not uncommon in large realistic grammars.

Fragility

The current position in the network is a very important piece of state information for the operation of an ATN. If input is slightly ungrammatical, even by a single word, it is very hard to find the appropriate state to jump to continue parsing. Kwasny and Sondheimer [21] and Weischedel and Black [22] had done good works in dealing with ill-formed input in natural language. Bates [23] in his work on island-driven ATN parsing, had given methods to solve this problems in speech input.

Inefficiency through back tracking search

Traversing an ATN requires search. The natural way to search an ATN is through back tracking. Because intermediate failures are not remembered in such a search, the possibility of repetition of the same sub parses arrived at through different paths in the network, is high. Chart parsing techniques were designed as alternatives to ATNs precisely to avoid these inefficiencies.

2.2.3.4 Semantic Grammar

It is a context free grammar in which the choice of non-terminals and production rules is governed by semantic as well as syntactic function. Semantic grammars were introduced by R.R.Burton for use in SOPHIE, a Computer-aided Instruction systems for electronic circuit debugging [24]. The goal was to eliminate the production of meaningless parses for practical systems in limited domains. It is often more useful to use meaningful semantic components instead of syntactic constituents such as noun phrases, verb phrases, prepositions etc. Thus in place of nouns when dealing with a naval database, one might use ship, captains, ports, cargoes etc. This approach gives direct access to the semantics of a sentence and substantially simplifies and shortens the processing.

Hendrix et al [8] developed a system named LIFER with semantic grammar. The rules had the following format

```
S → <present> the <attribute> of <ship>
<present> → what is/[can you] tell me
<attribute> → length | beam | class
<ship> → the <ship name>| <class name> class ship
<ship name> → Kennedy| enterprise
<class name> → Kitty hawk| lafayettee
```

An expanded version of this grammar was used for access to a database of information about U.S. Navy ships in the LADDER systems[25]. In addition to defining a grammar, LIFER also allowed an interface builder to specify the interpretation to be produced from rules that were used in the recognition of

input. One semantic action was associated with each rule. Database query language statements were generated as a direct result of the recognition.

The principal advantages of semantic grammars are:

- When the parse is complete, the result can be used immediately without the additional stage of processing that would be required if a semantic interpretation had not already been performed during the parse.
- Many ambiguities that would arise during a strictly syntactic parse can be avoided since some of the interpretations do not make sense semantically and thus cannot be generated by the semantic grammar.
- Syntactic issues that do not affect the semantics can be ignored.

Some drawbacks to the use of semantic grammars are

- New grammar is to be developed for each domain, since the semantic categories for each domain will be quite different.
- The number of rules required can become very large since many syntactic generalizations are ruined.
- Because the number of grammar rules may be very large, the parsing process may be expensive.

TEAM system [9] is an attempt to resolve the above problem. It focuses on a specific class of applications, access to relational databases, and abstract out the linguistically common aspects of a semantic grammar for such a class.

Building a specific interface, then, requires only instantiating a template with the vocabulary and morphological variation required for a specific database. This approach has the potential to produce highly efficient natural language interfaces, at the expense of the inability to go beyond a particular class. DYPAR system [29] combines the strengths of semantic grammars, syntactic transformation and pattern matching into a single system that maps structures into canonical forms before attempting to use the full semantic grammar. That allowed many redundant and unnecessary constructions to be eliminated. Although richer in expressive power, this approach demands more sophistication of the grammar writer, requiring knowledge of how to write transformation, context free rules and patterns.

2.2.3.5 Lexical Functional Grammar

It is a strong computational formalism that addresses how to extract grammatical relations from a sentence in a positional language such as English. LFG has been designed by Kaplan and Bresnan [26]. It postulates two levels of representation: one based on constituent structure and the other on grammatical functions such as subject, object etc. In fixed word order languages like English, positions are used for coding both theta relations. Considerable effort had gone into the design of LFG so that it can deal with and separate these two kinds of information. Mapping from grammatical functions into theta roles is enumerated exhaustively in the lexicon.

LFG formalism has two major components, a context free grammar and a functional specification. The former gives the c-structure for a sentence and the latter gives the f-structure. The major strength of LFG is that it gives explicit algorithms for extracting grammatical functions. Its weakness is that it

does not offer any theory regarding lexical ambiguity, adjuncts and optional theta roles and mapping from grammatical relations to theta roles.

2.2.3.6 Government and Binding

It is the dominant linguistic theory. Its goal is to identify the innate structure in human mind, which enables a child to acquire language so effortlessly. It does not address the problems of either parsing or generation. As a result it proposes its formalism in a form which is not amenable to computation directly. GB keeps changing so much and so rapidly that it is difficult to know what GB is at any given time and implement it. Hence this theory is not popular with computational linguistics. GB grammar has three levels of representations of a sentence - D-structure, S-structure and LF- representation. In the GB model a crucial role is played by interacting systems of principles. These systems of principles are X-bar theory, thematic theory, government case theory, bounding theory and control theory. These systems of principles place constraints thus filtering out ungrammatical representations. Typically, various principles have some parameters associated with them. These parameters are meant to make the grammar flexible enough to account for all the different languages.

2.2.4 Case Grammar

It is a form of transformational grammar in which the deep structure is based on cases – semantically relevant syntactic relationships. Case grammar was proposed by Charles Fillmore. In this formalism syntactic and semantic interpretations are combined [27]. The central idea is that the deep structure of a simple sentence consists of a verb and one or more noun phrases associated with the verb in a particular relationship. Fillmore proposed the

following cases: agent, experience, instrument, object, source, goal, location, type and path. The cases for each verb form an ordered set referred to as a 'case frame'. It indicates that the verb open always has an object. But the instrument or agent can be omitted. Thus the case frame associated with the verb provides a template which builds in understanding a sentence. Consider the sentence. *John killed Jim with a knife for Mary.* The case frame corresponding to this sentence is

```
[Kill
[Case frame
Agent      : John
Dative     : Jim
Instrument  : Knife
Beneficiary : Mary
Co-agent   :
Location   :      ]

[Modals
time       : past
voice     : active ]]
```

Case frame differ noticeably from simple, purely syntactic, parse trees. The relation between the head of the case frame and the individual cases are defined semantically, not syntactically. Hence a noun in the subject position can fill the agent case, as in the example above or it can fill an object case as in *the window broke* or it can fill the *instrument* case as in *the hammer broke the window*. Since the purpose of a natural language interface is to extract the semantics of the input, case frame representation is powerful than syntactic parse trees. Each case frame defines some required cases, some optional cases and some forbidden cases. A required case is one that must be present in order for the verb to make sense. An optional case is one that, if present, provides more information to the case frame representation but, if absent, does not harm its semantic integrity. Forbidden cases are those that cannot be present with the head verb.

2.2.5 Conceptual Dependency

It is a semantic representation formalism developed by Schank [28]. It attempts to represent every action as a composition of one or more primitive actions, plus intermediate states and causal relations. Consider the sentences *John gave Mary a ball* and *Mary took a ball from John*. Even though these sentences differ syntactically, both sentences express the proposition that a ball was transferred from John to Mary. In CD the primitive action ATRANS could be used to encode the semantics of these verbs (took and gave). The CD representations of these sentences are given below.

[ATRANS

rel	possession
Actor	John
Object	Ball
Source	John
Recipient	Mary]

John gave Mary a ball

[ATRANS

rel	:	possession
Actor	:	Mary
Object	:	Ball
Source	:	John
Recipient	:	Mary]

Mary took a ball from John

These two structures determine precisely in what aspects these two propositions differ and in what aspects they are identical. Moreover inference rules associated with ATRANS could be invoked automatically when *give* and *take* verbs are parsed.

Thus the CD representation of a verb is at a lower level than that of a verb in a case grammar. It provides a greater degree of predictive power. The first step

in mapping a sentence into its CD representation involves a syntactic processor that extracts the main noun and verb. It also determines the syntactic category and aspectual class of the verb. The conceptual processor then takes over. It makes use of a verb-ACT dictionary, which contains an entry for each environment in which a verb can appear. Once the correct dictionary entry is chosen, the conceptual processor analyses the rest of the sentence looking for components that will fit into the empty slots of the verb structure.

2.2.6 Expectation-Driven parsing

It refers to a family of natural language understanding systems. An expectation driven parser is a top-down parser that looks for concepts rather than grammatical elements. Unlike a syntactic parser which might look for a noun phrase followed by a verb phrase, an expectation driven parser would look for an action followed by an object that the action could apply to. The availability of conceptual structures during the parsing process is the most distinctive feature of expectation-driven parses. A proper description of any particular expectation driven parser needs to include the kinds of text it was intended to handle and the kind of syntax it was extended to talk to [29].

A request has two points, a test and an action. Expectation driven parsing algorithms uses a list of active request, initially empty and a set of global variables where the actions of the requests can put conceptual structures. The first request based expectation driven parse was developed for the MARGIE inference system. MARGIE was a program developed by Roger Schank and his students at the Stanford AI laboratory in 1975. Its intent was to provide an intuition model of the process of natural language understanding.

The MARGIE system has 3 components - A conceptual analyzer, an inferencer and a text generator. The analyzer takes English sentences and converts them into an internal conceptual-dependency representation. The analyzer reads a sentence from left to right word by word, putting the requests attached to each word in a list and executing the actions of any requests whose tests are true. Usually the basic conceptual framework for the sentence is built as soon as the main verb's requests are loaded. The various slots of this frame would be filled while processing the remainder of the sentence.

The inference accepts a proposition stated in CD and deduces a large number of facts from the proposition in the current context of the system's memory. The reason behind this component was the assumption that humans understand far more from a sentence than is actually stated. Sixteen types of inferences were identified, including case, effect, specification and function. The inference knowledge was represented in memory in a modified semantic net. Consider the sentence *John hit Mary*. From this the systems may infer

John was angry with Mary.

Mary might hit John back.

Mary might get hurt.

The text generation module converts the internal CD representation into English-like output. MARGIE runs in two modes. In *inference mode*, it would accept a sentence and attempt to make inferences from that sentence as described above. In *paraphrase mode*, it would attempt to restate the sentence in as many equivalent ways as possible. For example, given the input *John killed Mary by choking her*, might produce the paraphrases

John strangled Mary.

John choked Mary and she died because she was unable to breathe.

The successor to the MARGIE parse was ELI (English Language Interpreter [29]). It was a part of the SAM (Script Applying Mechanism) [30] systems. The goal of the SAM system was to read short text about events occurring in a stereo typical situation like story about going to a restaurant, a car accident or a news paper report and understand events that were not mentioned explicitly in the text and answer questions about what did and did not occur in the story. ELI made more explicit the functional structure of requests. A new field called SUGGESTIONS was added to each request, and it contained request to be activated if the first request was executed. Requests adding requests were more common in ELI than in the MARGIE parser. Nesting request in this way reduced the number of requests active at any one time and increased the expectational nature of ELI.

ELI also made explicit in each request, which variables its action affected. By doing this, it was possible to dynamically chain requests together, resulting in several improvements in request management. When the value of a global variable changed, ELI could immediately tell which tests of which requests might be affected. A request whose action would set a variable to a value that wants no test could be rejected immediately. Default constraints placed on variables could be propagated automatically to these requests where actions were attempting to fill those variables. Once a variable was filled, other requests attempting to fill that some variable could be removed from the list of active requests as they are no longer needed.

ELI was succeeded by the conceptual Analyzer (CA) [31]. CA's Control Structure was simpler than ELI. CA hid requests when parsing noun phrases and kept a list of built concepts called the C-LIST, from which larger structures were built. Request could not only test for particular concept in the

C-LIST but could also test the order in which concepts appeared in the C-LIST. CA's active request list was sub divided into pools where each pool contained those requests that had been added at the same time. CA always had the most recent requests first, so that a loose stack discipline was maintained for reducing the number of requests considered. AD-HAC developed by Cater was an expectation driven parser. It was developed to deal with very ambiguous sentences using a preferential approach [32]. AD-HAC's theories were a way of exploring possible conceptual parses in a best first manner. One novel feature of AD-HAC was that it used an ATN syntactic parser to pre analyze the input tests and label verb groups, noun groups, prepositional phrases and conjunctions. It removed the need for requests attached to articles and other function words.

Micro ELI [32] also used an expectation driven parser. Each request had 3 fields; TEST, ASSIGN, and NEXT-PACKET. The ASSIGN field reduced all actions to variable assignment. The NEXT-PACKET field of a request contained requests to be added if the request was executed. As in CA, requests were added in separate pools rather than merged into one list. The most recently added packet requests had to be used or removed before the next most recent packet could be accessed.

The MARGIE parser, ELI and CA used lexically indexed requests to construct conceptual forms with only simple syntactic cues. Several other systems while preserving the primary goal of producing conceptual representations proposed significant alternatives in control structure.

2.2.7 Word-Expert Parsing

Small and Riegers [33] had developed this parsing formalism in a Computer program that analyses fragments of natural language text in order to extract their meaning in context. The parser successfully analyses input fragments rich in word-sense ambiguity and idioms and handles a range of interesting syntactic constructions.

Word-expert parsing (WEP) views individual words as active lexical agents called word experts, which participate in the overall control of the parsing processing. Each active word expert must (a) determine its own meaning or function role in the larger text and (b) provide conceptual and control information to other experts to enable them likewise to coordinate this complex task. Isolated word sense discrimination is not possible, since much of the dynamic knowledge required to complete that task must come from other experts. Thus each expert must both ask questions of other expert and answer ones posed to it.

Each word is modeled by a separate computer program and parsing takes place through the successive execution of the expert programs corresponding to the words of the input. The effect of these programs is to augment the overall meaning representation and to alter the control flow of the overall process. One problem in reading from left to right is that some times the meaning of a word does not became clear until later in the sentence, a few more words have been seen. Since each word expert is responsible for fitting its word on to the overall interpretation of the sentence, it sometimes needs to wait a while. The individual word expert thus affects the high-order processing of the word by waiting for what it needs and then forcing itself

back into the action, obviously disrupting the normal left-to-right flow of things.

The parser was developed in Maryland LISP on the Univac 1100/42 at the university of Maryland. It operates with a small vocabulary of 40 implemented word experts. The existing collection of experts is sufficient to analyze sentences containing many different contextual usages of the content words *eat, deep, throw, pit, case, by* and *out*. The analysis of a sentence containing such a word entails the determination of exactly what it means in context. The parser correctly determines the meaning of fragments such as *throw in the towel, throw the ball in the pit, throw out the garbage, throw out the court case* and *throw a party*. G.Adrianes of University of Leuven, Belgium has developed a version of WEP to analyze Dutch sentences with multiple lexical ambiguities particular to that language.

2.2.8 Integrated Partial Parser

Contrary to WEP, Integrated Partial Parser (IPP) simplified word definition to the base conceptual minimum [34]. The most ambiguous words like *be* and *false* had no definitions at all. Parsing was driven by requests attached to concepts pointed to by more meaningful words like *hijack* and *shooting*. IPP was designed to parse hundreds of newspaper articles about terrorism and store them in a hierarchical long-term database using memory structure called the memory-organization packet. The parser was intelligent though to face unknown words and unexpected grammatical constructs. Words are divided into 5 classes: event builders, event refiners, token makers, token refiners and function words. IPP was not as verb-centered as most expectation driven parsers.

2.2.9 Connectionism

It is a highly parallel computational paradigm that supports many intelligent activities like vision, knowledge representation, natural language understanding, learning etc. There are several reasons to believe that connectionist algorithms are suitable for language users. Language is acquired by some form of learning. Human linguistic activity continually adapts to its environment. Thus, models which show learning and adaptation should be preferred to models in which adaptation is difficult. Rule based systems are brittle in the sense that they often fail to generalize across input context. Networks can often make accurate classification from partial data and they are sensitive to context. They can handle ill-formed ungrammatical input and are able to generalize novel outputs, by generating combinations of previously trained outputs. A brief overview of some important works done in NLP using this approach is given below.

One early influential connectionist model for natural language concepts has been the model of learning the past tense of verbs developed by Rumelhart and McClelland [35]. This model demonstrated that rule like behavior for the building of past tenses could be realized in a connectionist pattern associator which did not contain any explicit symbolic rules. Each word was represented with a set of overlapping triples called wickelphones. This set of wickelphones for a word was coded into a distributed representation of 460 phonetic features called wickelfeatures. Sequence was not represented explicitly in this model, but implicitly in the parallel representation of overlapping triples of wickelphones.

Another model which used a spatial parallel representation, focussed on case role assignment in sentences [36]. A pattern associator learned and

represented the syntactic and semantic knowledge for making case role assignments. The input was a representative of the surface structure of a sentence. The output of this model was the representation of the case role. While this model could deal with several difficult problems of structural disambiguation and lexical disambiguation, its representation was restricted in length by the predefined number of surface structures and case role units. This factor inhibited the processing of longer or more complex sentences.

Sliding windows model could represent a restricted part of a natural language sequence spatially. Instead of presenting the whole sequence to the network, only that part that could fit into the window was sent to the network. By moving the window across the whole sequences, sequences of an unrestricted length could be processed. Sejnowski and Rosenberg had used this technique in NET talk architecture [37]. A window of seven letters was moved over text and the task of the network was to produce the central phoneme corresponding to the fourth of the seven letters. Sliding window technique has the disadvantage of restricting the sequential context by the length of the window.

A. Waibel et al [38] developed a Time Delay Neural Network (TDNN) for sequence learning in speech processing. Each TDNN unit receives the current input as well as the input of previous time steps. Hence each unit can keep track of the history of input values which can support sequentiality. Based on these TDNN units, various TDNN architectures have been successfully built in modeling various aspects of time-dependent speech analysis.

While spatial parallel representations and sliding windows use fixed length of the sequential context, recurrent models represent arbitrarily long sequences. M.I. Jordan [39] proposed a recurrent model for processing sequences which

represents plans, states and actions as distributed vectors for a certain plan. This network generates a corresponding sequence of output actions.

While a Jordan network can represent the preceding context, it also depends crucially on the values of the output units. Rather than using the output units for recurrent feed back to the next input, Elman [40] has suggested representing context by connecting the units of a hidden layer to the input layer. This enables the network to use the learned distributed internal representation of the preceding context rather than the values of the output units of the directly preceding step. These Elman networks have been trained and tested on prediction tasks for which the next item of the sequence is predicted based on the current input and the preceding internal context. A typical prediction task is predicting language sequences based on simple grammar.

Pollack J.B [41] developed a Recurrent Recursive Auto Associate Memory (RAAM), model for representing recursive data structures. The architecture of a RAAM is a feed forward model consisting of a compressor and reconstructor.

The compressor maps the input into an internal reduced representation which can be decoded by the reconstructor. The recursive use of reduced internal representation for input and output in the RAAM leads to learning compositional representations in a dynamically changing environment, since the internal representations evolve over time starting from random values. In general, recurrent models like Jordan networks, Elman networks and RAAM are more powerful than spatial models, since they can process language sequences of arbitrary length while spatial models are restricted by the size of the network or the use of sliding window.

Another model for representing and learning syntactic, semantic and inferential knowledge of natural language concepts is the sentence gestalt model developed by St. John and Mcclleland [42]. It is based on an extended Jordan network with additional hidden layers. Starting with the first constituent, the network is trained to predict the role/filler pairs of the complete event, even though the corresponding constituents occur only later in the sentence. Using this architecture, natural language processing aspects like word disambiguation, role assignments, simple form of garden path effects could be demonstrated. This model is not capable of representing embedded structures or phrases attached to single constituents.

Cottrell et al [43] gave an integrated connectionist model for grounding language in perception. It combines two feed forward encoder networks, which compress faces and names in their hidden layers. Association between names and faces are learned by connecting the hidden layers of the two encoder networks via additional hidden layers. After a separate training of the two encoders and a subsequent training of the mutual association, this model could associate names with faces and vice versa. This work has been extended towards dynamic grounding for very simple movies.

Recently several hybrid models combines certain advantageous properties of symbolic representation with connectionist representations have been emerged. One of the first hybrid models for language understanding, developed by Pollack [44], represents syntactic, semantic and contextual knowledge for sentence understanding. A symbolic parser is used to generate the syntactic nodes and connections of a local connectionist network. These syntactic nodes interact with the manually encoded lexical, semantic and contextual nodes of the local network. The local connectionist model is based

on interactive architecture in which competing syntactic, lexical semantic and contextual constraints can be integrated in parallel. This integration allows parsing difficult garden path sentences, which have some initial syntactic structure or semantic interpretation, which has to be connected later when additional semantic or structural knowledge is available. The main contribution of this work is the interactive view of lexical, syntactic, semantic and contextual knowledge in a parallel hierarchical localist network. The main weakness of this approach is the hand coded semantic and contextual part of the local network that has to be generated for particular sentences, since learning semantics is not part of the original model.

The hybrid symbolic connectionist model, CIRCUS by Lehnert [45] focussed more on conceptual analysis. This model combines a symbolic syntactic analysis, a symbolic semantic top down analysis and a local connectionist bottom-up analysis based on numerical relaxation. Top-down predictions for conceptual frames can be triggered by specific associated words and interact with bottom-up predictions from the local network in order to transform sentences into conceptual frames. The syntactic analysis is based on a stack oriented conceptual analyzer that stores sentences fragments in global syntactic buffers. Each time a new syntactic fragment has been found, the predictive semantics module takes control to check if a slot in a top-down concept frame can be filled. Each slot in a concept firm can have associated hard constraints, which must be fulfilled and soft constraints, which may be fulfilled. Since soft constraints may be violated they provide a mechanism for robust symbolic conceptual analysis. A local relaxation network is created for solving attachment problems. If the local network is able to resolve an ambiguous attachment, an additional slot can be inserted into a concept frame. In this way, top down predictive slot filling is combined with bottom-up data-driven slot insertion. An extended version of this conceptual analyzer has

been successfully applied to the parsing and categorizing of stories into various claims of conceptual frames.

Hendler [46] has developed a hybrid symbolic/connectionist model that combines symbolic marker parsing with connectionist processing for making inferences in a natural language environment. The task of the marker parser is to find inference paths in a symbolic semantic network. However a semantic network can not anticipate all potentially useful connection to similar objects. Therefore connectionist networks are used for representing similarity between various objects. When a symbolic marker reaches a node in the semantic network that is associated with micro features in a local network, the activation of the marker in the semantic network is used to initialize the activator of the associated nodes in the local network. Then activation spreads within the local network and similarities between objects can be detected. If a node in the local network receives enough activator it can activate a marker in the semantic network and marker parsing continues. This enables the network to make unanticipated associations between similar objects.

2.3 The Indian Context

India is a multilingual country. Developments in technology world wide are influencing Indian languages. Indian languages exhibit a high degree of structural homogeneity even though the scripts are different. Demands for computer systems with input/output facilities in these scripts have been growing steadily. The work in the direction of developing natural language understanding systems for Indian languages was started in India by the initiative of the Department of Electronics (DoE), government of India in the mid 1980's. The motivations of DoE were to make computers reach the common man, development of application software for Indian languages,

development of Indian language interfaces to various application packages like databases, spreadsheets etc.

Many projects were initiated with DoE funding at many educational and research institutions all over the country [47]. Indian Institute of Technology (IIT), Kanpur and Center for the Development of Advanced Computing (C-DAC) are in the forefront of the works done in this area. C-DAC's GIST technology and ISFOC font standards has made Indian languages usable on computers along with many accepted software. GIST technology also makes it possible to transliterate between different scripts. Transliteration packages are being used increasingly for printing telephone directories, passenger lists, degree certificates etc in bilingual formats. Solutions both in the form of hardware and software are being offered for word processing, desktop publishing and data processing applications.

These developments have paved the way for serious research in the areas of natural language processing like machine translation, computer assisted teaching, corpora development, natural language interfaces etc.

Machine Translation : IIT Kanpur is actively involved in this field from early 1980's. The important projects carried are ANUSARAKA [48,49] and ANGALABHARATHI [49]. In 1984, Prof. R.M.K Sinha, proposed an interlingua approach using Sanskrit as the intermediate language for meaning representation. During 1986-88 a prototype for translation from Hindi to Telgu was developed by Chaitanya and Sangal which was capable of translating very simple sentences with limited vocabulary. A karak based conceptual graph was used for internal representation and disambiguation. Then it was realized that for Indian languages, which are structurally close to each other, a direct lexical substitution in a language pair could lead to

reasonable results in a simplistic manner. This led to the development of the software ANUSARAKA. Here the commonality of the source and destination language (both being Indian languages) are exploited to the fullest extent. The approach consists of grouping the words in the source language as a single syntactic entity and then handcrafting the lexical substitute needed for the target language. Certain rules are devised for the postposition. Word ordering is mostly preserved. A system between language pairs Kannada and Hindi has been developed.

ANGALABHARTI project was launched by Prof.R.M.K Sinha for machine translation from English to Indian languages in 1991. It is a machine aided translation system. A pattern directed rule-based system with context free grammar like structure is used to generate a 'pseudo-target'. The idea of the pseudo-target is to exploit structural similarity. A number of semantic tags are used to resolve sense ambiguity in the source language. A text generator module for each of the target language transforms the pseudo-target language to the target language. These transformations may lead to sentences, which may be ill formed. A corrector for ill-formed sentence is used for each of the target language. Finally a post-editing is done by a human being. This person needs to know only the target language. A system for translation from Hindi to Telgu has been developed.

Computer assisted teaching systems: Several software packages have been available for the teaching of Indian languages. The following software packages are available for teaching the grammar of Hindi and Sanskrit languages.

Hindi

- UPMA - Upma – Alankar package
- A-AB - Alankar – Arthabhed package

- WORD - Word formation package
- CASMOR – Cases generation and Morphological analysis package
- GNSA – Gender, Number, Synonyms, Antonyms package
- VARNA SHISHAK – Includes 4 modules for pronunciation, formatting,
exercise and evaluation
- SHABDA KOSH – for beginners and non-native Hindi learners

Sanskrit

General modules which include Varnamala, Sandhi, Subanta, Tignata, Kridanta, Taddhita and Samasa are available . Word identification Modules include Sandhi Vacheda, Subanta vachheda and Tiganta Vacheda. MEDHA package gives equivalent words in Sanskrit, Hindi, English, Kannada, Tamil, Telgu and Malayalam. PRATHIBHA is a machine translation software to translate Sanskrit sentences into Kannada and Tamil. DESIKA is a software developed for the analysis and generation of Sanskrit words.

Corpora Development: Corpora of texts in machine readable form has been developed for nearly 20 lakhs words in Tamil, Telgu, Kannada and Malayalam. 15 lakhs words in Gujarathi, Marathi, Oriya, Bengali and Sanskrit. Works pertaining to kashmiri, Urdu, Assammi, Punjabi and Hindi are progressing. Categories identified for manual as well as automatic tagging of the corpora at word level include noun, pronoun, verb (finite and non-finite) adjectives, adverbs and indeclinables.

Chapter 3

System Architecture

3.1 Introduction

Language comprehension requires human competence at various levels of knowledge. These levels include morphological analysis, syntactic analysis, semantic analysis, discourse integration and pragmatic analysis [51]. Due to ambiguity, imprecision and incompleteness, language comprehension is a complex task for computers. But surprisingly it is *achieved without much effort by human beings*.

Automated Natural language understanding systems have several potential applications. The standard technique used in NLU is to use a context-free grammar formalism for processing sentences. But this methodology is suitable only for fixed word order languages [52]. But Dravidian languages belong to the class of free word order languages. Hence a different approach is used in this work. It makes use of the 'karaka' relations in the sentence for its understanding. The validity of the approach is established by two case studies. The model developed is put to study in two important applications. One is machine translation and the other is development of natural language interface to databases. In the first case study reported, translation of simple and complex sentences in Dravidian languages to English is tried. In the second case study natural language queries in Dravidian languages are converted to SQL (structured query language) statements and are used for information retrieval from databases.

3.2 Dravidian Languages

The members of the family of Dravidian languages are the languages spoken by the people of South India. About 17 languages are there in this family. But about 95% of the south Indian population speak the prominent languages Telgu, Kannada, Tamil and Malayalam [53].

3.2.1 The Common Characteristics of Dravidian Languages

The scholars and grammarians of the ancient Sanskrit language held the view that all the South Indian languages were derived from Sanskrit and that there was no common criteria for claiming an identity for those languages. It seems they forget the fact that the individuality of a language is based on grammatical structures. It is true that some of these languages have borrowed words from Sanskrit. But in all basic details these languages maintain distinctness. If at all there were fragile contacts between Sanskrit and the Dravidian languages in the long past, it would be absurd to consider the Dravidian languages as an off-shoot of Sanskrit . According to Caldwell "The hypothesis of the existence of a remote original affinity between Dravidian languages and Sanskrit or rather between those languages and the Indo-European family of tongues inclusive of Sanskrit is of such a nature so as to allow us to give the Dravidian languages a place in the Indo-European group is altogether different from the notion of the direct derivation of those languages from Sanskrit" [53]. Some basic grammatical aspects are provided below which could be used to highlight the structural homogeneity or the cognate nature of Dravidian languages.

- Indo-European languages belong to the synthetic group and the Dravidian languages belong to the analytical group of languages. The Dravidian system of declensions is by means of suffixed postpositions and separable particles. For instance, the various case markers for the

noun in the singular are affixed to the singular base of the noun and plural affixes are added to the plural base of the noun. Whereas the languages belonging to the synthetic group make use of strong affixing structures capable of expressing different linguistic concepts.

- The condition that the object described should have accord with the adjectives with regard to gender, number etc. is absent in Dravidian languages. To be precise, the Dravidian languages do not possess correct adjectival words. This deficiency is compensated by participles. The rare modifiers which are exceptions to this also have the structure of participles.
- Another peculiarity of the Dravidian languages is the total absence of a device for forming degrees of adjectives. The availability of forms for comparison of adjectives is widely seen in all Indo-European languages.
- Another feature of the Dravidian languages with regard to verbs is the existence of affirmative as well as negative voice. The negatives have separate methods of expression. Eg. : Varum - Vara. In Sanskrit, the negative is indicated by a separate word. In Dravidian language, the adjectival and adverbial forms of two negatives, “alla” and “illa” are seen.
- The passive voice is absent in the Dravidian languages. A few languages like Malayalam have adopted passive voice in imitation of Sanskrit. On such occasions, they make use of another auxiliary verb to execute the structure.
- Two types of plural forms for expressing the first person is seen in most of the Dravidian languages. These plural forms either include or

exclude the listener, a method totally unknown to the Indo-European languages.

- Dravidian languages have only two forms, singular or plural. Dual is absent.
- The prefixes and suffixes used to indicate case relations are similar in both singular and plural.
- Plurals forms are very rare in neuter gender. In some languages, plural form is ruled out if the structure involves numerical adjectives.
- Common gender is another feature of the Dravidian languages.
E,g : Midukkan - Midukkanmar

- Midukkar

Midukki – Midukkikal

- Dravidian languages have distinctive features in the word order in sentences. Normal procedure is to join adjectives and adverbs before the noun or verb. The sequence is in the form subject, object and verb.
- The rules of sandhi in Sanskrit and in the Dravidian languages are quite different.
- Nouns are divided into two, humane and non-humane. The gender difference is shown only in the first case. Meaning rather than form is the basis of gender.

When we consider the forgone discussion, it becomes quite evident that the Dravidian languages belong to an independent family quite distinct from the Indo-European language families. Many linguists have attempted to link the Dravidian languages with the other language families like

Tibeto-Burman, Ural-Altai etc. But there is yet no unanimity of opinion among scholars about these propositions.

3.3 Design Methodology

The karaka based approach of sentence comprehension has its basis in the Paninian grammar. Panini, a great Indian scholar wrote a grammar system called Ashtadhyayi for Sanskrit somewhere around 500 BC. Since Sanskrit could be considered as a prototype for Indian languages to some extent, this approach could be also extended to Dravidian languages. Paninian approach addresses how to extract meaning from an utterance [54]. The karaka relations are made use for that. Karaka relations are the semantico-syntactic relations between the verbs and other related constituents in a sentence. They themselves do not impart any meaning, but tell how the nouns and other parts of speech are related to the verbs present in the sentence.

In this work the karaka relations are analysed for the understanding of Dravidian languages. It emphasizes the roles of vibakthi and postpositions markers . Position and word order is brought into consideration only when necessary. This approach is comparable with the broad class of case based grammars.

A sentence is not only a statement of an activity but also contains information regarding the speakers view point. The speaker's view point usually affects the choice of verb form and it in turn affects the choice of participants and their relation with the action. For example consider the sentences .

- 1. The boy opened the door.**
- 2. The wind opened the door.**
- 3. The door was opened.**

In the first sentence the speaker gives emphasis to the role of the boy. In the second sentence the emphasis is on the role of the wind and in the third sentence the emphasis is on the fact that the door was opened.

There are about six types of karaka relations [65]. It is clearly not possible for them to capture the innumerable types of semantic relations among all possible actions or states and all possible objects in the world. But they do specify the relations of nominals that participate in the action specified by the particular verb. They provide the maximum necessary information relative to a verb. According to Paninian perspective there are four levels in the understanding process of a sentence [52]. They are surface level (uttered sentence), vibakthi level, karaka level and semantic level. The karaka level has relationship to semantics on one side and to syntax on the other side. The position or order of occurrence of a noun group does not contain information about the karaka or theta roles in a sentence. The postposition markers after nouns in north Indian languages or surface case endings of nouns in Dravidian languages play a key role in specifying semantic relations. These markers and case endings are called vibakthi . Consider the following four sentences.

Raman Krishnane addichhu. (Raman beat krishnan.)

Krishnane Raman addichhu. (Raman beat krishnan.)

Ramane krishnan addichhu. (Krishnan beat Raman.)

Krishnan ramane addichhu. (Krishnan beat Raman.)

In sentences 1 & 2 Raman has the same semantic relation to the verb. In sentences 3 & 4 semantic relation of Raman is interchanged with that of Krishnan by interchanging their vibakthi. So vibakthi is crucial in determining the semantic roles.

The vibakthi forms could be seven . They are nirdesika, prathigrahika, samyojika, udeshika, prayojika, sambandhika and aadharika. For example, in table 3.1 the vibakthi forms of the Malayalam noun “kavi (poet)” are given.

Vibakthi	suffix	word
nirdesika	-	kavi
prathigrahika	e	kavie
samyojika	Ode	kaviyOde
udesika	ikke, ine	kavikke
prayojika	Al	kaviAl
sambandhika	inte, ude	kaviude
aadharika	il, kl	kaviyil

Table 3.1 Vibakthi forms of the noun "kavi(poet)"

The six karaka are karta, karma, sakshi, swami, hethu and adhikarna. Among the six the most independent one is called the karta karakam. The activity actually resides in or springs from karta. The corresponding noun is usually in the nirdesika vibakthi form. The result of the verb is reflected in karma karakam. For transitive or sakarmaka verbs karta and karma karaka will be different. In such cases the karma karakam will be in prathigrahika vibakthi if it is a lifeless entity. Otherwise it will be also in nirdesika form. Some activities require a participant with the karta. That participant is the sakshi karakam. For example in the sentence

Raman KrishnanOdu vazhakittu. (Raman quarreled with Krishnan.)

“Krishnan” is the sakshi karakam. Sakshi karakam will be in samyojika vibakthi form. Swami karakam is the beneficiary of the activity. For example consider the sentence.

Alice pattikke mamsom koduthu. (Alice gave meat to the dog.)

Here “patti (dog)” is the beneficiary of the activity gave. The swami karakam is usually in udesika / prathigrahika vibhakthi form . The hethu karakam is the instrument for performing the action. It will be in the prayojika vibakthi form. For example consider the sentence.

Amma kuttiye vadiyal adichhu. (Mother beat the child with a stick.)

Here “vadi (stick)” is the instrument for beating and it is in the Prayojika vibakthi form. Adikarana karakam refers to the object (in time or space) which helps in the execution of the activity. It will be in the aadharika vibakthi form. For example consider the sentence.

Pustakam mesail erripunddu. (The book is on the table.)

Here “mesa (table)” is the adhi-karna karakam and it is in aadharika vibhakthi.

However things are not always straight forward as given above. A different vibhakthi can be used for the same semantic relation with a given verb in a different sentence. For example consider the sentences

Raman kathakku thurannu. (Raman opened the door.)

Ramannu kathakku thurakkanam. (Raman wants to open the door.)

Ramanal kathakku thurakkappettu. (The door was opened by Raman.)

In the above sentences the noun Raman has the same semantic relationship with the verb open in all the cases . Raman is the karta karakam in all the three cases. But the vibakthi forms are different. This could be attributed to the verb forms. Several verb forms could be identified taking in to account

tense ,mood and compounding. For example for the verb padikkuka (learn), some commonly used verb forms are given in table 3.2

1.	padichhu	(learned)
2.	padikkunnu	(is learning)
3.	padikkum	(will learn)
4.	padikkanam	(has to learn)
5.	padikku	(learn)
6.	padichhukazhinju	(had learnt)
7.	padichhekkum	(may learn)
8.	padikkula	(will not learn)
9.	padikkuvan	(to learn)
10.	padichittu padichittu	(having learnt)
11.	padikkam	(shall learn)
12.	padichhappol	(when learnt)
13.	padikkukayanennu	(that learning)

Table 3.2. Verb forms of the verb "padikkuka"

It is not mandatory that all the six karakas should be there in a sentence for its understanding. Some karakas may be optional and some may be mandatory. In this work verbs belonging to the class of movement, perception, emotion, hurting, vocation, and transaction are taken for analysis. For each verb a karaka chart is stored. It contains the mandatory and optional karaka, the vibakthi forms for each karakam and the semantic properties. The karaka chart of some verbs in Malayalam are given in table 3.3.

Verb	Karaka	Presence	Vibakthi	Semantic properties
Chadukka	Kartha	mandatory	nirdeshika	animate
	Adikarna	optional	aadharika	Location in space or time
Parayukka	Kartha	mandatory	nirdesika	Human
	Karma	mandatory	nirdesika	piece of knowledge
	Adikarna	optional	aadharika	location in space or time
	Sakshi	mandatory	samyojika	Human
Uranguka	Kartha	mandatory	nirdesika	animate, having life
	Adikarna	optional	aadharika	Location in space or time
Kelkkuka	Kartha	mandatory	nirdesika	human
	Karma	mandatory	nirdesika	piece of knowledge
	Adikarna	optional	aadharika	location in space or time
Padippikkuka	Kartha	mandatory	nirdesika	Human
	Karma	mandatory	nirdesika	piece of knowledge
	Swami	mandatory	prathigrahika	human
	Adikarna	optional	aadharika	location in space or time

Fig 3.3 Karaka – vibakthi chart

The vibakthi forms are utilised for finding the various karaka. If more than one candidate happens to be in the same vibakthi form, then the semantic tags are used for finding the apt value. For example in the sentence

Teacher oru katha paranju. (teacher told a story.)

Here the verb paranju is in form 1 and from the above table this verb has two mandatory karakas. Both of them can be in nirdesika vibakthi form. Hence in the above sentence both “teacher” and “katha” are valid candidates for both karakas. This ambiguity is resolved by taking in to account the semantic properties to be satisfied by the members. Hence “teacher” becomes the kartha karakam and “katha” becomes karma karakam.

One word can have different senses. A verb or an adjective can have different meaning in different contexts. For example the verb “pidikkukka” in Malayalm has two senses. One is “to catch” and the other is “to like”. In both these cases the constituents of the sentences will be different. Hence word sense disambiguation is very important in sentence understanding.

3.4 Ambiguity Resolution

Lexical ambiguity could be classified in to three types. They are polysemy, homonymy and categorical ambiguity [55]. Polysemy refer to words whose several meanings are related to one another. For example the verb “open” may mean unfold, expand, reveal etc. Homonymy refers to words whose meanings are unrelated. For example the noun “bark” may mean covering of a tree or noise made by a dog. Categorical ambiguity refer to words whose syntactic category can vary. For example the word “sink” may be a noun or a verb. In the sentence “I fitted a steel sink in the kitchen”, sink is a noun whereas in the sentence “ Nails sink in water” sink is a verb.

In this work three information is used for word sense disambiguation. They are local word grouping (grouping of words those can collectively perform a syntactic role in a sentence) , syntactic information and semantic tags. Semantic tags are key words that denote the real world usage of a word. Roger Schank had suggested eleven primitives for actions and Y. Wilks had also advocated a similar concept called semantic formula. The

idea of using semantic tags to link a word and the object it denotes is closely related to the concept of reference in linguistics. Some of the semantic tags used in the work are animate, in animate, human, non-human, physical, abstract, speech, place, event, quantifiable etc. Even though this method of word sense ambiguity resolution is simple, all the rules for disambiguation must be manually identified and put in the lexicon. Considerable skill is required for the identification of semantic tags. But if the domain of the text is known a priori, this approach is computationally feasible.

3.5 System Architecture

The schematic diagram of the system is given in fig 3.1. The important modules are the morphological analyser, local word grouper and the parser.

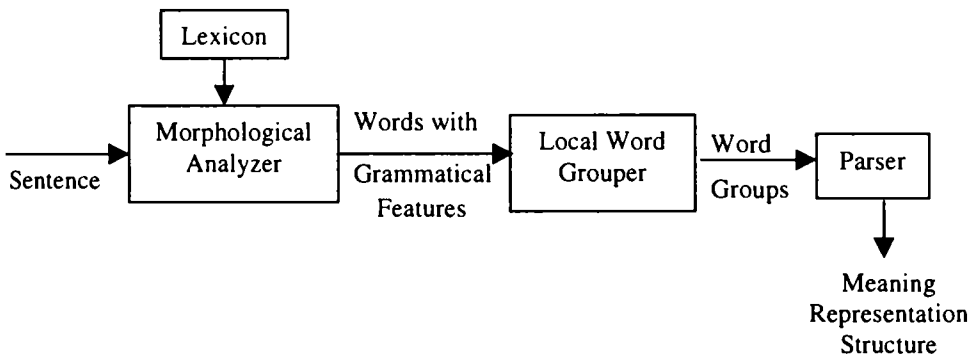


Fig 3.1. Schematic diagram of the system

3.5.1 Morphological Analyser

All natural languages have a large vocabulary of words. But all words are not root words. Most of them are derivatives of root words. For example in English the root word “come” has several derivatives. They are come, comes, coming and came. Hence storing all the derivatives of all words

will lead to a lexicon of very huge size. Hence in this work to keep the lexicon size optimum, root words along with the endings of derivative words are stored.

The complexity in the morphological structure of words could be attributed to the irregularity of languages [56]. Much of the irregularity comes from the fact that words come into a language from other languages and their internal structure reflects the morphological structure of the source language as well as that of the host. For example in Malayalam lots of Sanskrit and Tamil words could be seen.

Words could be analysed as consisting of a root and a prefix or suffix. In this work the prefix derivative words are considered as separate words since prefixes change the meaning of words. But suffixes only change the category information of a word.

The morphological analyser processes each word in the input sentence and gives the various syntactic features of the word. They include category, gender, number, person, case, tense etc . In Dravidian languages there are five parts of speech. They are noun, pronoun, verb, modifiers and dhyodhakam. Dhyodhakam includes suffixes, prefixes, postpositions etc. Since they are not independent words, they are not stored in a lexicon. Separate lexicons are kept for nouns, pronouns, verbs and modifiers. The format of the records stored in each lexicon is given below .

Noun {root, {suffix, vibakthi}, gender, number, person, semantic properties}

Pronoun {root, {suffix, vibakthi}, gender, number, person, semantic properties }

Verb {root, {suffix, verbform}}

Modifier {root, category}

In the lexicon for modifiers, category information is also stored. This is because modifiers could be of seven types. They are pure, demonstrative, qualitative, quantitative, numerical, participle, and adverbial. A sample of the dictionary entry is given below.

The noun “radha” is stored as

(radha,0:nirdeshika,e:prathigrahika,Odu:samyojika,ikkz:udeshika,Al:prayogika,il:aadharika,ude:sambandhika,female,singular,third, {human})

The pronoun “aval” is stored as

(aval,0:nirdeshika,ai:prathigrahika,odu:samyogika,kkuz:udeshika,Al:prayogika,il:aadharika,udei:sambandhika,female,singular,third, {human})

The verb padikkuka is stored as

(padi,chhu:1,kkunnu:2,kkum:3,kkanam:4,kku:5,chkukazhinju:6,chkhekkum:7,ikkula:8,kkuvan:9,chittu :10,kkam:11,chkhappol:12,kkuyanennu:13)

The lexicon of modifiers contains information like the following.

(ethu, demonstrative),

(venthingal : pure),

(nazhi, quantitative),

(randu, numerical),

(vegam, adverbial),

(padichha participle),

(nalla : quality).

3.5.2 Local Word Grouper

The function of this module is to form the word groups using the information based on adjacent words. The module takes into account the modifiers preceding nouns, pronouns and verbs. As given in the above section modifiers could be mainly of seven types. From the morphological

analyser category of each word could be obtained. If any modifier is present it will be grouped with the following noun/pronoun or verb. For example in the following sentences the word groups are underlined. The local word grouper, groups the words .

1. Ee roopa ennikku venda. (I don't want this money)
2. Venthingal udichhu (The moon had risen)
3. Orupara nelli thannu . (One litre paddy was given)
4. Nooru roopakku nallu pattiya vang. (Four dogs were bought for hundred rupees)
5. Oduuna vandikku brake venum. (Running car needs brake)
5. Vellutha pakshi parannupoi . (The white bird flew away)
6. Avan A paadam nannayi padichhu .(He learnt that lesson properly)

3.5.3 Parser

The function of the parser is to accept the local word groups and produce the parse structure. After parsing the meaning content of the sentence will get stored in a frame like structure as given fig 3.2.

The verb structure given in the fig 3.2 is only general. The parser has to identify the karaka relations among the word groups and to identify word senses. As given in the above section the karaka chart for each verb gives information about the mandatory and optional karaka of each verb. Expectation driven parsing is used for filling the slots. First the verb in the sentence is spotted. Since Dravidian languages are verb ending, parsing starts from right. Word sense ambiguity is resolved using semantic tags. The following constraints are observed while parsing.

1. For each of the mandatory karaka there should be exactly one word group satisfying the syntactic and semantic criterion.

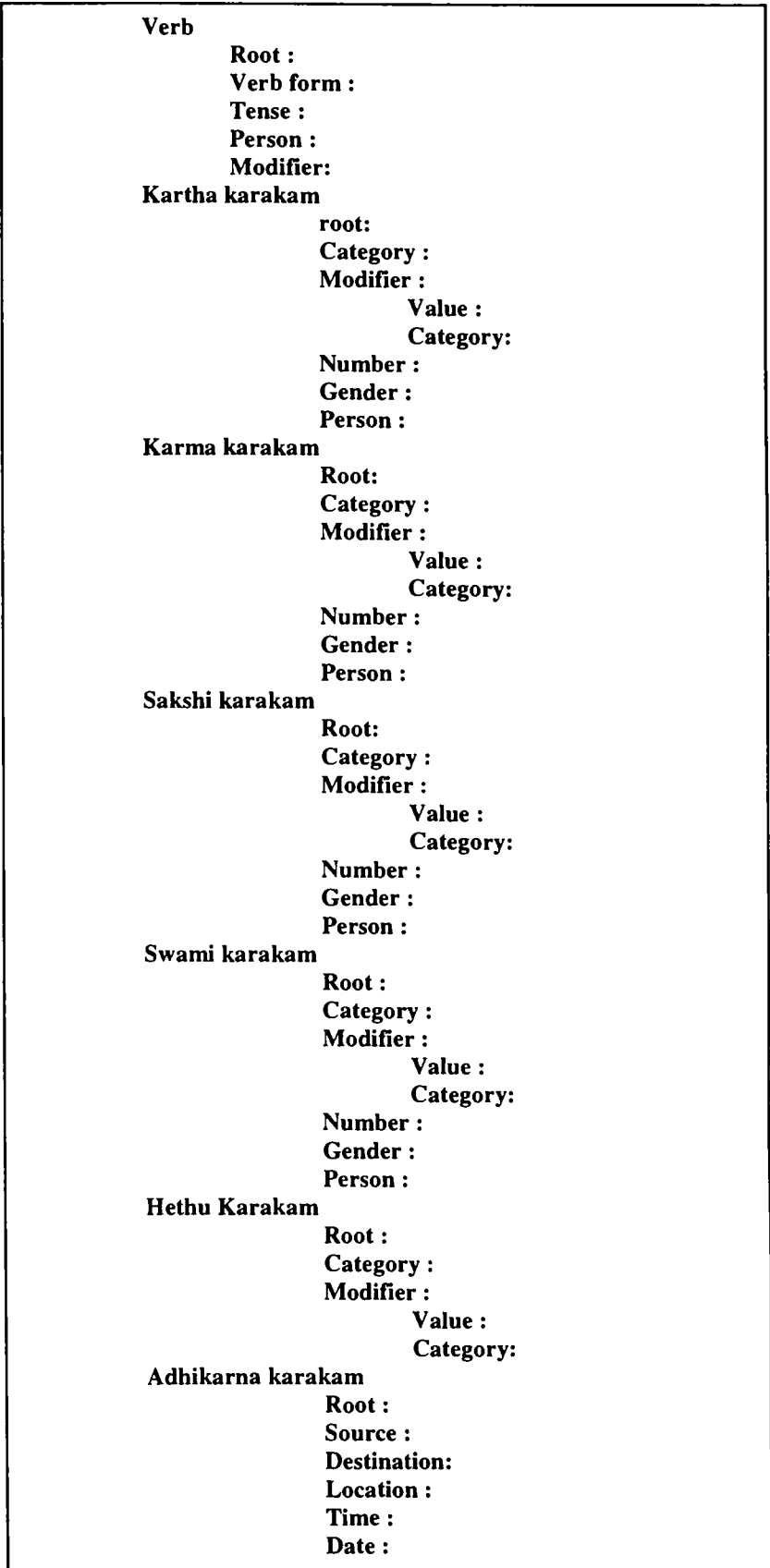


Fig. 3.2 General frame structure of verbs

2. For each of the optional karaka there should be at the most one word group satisfying the syntactic and semantic criterion.
3. There should be exactly one karaka relation for each word group.

The analysis of simple and complex sentences are carried out. Complex sentences contain one principal clause and one or more subordinate clause. In complex sentences, component phrases are demarcated by verbs. In this work complex sentences with one principal clause and one subordinate clause are considered. The analysis of several complex sentences showed that the subordinate clause verbs usually end with one of the following forms.

1. van (eg. parayuvan, kelkkuvan, chaduvan)
2. ittu (eg. paranjittu, kettittu, chadittu)
3. ppol (eg. paranjappol, kettappol, chadiappol)
4. kondu (eg. paranjukondu, kettokondu, chadikondu)
5. athu (eg. parajathu, kettathu)
6. ennu (eg. parayukayanennu, kelkukkayanennu)

Meaning representation structure corresponding each clause is developed. While filling the slots, if it is found that the mandatory karaka for subordinate clause or main clause is not available in the concerned phrase then they are to be obtained from adjacent phrases. That is karaka sharing is required. Studies enabled to come up with the following rules in filling the slots in the structure appropriately.

1. If the subordinate clause verb is having the ending “van” and if the sakshi karakam of the main verb is not given explicitly in the principal clause, then the kartha karakam of the subordinate clause is shared with the sakshi karakam of the principal clause. For example the Malayalam sentence.

krishnanodu markettilakku pokkuvan raman paranju.

To krishnan to market to go raman told
 (Raman told Krishnan to go to market.)

2. If the kartha karakam of the main verb is not given explicitly in the principal clause, then the kartha karakam of the subordinate clause is shared with the kartha karakam of the principal clause. For example the Malayalam sentences

kutti ammayude pattu kettittu urangi.
 Child mother's song having heard slept
 (The child slept having heard mother's song.)

raman manga thinnukondu veettilakku pooi.
 raman mango eating to home went
 (Raman went home eating mango.)

seetha ramane kandappol punchirichhu .
 seetha raman when saw smiled
 (Seetha smiled when she saw Raman.)

A set of simple and complex Malayalam sentences is given below. The parse tree and meaning representation structure of each of them are also given.

Simple Sentences

S.1 kutti avante pusthakam vayichu.

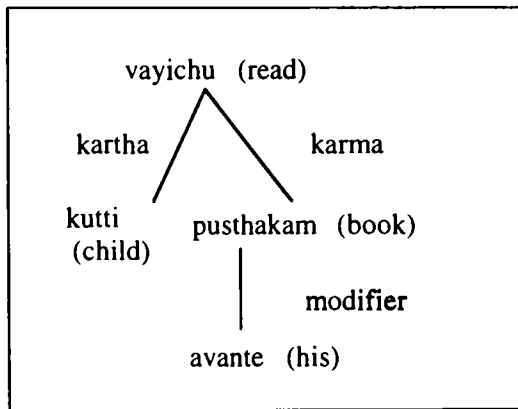


Fig. 3.3. Parse structure for sentence S.1

Verb	
Root	: vayi
Verb form	: form-1
Tense	: Past tense
Number	: Singular
Person	: 3 rd
Modifier	: NIL
Kartha Karakam	
Root	: Kutti
Category	: Noun
Modifier	: NIL
Number	: Singular
Gender	: Alinga
Person	: 3 rd
Karma karakam	
Root	: Pusthakam
Category	: Noun
Modifier	
Value	: avan
Category	: Sambandhika
Number	: Singular
Gender	: Neuter
Person	: 3 rd

Fig. 3.4. Frame structure for sentence S.1

S.2 Indiyala janangal narayanane pradhanamanthriyai thiranjeduthu.

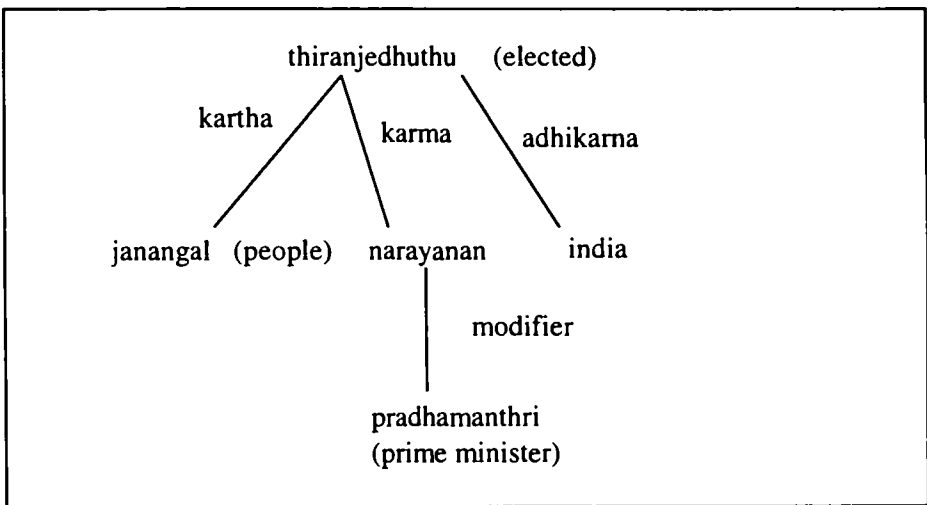


Fig. 3.5. Parse structure for sentence S.2

Verb	
Root	: thiranjad
Verb form	: form-1
Tense	: Past
Number	: Plural
Person	: 3 rd
Modifier	: NIL
Karthakarakam	
Root	: Janam
Category	: Noun
Modifier	: NIL
Number	: Plural
Gender	: Alinga
Person	: 3 rd
Karmakarakam	
Root	: narayanan
Category	: Noun
Modifier	
Value	: prathanamanthri
category	: complement
number	: singular
gender	: male
person	: 3 rd
Adhikarnakarakam	
Root	: India
Category	: noun
Modifier	: NIL
Number	: singular
Gender	: Neuter
Person	: 3 rd

Fig. 3.6. Frame structure for sentence S.2

S.3 Raman avante pusthakam nallavanaya syaminu koduthu.

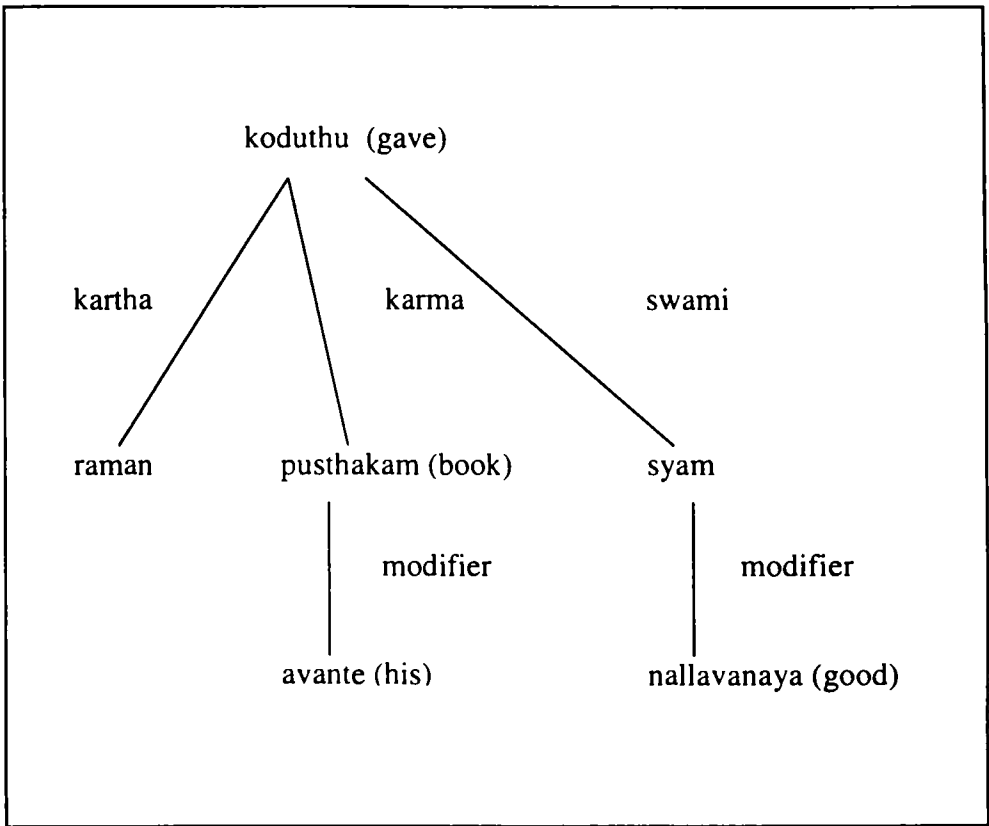


Fig. 3.7. Parse structure for sentence S.3

Verb	Root	: kodu
	Verb form	: form-1
	Tense	: past
	Number	: singular
	Person	: 3 rd
	Modifier	: NIL
Karna karakam	Root	: raman
	Category	: noun
	Modifier	: NIL
	Number	: singular
	Gender	: male
	Person	: 3 rd
Karma karakam	Root	: pusthakam
	Category	: noun
	Modifier	
	Value	: avan
	Category	: sambandhika
	Number	: singular
	Gender	: neuter
	Person	: 3 rd
Swami karakam	Root	: syam
	Category	: Noun
	Modifier	
	Value	: nallavan
	Category	: qualitative
	Number	: singular
	Gender	: male
	Person	: 3 rd

Fig. 3.8. Frame structure for sentence S.3

S.4 E kathakku vegam thurakkulla.

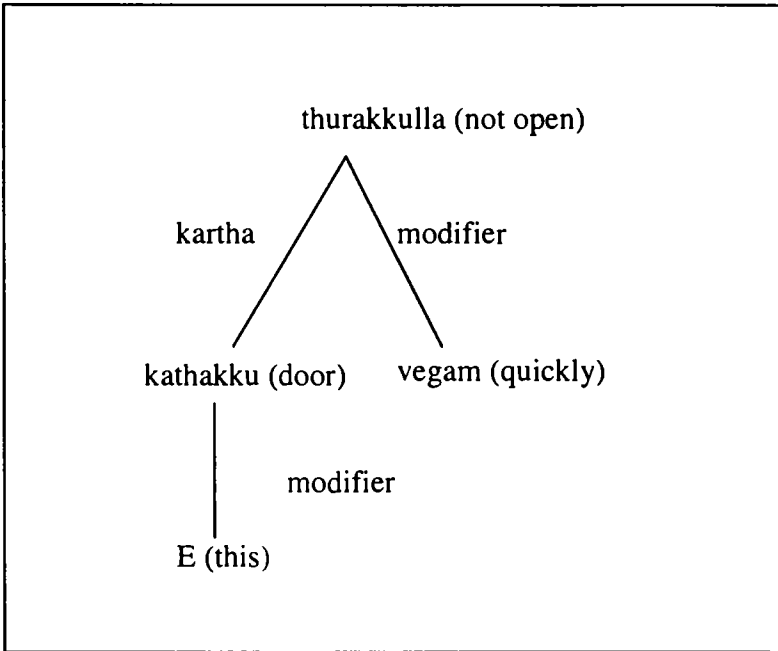


Fig.3.9. Parse structure for sentence S.4

Verb :	
Root	: thura
Verb form	: form-9
Tense	: future
Number	: singular
Person	: 3 rd
Modifier	
Value	: vegam
Category	: adverbial
Karthakarakam	
Root	: kathakku
Category	: noun
Modifier	
Value	: E
Category	: demonstrative
Number	: singular
Gender	: neuter
Person	: 3 rd

Fig. 3.10. Frame structure of sentence S.4

S.5 kurachhu ezhuthukal innu labichu.

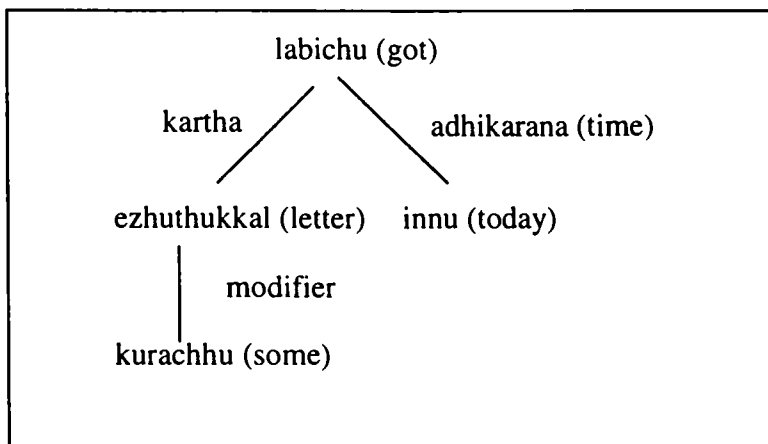


Fig. 3.11. Parse structure of sentence S.5

Verb	
Root	: labi
Verb form	: form-1
Tense	: past tense
Number	: plural
Person	: 3 rd
Modifier	: NIL
Karthā karakam	
Root	: ezhuthukal
Category	: noun
Modifier :	
Value	: kurachu
Category	: quantitative
Number	: plural
Gender	: neuter
Person	: 3 rd
Adhikāna kanakam	
Source	: NIL
Destination	: NIL
Location	: NIL
Time	: innu
Date	: NIL

Fig. 3.12. Frame structure of sentence S.5

S.6 nammal pavangalle sahayikkanam.

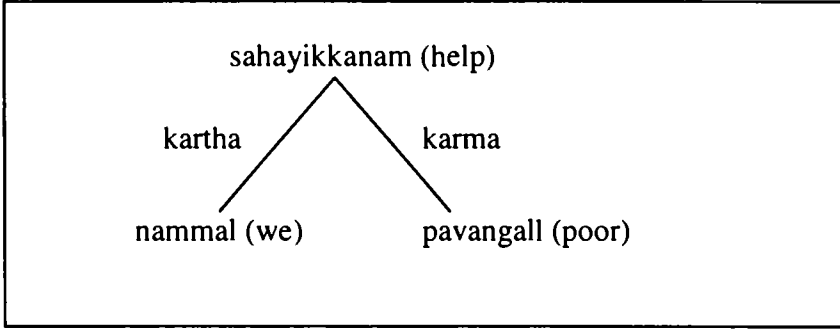


Fig. 3.13. Parse structure of sentence S.6

Verb	
Root	: sahayi
Verb form	: form-4
Tense	: future
Number	: plural
Person	: 1 st
Modifier	: NIL
Karthakarakam	
Root	: nammal
Category	: pronoun
Modifier	: NIL
Number	: plural
Gender	: alinga
Person	: 1 st
Karma karakam	
Root	: pavangall
Category	: noun
Modifier	: NIL
Number	: plural
Gender	: alinga
Person	: 3 rd

Fig. 3.14. Frame structure of sentence S.6

Complex sentences

C.1 Hariyude varthamanam kettittu Shyam chirichhu.

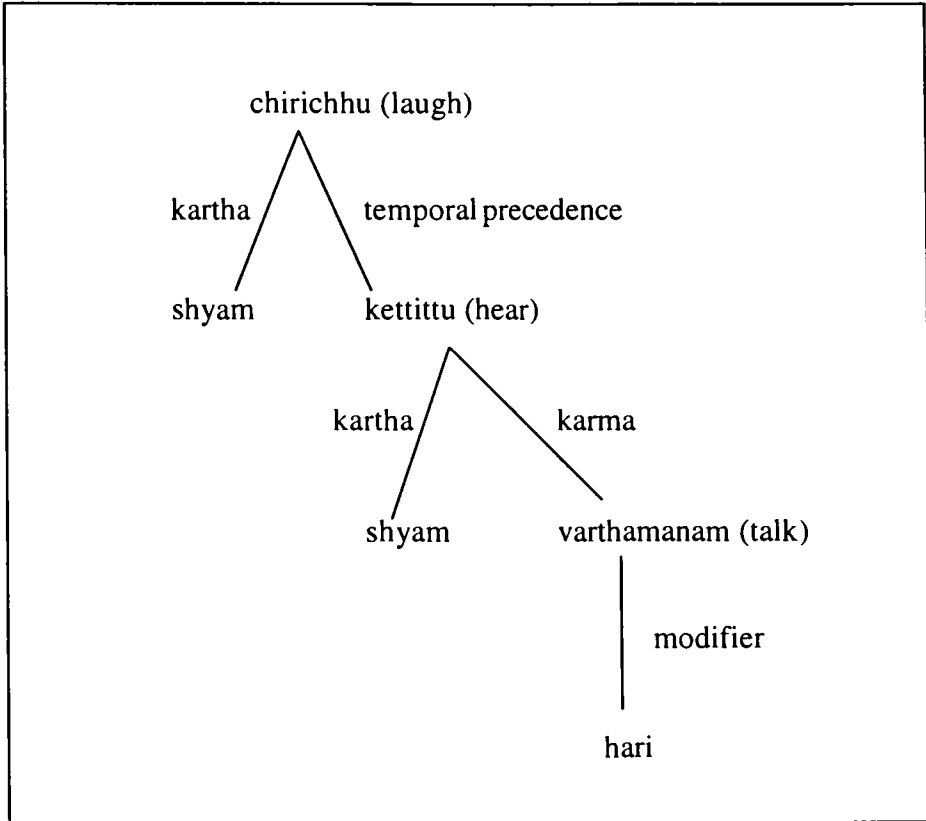


Fig. 3.15. Parse structure for complex sentence C.1

Main verb	
Root	: chiri
Verb form	: form-1
Tense	: past tense
Number	: singular
Person	: 3 rd
Modifier	: NIL
Karthā karakam	
Root	: Shyam
Category	: noun
Modifier	: NIL
Number	: singular
Gender	: male
Person	: 3 rd
Subordinate verb	
Root	: Ke
Verb form	: form-10
Tense	: past tense
Number	: singular
Person	: 3 rd
Modifier	: NIL
Karthā karakam / from main clause/	
Root	: Shyam
Category	: noun
Modifier	: NIL
Number	: singular
Gender	: male
Person	: 3 rd
Karma karakam	
Root	: varthamanam
Category	: verbal noun
Modifier	
Value	: Hari
Category	: sambandhika
Number	: singular
Gender	: neuter
Person	: 3 rd

Fig. 3.16 Frame structure of the complex sentence C.1

C.2 Teacher kuttiyodu pusthakam vayikkuvan paranju.

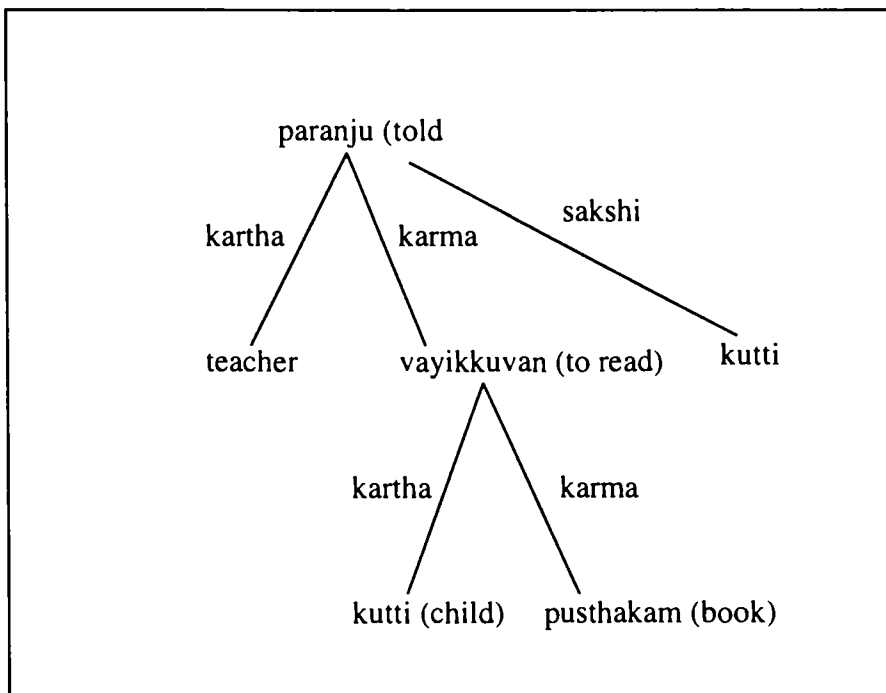


Fig. 3.17. Parse structure for complex sentence C.2

Main verb	
Root	: para
Verb form	: form-1
Tense	: past tense
Number	: singular
Person	: 3 rd
Modifier	: NIL
Kartha karakam	/ from sub-ordinate clause/
Root	: teacher
Category	: noun
Modifier	: NIL
Number	: singular
Gender	: alinga
Person	: 3 rd
Sakshi karakam	
Root	: kutti
Category	: noun
Modifier	: NIL
Number	: singular
Gender	: alinga
Person	: 3 rd
Subordinate verb	
Root	: vayi
Verb form	: form-9
Tense	: future
Number	: singular
Person	: 3 rd
Kartha karakam	
Root	: kutti
Category	: noun
Modifier	: NIL
Number	: singular
Gender	: alinga
Person	: 3 rd
Karma karakam	
Root	: pusthakam
Category	: noun
Modifier	: NIL
Number	: singular
Gender	: neuter
Person	: 3 rd

Fig. 3.18. Frame structure of the complex sentence C.2

C.3 Raman kathaku thurannappol poocha chadi.

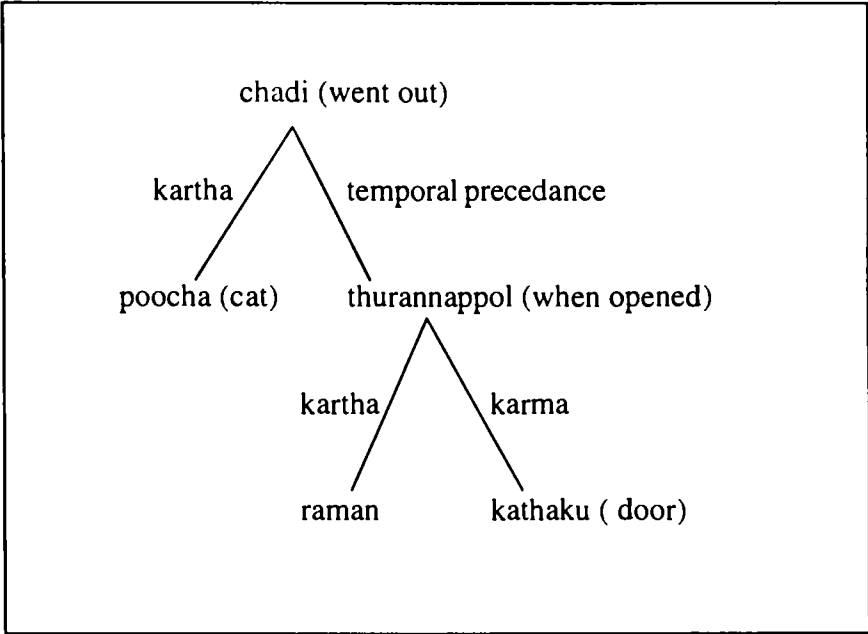


Fig. 3.19. Parse structure for complex sentence C.3

Main clause : verb	
Root	: chadi
Verb form	: form-1
Tense	: past tense
Number	: singular
Person	: 3 rd
Modifier	: NIL
Kartha karakam	
Root	: poocha
Category	: noun
Modifier	: NIL
Number	: singular
Gender	: neuter
Person	: 3 rd
Subordinate clause	
Verb root	: thura
Verb form	: form-12
Tense	: past tense
Number	: singular
Person	: 3 rd
Modifier	: NIL
Kartha karakam	
Root	: raman
Category	: noun
Modifier	: NIL
Gender	: Male
Person	: 3 rd
Karma karakam	
Root	: kathakku
Category	: noun
Modifier	: NIL
Number	: singular
Gender	: neuter
Person	: 3 rd

Fig. 3.20. Parse structure for complex sentence C.3

C.5 kutti pattu padikkukayanennu amma paranju.

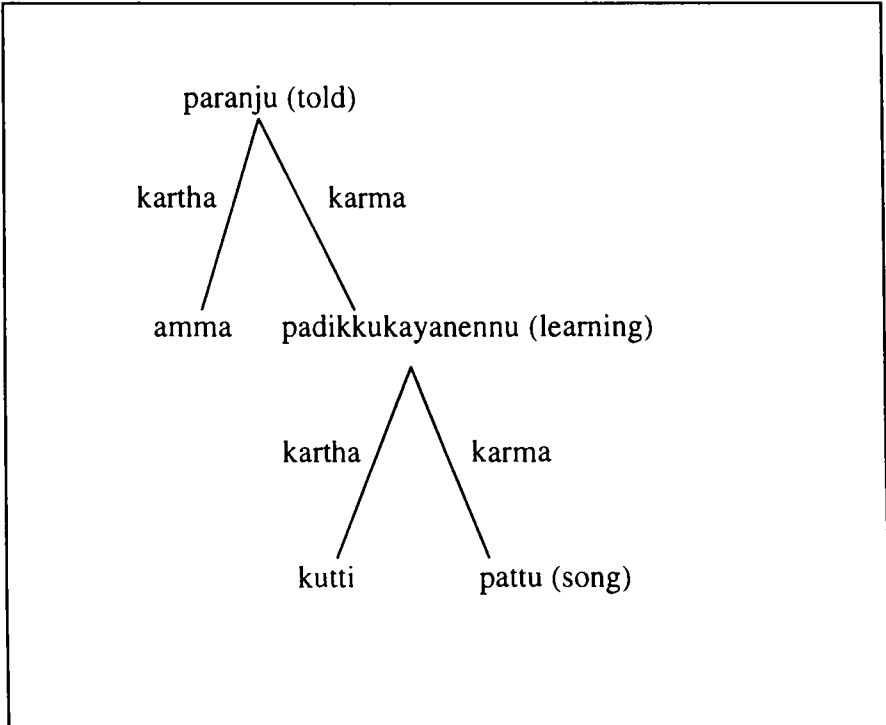


Fig. 3.23. Parse structure of complex sentence C.5

Main verb		
Root	:	para
Verb form	:	form-1
Tense	:	past tense
Number	:	singular
Person	:	3 rd
Kartha karakam		
Root	:	amma
Category	:	noun
Modifier	:	NIL
Number	:	singular
Person	:	3 rd
Gender	:	female
Subordinate verb		
Verb	:	para
Verb form	:	form-13
Tense	:	present tense
Number	:	singular
Person	:	3 rd
Modifier	:	NIL
Kartha karakam		
Root	:	kutti
Category	:	noun
Modifier	:	NIL
Gender	:	neuter
Person	:	3 rd
Karma karakam		
Root	:	pattu
Category	:	noun
Modifier	:	NIL
Number	:	singular
Gender	:	neuter
Person	:	3 rd

Fig. 3.24. Frame structure of sentence C.5

C.5 kutti pattu padikkukayanennu amma paranju.

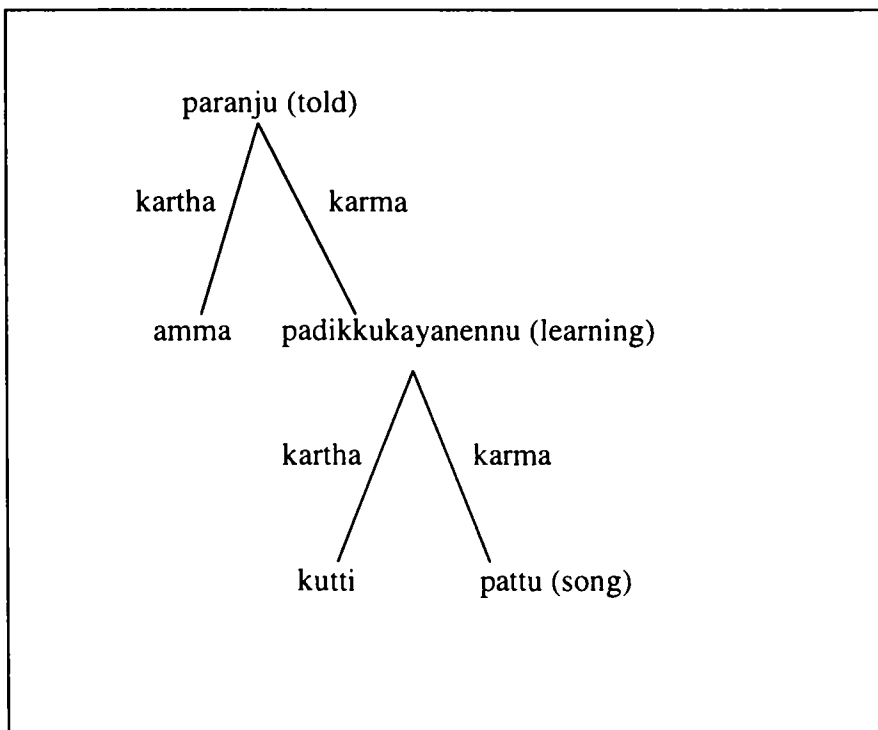


Fig. 3.23. Parse structure of complex sentence C.5

Main verb

Root : para
Verb form : form-1
Tense : past tense
Number : singular
Person : 3rd

Kartha karakam

Root : amma
Category : noun
Modifier : NIL
Number : singular
Person : 3rd
Gender : female

Subordinate verb

Verb : para
Verb form : form-13
Tense : present tense
Number : singular
Person : 3rd
Modifier : NIL

Kartha karakam

Root : kutti
Category : noun
Modifier : NIL
Gender : neuter
Person : 3rd

Karma karakam

Root : pattu
Category : noun
Modifier : NIL
Number : singular
Gender : neuter
Person : 3rd

Fig. 3.24. Frame structure of sentence C.5

3.6 Casestudy – 1 (Machine Translation)

The capability of the meaning representation frame was tested by using it in translation of sentences given in Dravidian languages to English. Here the source language is a free word order and the target language is fixed word order. The frame in which the meaning is stored is given to an English language generator. Hence in the lexicon the English equivalents of the words are also stored. In this work the sentences are considered as isolated. The schematic diagram of the machine translation system is given in Fig 3.25.

English is a rich language with enormous variety of patterns or sentence structures [65]. Precisely, a universal set of patterns is yet to be identified. It is really a tough task to develop a system capable of handling all these structures. Hence the domain of analysis is restricted to 15 patterns. The patterns considered are given below. Sample sentences belonging to each category and the corresponding Malayalam equivalents are also given.

1. Subject + verb

Birds are flying

Pakshikkal parakkunnu

2. Subject + verb + direct object

Mohan opened the door

Mohan kathakku thurannu

3. Subject + verb + indirect object + direct object

I gave her my pen

Njan entta pena avalkku koduthu

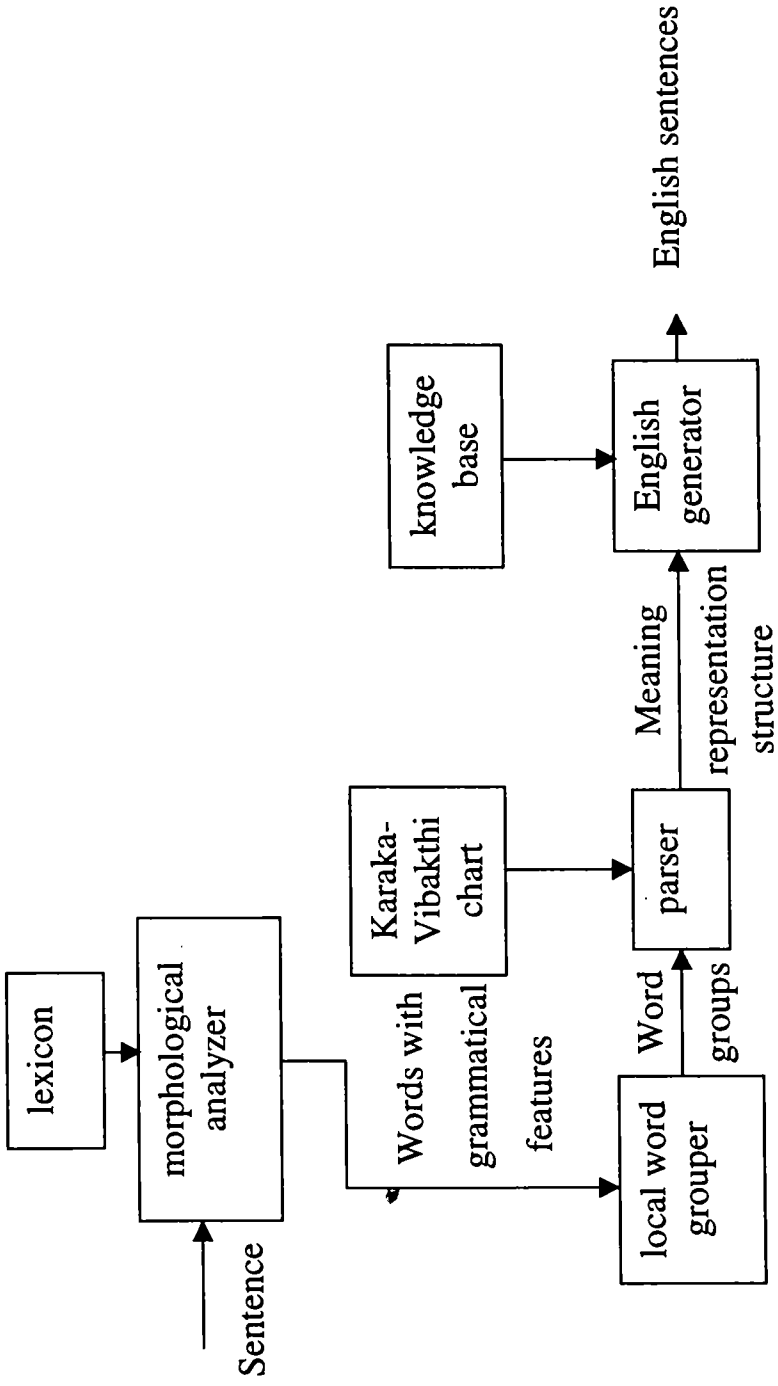


Fig 3.25 Schematic diagram of Machine Translation System

4. Subject + verb + object + object complement
We found the trunk empty.
Njangal petti kaliyayi kkandu
5. Subject + verb + preposition + prepositional object
He failed in examination
Avan pareeshakku thottu
6. Subject + verb + to-infinitive
She desired to go there
Aval avedi pokkuvan aaghrachhu
7. Subject + verb + object + to-infinitive
She ordered Gopi to stay in bed
Kattillil aayirikkuvan aval gopiyodu aavashyapettu
8. Subject + verb + object + present participle + clause
I found him playing cards
Avan cheettu kalikkunnathu njan kandu
9. Subject + verb + perfect participle + clause
Syam laughed having heard Hari's talk
Hariyudai varthamanam kettittu syam chirichhu
10. Subject + verb + object + perfect participle + clause
Raman drew a figure having taken sudha's pencil
Raman sudhayudai pencil eduthittu padam varachhu
11. Subject + verb + object + past participle + clause
I saw raman taken my bag
Raman entta bag editthatthu njan kandu

12. Subject + verb + that + clause

He admitted that he had written a letter

Oru ezhuthu ezhuthiyennu avan samathichhu.

13. Subject + verb + object + that + clause

He told me that he is coming on Sunday

Avan thingalazhchha varukayanennu ennodu paranju

14. Subject + verb + when – clause

The bird flew away when the door was opened

Katakku thurannappol pakshi parannupoi

15. Subject + verb + object + when – clause

Father bought a shirt when he came yesterday

Acchhan ennalai vannappol oru shirt konduvannu

These patterns form the skeleton of the sentences to be generated. The sentences considered would get mapped to the most appropriate pattern. A set of rules were used for that. It is given below.

- The kartha karakam gets mapped to the subject.
- If the main verb is transitive then its karma karakam will be the direct object.
- If the main verb is transitive then the swami karakam of it will get mapped to the indirect object.
- If the verb has a sakshi karakam then it becomes the object of the sentence.
- The hethu karakam and ahikarna karakam are placed as prepositional clauses towards the end of the target sentence. Since the hethu karakam refers to the instrument case, the prepositions used in the target sentence are by or with. Since the ahikarna karakam refers to the position in time or space, the prepositions used in the target sentence are at, on, over, behind and in.

- If the modifier of the karma karakam belongs to the category of complement then the complement will appear after the object as in pattern 4.
- Certain verbs like wait, agree, count, belong etc attaches a preposition before the object. Such sentences get mapped to pattern 5.
- If the subordinate clause verb's ending is "van' and if the main verb has a sakshi or swami karakam the sentence get mapped to pattern 7 else to pattern 6.
- If the subordinate clause is verb's ending is form "athu ' and if the main verb has a sakshi or swami karakam the sentence get mapped to pattern 8.
- If the subordinate clause verb's ending is "ittu ' and if the main verb has a sakshi or swami karakam sentence get mapped to pattern 10 else to pattern 9.
- If the subordinate clause verb's ending is "thu' and if the main verb has a sakshi or swami karakam the sentence get mapped to pattern 11.
- If the subordinate clause verb's ending is "ennu' and if the main verb has a sakshi or swami karakam the sentence get mapped to pattern 13 else to pattern 12.
- If the subordinate clause verb's ending is "ppol' and if the main verb has a sakshi or swami karakam the sentence get mapped to pattern 15 else to pattern 14.

For translation, the meaning representation structure for each sentence is generated. Then using the rule base given above the corresponding English language equivalent is obtained. The English translation of a Malayalam story obtained by this system is given in chapter 5. It contains only sentences that could be mapped in to one of the 15 structures given above.

Dravidian languages are less ambiguous than English. This is due to the fact that these languages are highly case-inflected [57]. These inflections

disambiguate the semantic content of the sentences. For example, the following sentences in English are syntactically and semantically ambiguous.

I saw a man on the hill with the telescope.

When this sentence is translated and written in Malayalam, the sequence of words and particularly their inflections disambiguate the semantics as illustrated below.

Njan telescoppinal malayudai mukallil oru manushanai kanddu.

Njan telescoppulla manushanai malayudai mukallil kandu.

Njan telescoppulla malayudai mukkallil oru manushanai kandu.

These sentences respectively indicate that the telescope was used to spot a man, or that the man had a telescope, or that the telescope was installed on a hill. The low ambiguity in sentence meaning facilitate translation from a Dravidian language to English comparatively easier than the other way.

3.7 Casestudy –2 (A Natural Language Interface for RDBMS)

In the second application an NLI for information retrieval from databases is tried. As it is known, many of the shortcomings of the database languages could be overcome by putting an interface between the user's native language and the database language. This method has several advantages.

1. The interface can eliminate the necessity for the user to conform to an artificial syntax.
2. It relieves the user from knowing about the details of the database model, data definition languages and data manipulation languages.
3. The interface enables to understand incomplete or slightly erroneous queries, elliptical requests, anaphoric references etc.
4. It can also recognize the logical inconsistencies in a query and warn the user.

Since nowadays relational database management systems are de facto standards and SQL or SQL like languages are commonly used, the internal meaning representation is mapped to SQL commands. In this application the design is oriented towards solving the following important issues.

1. If the NLI uses only a very small subset of the language, then it doesn't have good practical significance. Because the user may find to his irritation that he has to remember which particular way of saying things is acceptable.
2. If the database is not large and not used for reasonably complex information retrieval then it is not worthwhile constructing the NLI. Because the nature of this work is manpower intensive.
3. In addition to straightforward queries, the system should be capable of handling complex queries, which include multi-relational queries and those containing conjunctives and quantifiers.
4. The NLI should give quality responses and should be capable of handling user misconceptions.
5. The NLI should handle anaphoric and elliptical requests.

The NLI prototype answers written Malayalam questions. These questions are translated to SQL commands and they are executed by the DBMS. First the meaning of the query is extracted and stored in a frame. The karaka relations are used for it. Here the source language is a free word order language and the destination is a formal language having a definite format. The schematic diagram of the NLI system is given in fig 3.26.

Any SQL statement will have a SELECT clause, FROM clause and a WHERE clause. The SELECT clause has the field names to be projected. The WHERE clause contains the conditions to be satisfied for selecting records from tables. The FROM clause contains the tables to be joined.

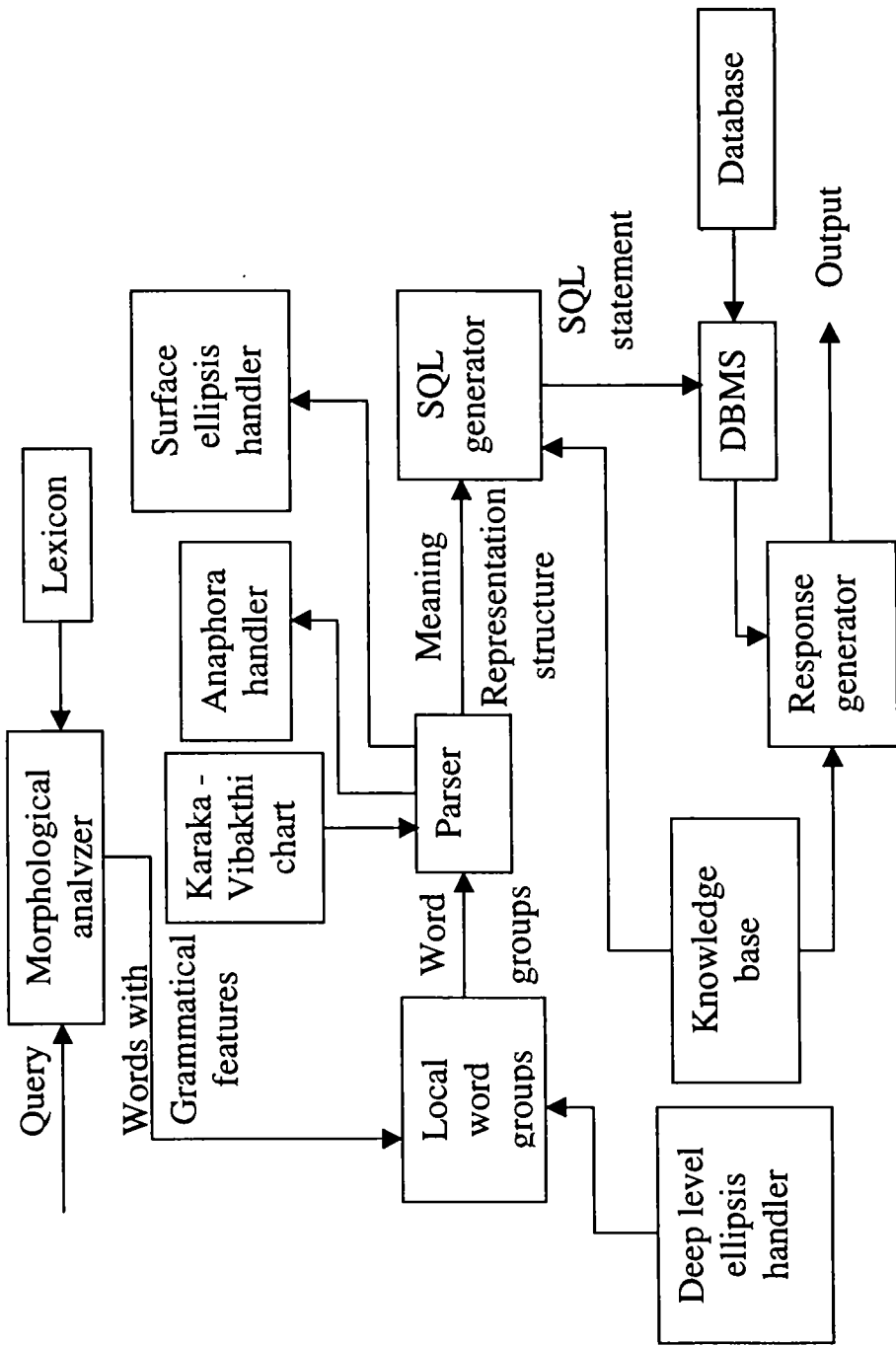


Fig 3.26 Schematic diagram of the NLI system

Domain knowledge is essential for an NLI to achieve an acceptable level of performance [58]. The domain knowledge is stored in the Entity-Relationship (E-R) diagram. The E-R diagram consists of entities, attributes and relationship between entities. An election information system, an academic information system and a library information system are considered for study. Their E-R diagrams are shown in Appendix 1, 2 & 3. The entities are represented by rectangles, relationship between entities by diamonds and attributes of entities by rectangles with rounded corners.

The words appearing in the query are mapped to entities, attribute names, values and relations. The proper nouns can be mapped to values, common nouns to attribute names and entities and verbs to relationships between entities. Taking into account the application, the above appropriate values are also stored along with each category of words. Separate lexicons are kept for proper nouns, common nouns, pronouns, verbs, relational words , query words and quantifiers. The formats of the records stored in the various lexicons are as given below.

Proper noun

(root, {suffix, vibakthiforms}, gender, noun, person, databasefields)

Common noun (root, {suffix, vibakthi forms}, gender, noun, person, database field names / database tables)

Pronouns (root, {suffix, vibakthiforms}, gender, number, person)

Verb (root, tables involved)

Query word (root, database field names)

Relational words (root, operator)

Quantifiers (root)

3.7.1 Meaning Extraction

Investigations have shown that in the languages considered, conditional clauses come towards the beginning of the query and the values to be projected or outputted come next. Certain patterns are noted in the format of the conditional clauses. They are

1. <proper noun><attribute>
2. <numeric> <attribute><relation>
3. <numeric> <relation><attribute>
4. <numeric> <attribute>
5. <quantifier><relation.<attribute>

Certain patterns are also observed in the format of phrases containing the values to be outputted. They are

1. <attribute><query word>
2. <entity> <query word>
3. <entity's> <attribute> <query word>
4. <query word> <entity> <verb>
5. <verb> <entity's> <attribute> <query word>
6. <query word><verb>
7. <verb><entity><query word>

The conditional clauses form the karma karaka of the representation. The values to be outputted form the kartha karakam. It is also possible to write a query without a verb in Malayalam. In such cases the verb is considered as “be” verb. For example the query “Indiyudai pradhanamanthri aare ? . (who is the prime minister of India) “ doesn’t contain a verb.

The meaning content of the query is stored in a structure as given below.

Verb**Root:****Kartha karakam****Query word :****Attribute :****Entity :****Karma karakam****Value :****Attribute :****Relation :****Quantifier :**

Since there can be many number of conditional clauses in a query, the karma karakam is kept as a list. The first step in the processing of the query is morphological analysis. Then word grouping according to the patterns identified above is done. Then the parser will fill the slots of the meaning representation structure appropriately.

The above process can be explained by considering the query S1 given against the election database.

S1. ernakulam mandalathil ethra sthanarthikal malsaricchu ? (How many candidates contested from Ernakulam constituency ?)

After morphological analysis, we get the word categories as (ernakulam – proper noun), (mandalathil, attribute),(ethra, query word), (sthanarthikkal, entity) and (malsaricchu, verb). The local word grouper then finds that there is one conditional clause according to pattern 1 and one output clause according to pattern 4. The parser then builds the following structure.

Verb : malsarichhu**Kartha karakam****Query word : ethra****Attribute :****Entity : sthanarthikkal**

Karma karakam

Value : ernakulam

Attribute : mandalam

Relation: =

Quantifier :

3.7.2 SQL Generator

This module generates the SQL statement corresponding to the information content stored in the meaning representation structure. The algorithm used for it is given below.

1. From the kartha karakam get the database fields to be included in the **SELECT** clause and the tables to be included in the **FROM** clause.
2. From the karma karakam list get the conditions to be included in the **WHERE** clause and the tables to be included in the **FROM** clause.
3. Get the complete list of tables to be included in the **FROM** clause and find the join conditions, if any. Include these join conditions in the **WHERE** clause.
4. Write the SQL command.

The election database has got five tables. They are candidate, constituency, party, win and contest. Certain tables could be joined directly. But certain other tables require a mediator to get joined. This factor is judged by the E-R diagram of the database. The E-R diagram of the election database is given in appendix 1. From the figure it could see that if a join operation is required between the tables win and party, it could be done via the table candidate. Hence a list is maintained which helps in the joining of various tables involved. A sample is given below.

(contest, cand) ::- contest.candcode = cand.candcode

(contest, cons) :: -contest.conscore = cons.conscore

(win, cand) ::- win.candcode = cand.candcode

(win, cons) :: -win.conscodcode = cons.conscodcode
(party, cand) ::- (party.partycode = cand.partycode).

Thus the SQL command for the query **S1** is

Select COUNT (candidate.candname)
From cons, contest, cand
Where cand.candcode = contest.candcode AND
contest.conscodcode= cons.conscodcode AND
cons.consname= "ernakulam".

Consider another query.

S2. Ettavum kuduthal vote labichha sthanarthi aare? (ഏറ്റവും കൂടുതൽ വോട്ട് ലഭിച്ച സ്ഥാനാർഥി ആര് ?)
(Candidate with the highest vote?).

The frame containing the meaning of the query is

Verb : labichha
Karthā karakam
Query word : aare
Attribute :
Entity : sthanarthi
Karma karakam
Value :
Attribute :vote
Relation
quantifier : ettavum

The analysis of the above representation shows that two tables are involved. (candidate and contest) . Hence the SQL command is

```

Select candidate.candname
From Candidate, contest
Where Candidate.candcode = contest.candcode
      AND
      Contest.vote = (Select MAX(contest.vote)
                    From contest.)

```

3.7.3 Ellipsis & Anaphoric References

To support natural interaction, it is desirable to allow the use of anaphoric reference and elliptical constructions across sentence sequences. In a general context, references and ellipsis are hard problems. But in this case the restricted domain of discourse that is defined by the database makes it possible to address these problems.[58]

3.7.3.1 Ellipsis

A question is called elliptic if one or more of its constituents are omitted. For brevity in communication, it is important for an NLI to handle ellipsis. There are two types of ellipsis.

- Surface level ellipsis. This is detected and handled by the parser at the syntactic level.
- Deep level ellipsis. This is detected and handled during morphological analysis. A rule base has been kept for it .

3.7.3.1.1 Surface Level Ellipsis.

The parse structure information is sufficient to handle this type of ellipsis. An elliptical request is processed by making use of contextual knowledge. For processing an elliptical request, first we have to recognize that the query is elliptical. A query is elliptical at the surface level if the kartha

karakam is missing. An assumption is made such that in the parse structure the elliptical fragment corresponds to that of the previous query and there is no syntax violation in the question. The parser is modified so that instead of failing for the above query, it invokes the ellipsis handler and generate the complete meaning structure. For eg if the query sequence is *kottayam districtillai congerssintta sthanarthikkal aarelam? Marxistinttayo ?*

(കോട്ടയം ജില്ലയിൽ കോൺഗ്രസിന്റെ സ്ഥാനാർത്ഥികൾ ആരെല്ലാം ? മാർസിസ്റ്റിന്റെയോ ?)

(who are the congress candidates contested from kottayam ? marxist?)

The first query is a valid query and after parsing , the structure returned is

Verb : be

Kartha karakam

Query word : **aarellam**

Attribute :

Entity : **sthanarthikal**

Karma karakam

Value : **Kottayam**

Attribute : **Jilla**

Relation: =

Quantifier :

Value : **Congress**

Attribute : **party**

Relation : =

quantifier:

The second query is processed as follows. The parse structure for the second query is

Verb :

Kartha karakam

Query word :

Attribute :

Entity :

Karma karakam

Value : **marxist**

Attribute : **party**

Relation : =
quantifier

Since the kartha karakam is missing it is considered as an elliptical request. The elliptical handler super imposes this structure over the structure of the previous query. Thus the complete structure of the second query becomes

Verb: **be**

Kartha karakam

Query word : **aarellam**

Attribute :

Entity : **Sthanthikal**

Karma karakam

Value : **Kottayam**

Attribute : **Jilla**

Relation/quantifier : =

Value : **Marxist**

Attribute : **party**

Relation : =

This structure is equivalent to that of the query *kottayam districtil marxist partyudai sthanarthikkal aarellam?*. Then the corresponding SQL command is generated.

3.7.3.2 Deep Level Ellipses

The user input is recognized as elliptical at deep level, if all the constituents needed for word grouping are not present in the input. The methods for resolution of deep level ellipsis are

- Use of domain knowledge
- User interaction

In this work both the methods are used. A knowledge base has been kept for deep level ellipsis resolution. The system will rephrase the query containing deep level ellipsis. Some examples of queries with deep level ellipses are discussed in the table 3.4

Original query with deep level ellipsis	System rephrased query
Aluvaude MLA aare ? ആലുവയുടെ എമെമല്ലെ ആര്	Aluva mandalathil ettavum kuduthal vote labhichha sthanarthi aare? ആലുവ മണ്ടലത്തിൽ ഏറ്റവും കൂടുതൽ വോട്ടു ലഭിച്ചതാർക്ക്?
Kollamkkar aarellam കൊല്ലംകാർ ആരെല്ലാം	Kollam mandalathila sthanarthikkal aarellam കൊല്ലം മണ്ടലത്തിലെ സ്ഥാനാ- ർഥികൾ ആരെല്ലാം?
Bjpude sthanarthikkal aarellam ബിജെപിയുടെ സ്ഥാനാർഥി- കൾ ആരെല്ലാം?	BJP partyudai sthanarthikkal aarellam ബിജെപി പാർട്ടിയുടെ സ്ഥാനാ- ർഥികൾ ആരെല്ലാം?

Table 3.4. Queries with deep level ellipses

3.7.3.3 Anaphoric References

The term “anaphora” refers to reflexive pronouns, general pronouns, definite noun phrases etc. Anaphora resolution involves finding referents

of these in a discourse which may consist of more than one sentence. The detection of anaphora is done by the parser. The referent is found out from the candidates using the following methods.

- Agreement of the referential pronoun with the candidates in the previous query in gender and number.
- Using the domain knowledge supplied by the E-R diagram.

In the first method all the candidates those are not in agreement in number and gender of the anaphor is filtered out. For example consider the following two query sequences.

1. Computergraphicsenna pusthakam ezhuthiyathu aare ?

Athinte vila enthu ?

2. Computergraphicsenna pusthakam ezhuthiyathu aare ?

Ayalude address enthu ?

In the first query the anaphor “athinte” refers to the book and in the second case the anaphor “ayalude” refers to the author. The first anaphor agrees with number and gender of the candidate “book”, while the second anaphor agrees with those of candidate “author”.

The second method is used if the first one fails in filtering out a candidate. It makes use of the domain knowledge. For example in the query

Computergraphicsenna pusthakathinnta vila enthu?

Athu aare ezhuthi ?

Here the anaphor “athu” could refer both to book and cost. These two candidates agree in number and gender with the anaphor. Hence the first method fails. But the verb “ezhuthi “ refers to the relation “publish” which is between the book and the publisher. Hence the candidate selected is book. This information is obtained from the domain Knowledge.

3.7.4 Processing of Null Responses from Databases

The SQL statements given to the underlying DBMS may some times produce null responses. Database systems rarely contain all of the information necessary to model their domain. Hence null values arise in many database access. Natural language front-ends are designed to accommodate naive users. The more informal the query language is, the more sophisticated the system needs to be in order to comprehend and answer queries properly. User misconception is an important cause of null responses. User misconception [59,60] can be classified into misconceptions that fail extensionally and misconceptions that fail intentionally. Intentional failure arises when the user has a misconception about some domain relationships and about entities that can participate in some relations. Extensional failure can be due to the non existence of certain object or due to the nonexistence of a tuple or due to the fact that the event which is responsible for the desired value has not been taken place as yet.

For generating quality responses during the occurrence of null values, a set of relations are stored in the knowledge base. This is in addition to the relations in the database schema. The newly added relations are E- relation (Event relation), EG-relation (Event- Graph relation), EXC-relation (Exception relation) and V-relation (View relation). These relations could be explained with a database which contains information about the academic matters of a university. The database schema is given below.

Student (Stud-id, name, address, birth-year, dept-id)

Courses (Course-id, name, text)

Department (dept-id, name, head, estd-year)

Teacher (teach-id, name, address, designation, dept-id)

Quarter (q-year, q-season)

Enrolls (stud-id, q-year, q-season, course-id, mark)

Offers (teach-id, q-year, q-season, course-id)

3.7.4.1 E- relation (Event)

It is a binary relation. It gives the exact or approximate date for the occurrence of the event. For example, in the E-relation table 3.5 of the event final exams, dates of commencement of exam of various semesters will be stored.

E – Relation	
Final-Exam	
Season	Date
Spring	20/4/96
Summer	22/8/96
Fall	10/12/96

Table 3.5 The E- relation

3.7.4.2 EG- relation (Event- Graph)

This is a binary relation which stores the precedence relationships of different events occurring in the application domain. There are two attribute fields in the relation. The SUP and SUB fields. An event that can appear in these two fields is just any attribute existing in the database or an event relation in the knowledge base. For example if *marks* is an attribute field in the database and *final-exam* is an event in the knowledge base, then the tuple (final-exam, enrolls.marks) in the EG-relation mean that the value of the final-grade will be known only after the final-exam event has happened. It is shown in table 3.6

EG – Relation	
SUP	SUB
Final-Exam	Entrolls.mark

Table 3.6 EG -relation

3.7.4.2 EXC- relation (Exception)

This is a binary relation which informs all the exceptional facts existing in the application domain. The tuples of this relation are concept- exception pair. The concept refer to attributes in the database and exception is a view in the V-relation or a basic entity in a database.

EXC - Relation	
Concept	EXC
Course.text	Thesis-course

Table 3.7 The EXC- relation

3.7.4.4 V- relation (View)

This is a unary relation which contains all the views defined in the EXC-relation. The definition of a view is similar to a view definition in system R. For example the view definition of *thesis-course* is

```

DEFINE VIEW thesis-course
AS SELECT courses.code
FROM courses
WHERE courses.course-id > cs51

```

The knowledge base contains a collection of view definitions of the V_relation.

V – Relation
Thesis-course

Table 3.8 V-Relation

The procedures Check-Exception and Check-Temporal-Event gives appropriate interpretation of the null values. This could be illustrated with the following example taken from the university academic database. Suppose the query is “CS51 enna courseintta pusthakam ethe ? . The SQL statement corresponding to it is

```

SELECT course.text
FROM course
WHERE course.course-id = “cs51”.

```

The result of the query is NULL. Instead of giving the answer “ don’t know”, The check-exception procedure gives a more apt response. This procedure checks whether any exception is stored for the attribute course.text in the EXC- relation. From fig Table 3.7 *thesis-course* is an exception to it. This exception is a view and it is executed. The course CS51 is a member of this set. Hence the message corresponding to this exception is given to the user . The response given is the Malayalam language equivalent of the statement “CS51 is a thesis course. Hence no textbook for it “.

Consider another query . “CS51innu thomasinnu ethra mark kitti “. The SQL statement corresponding to it is

```
SELECT entrolls.mark
FROM entrolls, student
WHERE student.stud-id = entrolls.stud-id AND
        Student.name = "thomas" AND
        Entrolls.course-id = "CS51".
```

The result of the query is NULL. First check-exception procedure is initiated. Since no exception is associated with the attribute, entrolls.mark, the procedure check-temporal-event procedure is initiated. From the EG-relation table, the event final-exam should precede the attribute entrolls.mark. From the E-relation the time for the final-exam is obtained. The reason for getting the NULL answer was that the final-exam was not over. Hence instead of giving the response "don't know", the response produced is "final-exam not over. Hence marks not available".

Chapter 4

Implementation

4.1 Software Design Methodology

Object-oriented methodology of software development is selected for this system. This method of software development was first proposed in the late 1960s. However it took almost 20 years for object technologies to become widely used. During the first half of the 1990s object-oriented software engineering became the paradigm of choice for many software product builders and information system professionals. As time passes, object technologies are replacing classical software development approaches. Object technologies do lead to a number of inherent benefits that provide advantages at both the management and technical level.

Object technologies lead to reuse, and reuses of program components lead to faster software development and higher-quality programs. Object-oriented software is easier to maintain because its structure is inherently decoupled. This lead to fewer side effects when changes have to be made and hence less frustration for the software engineer and the customers. In addition, object-oriented systems are easier to adapt and scale (ie. Large systems can be created by assembling reusable subsystems).

An object-oriented model of computer software exhibit data and procedural abstractions that lead to effective modularity. A **class** is an OO concept that encapsulates the data and procedural abstractions that are required to describe the content and behavior of some real world entity. The data abstractions (attributes) that describe the class are enclosed by a wall of procedural

abstractions . The only way to reach the attributes is to go through one of the procedural abstractions that comprise the wall. Therefore the class encapsulates the data and the processing that manipulates that data. This achieves information hiding .

4.2 Classes of the System

The classes identified for the general meaning representation system, machine translation system and natural language interface for databases are given in this section.

4.2.1 Meaning Representation System

The important classes identified for the system are given in fig 4.1.

No	Name of classes
1	Morphological-analyser
2	Local word grouper
3	Parser
4	Noun
5	Pronoun
6	Verbs
7	Modifiers

Fig 4.1 list of classes used in the general meaning representation system

The class definitions are given below.

Class morphological analyser

```
{
    protected:
        char query [ ];
        char querywords [ ][ ];
        char wordcategory [ ][ ];
        int wordpointer [ ];
    public :
        getquery ( );
        separatewords ( );
        getwordcategory ( );
};
```



Class localwordgrouper : public morphologicalanalyser

```
{
    Protected :
        char wordgroups[ ][ ];
    Public :
        Wordgrouping ( );
};
```

Class parser : public localwordgrouper

```
{
    protected :

        char Vroot[ ];
        int Verb form[ ];
        char Tense[ ];
```

```
int Vperson ;
char Vmodifier [ ] ;
char KKroot[ ] ;
int KKcategory ;
char KKmvalue[ ] :
int KKmcategory;
int KKnumber ;
int KKgender ;
int KKperson ;
char KRroot[ ] ;
int KRcategory ;
char KRmvalue [ ] ;
int Krmcategory;
int KRnumber ;
int KRgender ;
int KRperson ;
char SKroot [ ] ;
int SKcategory ;
char SKmvalue[ ] ;
int SKmcategory;
int SKnumber ;
int SKgender ;
int SKperson ;
char SMroot[ ] ;
int SMcategory ;
char SMmvalue[ ] ;
int SMmcategory;
int SMnumber ;
int SMgender ;
```

```

int SMperson ;
    char HKroot [ ];
    int HKcategory ;
    char HKmvalue[ ];
    int HKmcategory;
    char AKroot [ ];
    char AKdestination[ ];
    char AKtime[ ];
    char AKdate[ ];

    public :
        get_properties_of_words( );
        fill_slots ( );
        show_structure ( );
};

```

Class noun

```

{
    private :
        char root [ ];
        struct form
        {
            char suffix [ ];
            char vibakthi [ ];
        }ST;
        int gender, number, person;
        char semantic properties [ ];

    public :

```

```

        add_values_to_lexicon ( );
        search_value ( );
    }
Class pronoun
    {
        private :
            char root [ ];
            struct form
                {
                    char suffix [ ];
                    char vibakthi [ ];
                }ST;
            int gender, number, person;
        public :
            add_values_to_lexicon ( );
            search_value ( );
    };

```

```

Class Modifier
    {
        private :
            char root [ ];
            int category;
        public :
            add_values_to_lexicon ( );
            search_value ( );
    };

```

Class Verbs

```
{
  private :
      char root [ ];
      struct form
      {
          char suffix [ ];
          int verbform;
      }ST;
  public :
      add_values_to_lexicon ( );
      search_value ( );

};
```

The algorithm of the meaning representation program is as given below.

- . Display the start screen.
- . Read query.
- . Separate the words in the query.
 - Get categories of words.
 - Get the pointers to word properties.
 - Make word grouping.
 - Get the properties of the words.
 - Fill the slots of the meaning representation structure.
 - Display the meaning representation structure.

4.2.2 Machine Translation System

It makes use of all the classes given in section 4.2.1. In addition to that it has a class which generates sentences in English. Its definition is given below.

```
Class generator : public parser
{
    Protected:
        Char wordgroups[ ][ ];
    Public :
        Pattern_identification ( );
        Generate_English_sentences ( );
        Display_sentences ( );
};
```

4.2.3 Natural language Interface System

```
Class morphologicalanalyser
{
    protected:
        char query [ ];
        char querywords [ ][ ];
        char wordcategory [ ][ ];
        int wordpointer[ ];
    public :
        getquery ( );
        separatewords ( );
        getwordcategory ( );
};
```



```

Class localwordgrouper : public morphologicalanalyser
{
    Protected:
        char conditionalwordgroups[ ][ ];
        char outputwordgroups[ ];
    Public :
        Wordgrouping ( );
        DeepLevelEllipsisHandler( );

};

```

```

Class parser : public localwordgrouper
{
    protected :

        char Vroot[ ];
        char kkQueryWord [ ];
        char kkAttribute [ ];
        char kkEntity[ ];
        struct karmakarakam
            {
                char krValue [ ];
                char krAttribute [ ];
                char krRelation ;
                char krQuantifier [ ];
            }kr[ ];

    public :
        get_properties_of_words( );

```

```
        fill_slots ( );
        AnaphoricReferenceHandler( );
        SurfaceLevelEllipsisHandler( );
};
```

```
Class SQL_generator : public parser
```

```
{
    Protected :
        char SQL_statement [ ];
    Public :
        ProcessVerbs ( );
        ProcessKanthaKarakamList ( );
        ProcessKarmaKarakamList ( );
        JoinTables ( );
        FormSQLstatement ( );
};
```

```
Class Response_generator: public SQL_generator
```

```
{
    private:
        int error ;
    public :
        displayResults ( );
        displayErrorMessage ( );
        CheckException ( );
        CheckTemporalEvent ( );
};
```

Class prope noun

```
{
  private :
      char root [ ];
      struct form
      {
          char suffix [ ];
          char vibakthi [ ];
      }ST;
      int gender, number, person;
      char dbfieldnames[ ][ ];

  public :
      add_alues_to_lexicon ( );
      search_value ( );

};
```

Class common_noun_1

```
{
  private :
      char root [ ];
      struct form
      {
          char suffix [ ];
          char vibakthi [ ];
      }ST;
      int gender, number, person;
      char databasefieldnames[ ][ ];
```

```
public :  
    add_values_to_lexicon ( );  
    search_value ( );  
};
```

Class common_noun_2

```
{  
    private :  
        char root [ ];  
        struct form  
        {  
            char suffix [ ];  
            char vibakthi [ ];  
        }ST  
        int gender, number, person;  
        char databasetablenames[ ][ ];  
    public :  
        add_values_to_lexicon ( );  
        search_value ( );  
};
```

Class Verbs

```
{  
    private :  
        char root [ ];  
        char tablesInvolved [ ];  
    public :  
        add_values_to_lexicon ( );  
        search_value ( );  
}
```

Class queryWord

```
{  
    private :  
        char root [ ];  
        char databasefieldnames[ ][ ];  
    public :  
        add_values_to_lexicon ( );  
        search_value ( );  
};
```

Class relationWords

```
{  
    private :  
        char root [ ];  
        char operator;  
    public :  
        add_values_to_lexicon ( );  
        search_value ( );  
}
```

Class quantifier:

```
{  
    private :  
        char root [ ];  
    public :  
        add_values_to_lexicon ( );  
        search_value ( );  
};
```

Chapter 5

Performance Evaluation of the Model

5.1 Introduction

There are two basic types of performance evaluation. Black box evaluation and glass box evaluation. Black box evaluation measures system performance on a given task in terms of well defined input/output pairs and glass box evaluation examines the internal working of the system. Black box evaluation focuses on the accuracy of the output, user-friendliness, modularity, portability and maintainability. Black box evaluation is done without knowing anything about the inner workings of the system. The NLP systems could be judged on the basis of the following aspects

Coverage and Habitability

Coverage is a characterization of the linguistic competence of a system. Habitability measures how quickly and comfortably a user can recognize and adapt to system's limitations. The coverage of the NLP system could be measured along dimensions like lexical coverage, syntactic coverage and semantic coverage [61]. Lexical coverage refers to the size of the vocabulary, internal structure of the vocabulary and the easiness with which the vocabulary could be extended. Syntactic coverage refers to the range of syntactic phenomena the system can deal with. It includes complex verb forms, relative clauses, various question forms, passives, comparatives, subordinate clauses, ellipsis etc. Semantic coverage refers to the extent to which the system understands the domain. The critical issues regarding coverage are whether the system has enough coverage to let users meet a

reasonable proportion of their needs, whether the user can quickly find an appropriate way of expressing a request and whether the user can easily learn to avoid the system's blind spot.

A system's habitability is reduced if the user is led to believe that the system has capabilities that are beyond it and there is no clear indication of the boundaries. This can happen if the language the system presents to the user is not the language that the user can present to the system. The difficulties in achieving habitability with a semantic grammar are based on the fact that without great care such grammars can give users misleading clues as to coverage.

Inference

This is the process of drawing logical conclusions based on the data in the database or general knowledge of the subject domain. Retrieving only data that is explicitly stored in a database is usually insufficient to meet a normal user's needs. The system should have the capability to infer new information from that already existing in the database.

Anaphora

Pronouns are special case of the linguistic phenomena called anaphoric reference. Pronouns usually refer objects explicitly mentioned in previous discourse. Sometimes they can refer to objects mentioned later. Pronouns can also refer to actions. NLP systems should be capable of handling the usage of pronouns. Sometimes pronouns refer to objects in the computer's previous response, not just objects in the user's own language

Ellipses

In conversation, people often leave out large portion of sentences, assuming that the listener, who shares the context being discussed, can fill in the missing parts. A good NLP system should be capable of handling any kind of ellipses.

Quantification

The use of words like some, every, all and any can complicate NLI system because their interpretation often depends on wide ranging commonsense knowledge or on detailed knowledge of the particular domain. The NLI system should be capable of tackling quantification appropriately.

Presentation of output

This includes formatting reports and tables, interfacing to graphics modules and generating output in the user's own language.

5.2 Performance of the NLI System for Information Retrieval

The black box testing of the system was carried out with three databases. An election database, an academic database and a library database. A summary of the performance of the system is given in the table 5.1. The system was evaluated by giving about 30 typical queries from each application domain. The queries included simple straight forward queries, multi-relational queries, queries with conjunctions, and quantifiers, elliptical queries and queries with anaphoric references.

The table shows that the accuracy of the system is 100% for simple straight forward queries. The queries of this type manipulates only a single relation for getting the output. For multi-relational queries , excluding queries with anaphoric / elliptical references the accuracy of the system is around 90 % . The accuracy of the anaphoric / elliptical queries is around 85% . Accuracy could be increased by adding more rules to the knowledge base used by the parser module of the system.

Database Names	No. of Questions asked	Query type		Percentage of Queries correctly interpreted			
				Straight forward queries	Multi relational queries		
					Conjunctions	Quantifier	Elliptical/ Anaphor
Election	30	Straight forward Queries	15%	100%	92%	95%	85%
		Multi Relational Queries	85%				
Academic	30	Straight forward Queries	25%	100%	87%	92%	82%
		Multi Relational Queries	75%				
Library	30	Straight forward Queries	20%	100%	93%	92%	87%
		Multi Relational Queries	80%				

Table 5.1. Performance Table

The Lexicon creator program could be used for creating the lexicon for the various application domains. Since the content of the lexicon is very much dependent on the structure of the database, a person who is thorough about the database schema is responsible for the development of it. No change in the meaning representation format is required. But the tables and some rules associated with the SQL generator are to be rewritten since the rules in this module are domain dependent. For handling modifiers, which refer to special procedures and verbs which refer to the relationships between the basic entities, new rules are to be added to the SQL generator module. Hence a person having at least the proficiency of a programmer is required for customizing this system for any database. The system could be scaled to larger databases.

Glass box evaluation of the system focuses on the performance of the important functional modules of the system. In this case the performance of individual modules like morphological analyser, parser, semantic interpreter, response generator, SQL generator etc were evaluated separately. A black box evaluation of a particular component's performance could be considered as a form of glass box evaluation. For example the evaluation of a parser with a set of specified input/output pairs would be a black box evaluation of the parser. Since it is an evaluation of a component that cannot by itself perform an application, and since it will give information about the component's coverage that is independent of the coverage of the system in which it might be embedded, this can be considered as providing glass box information for the overall system [62]. The testing of the individual modules were carried out when the system was under development.

The table 5.2 gives a comparison of the various size aspects of the three databases. The important fact derived from the table is that the size of the

lexicon is significantly small when compared to the size of the database. An important issue in NLI to DBMS is the relative size of the lexicon as compared to the size of the database itself [63,64]. As mentioned chapter 3, the lexicon stores all words that are to be understood by the system. While attribute values are stored repeatedly in the database, they are stored only once in the lexicon. For example, in the election database, the party name is stored for each candidate in the database, while party names are stored only once in the lexicon. Also numeric values and code values are not stored in the lexicon. These two factors make the size of the lexicon significantly small when compared to the size of the database.

Some of the typical queries processed by the systems are given next. First the text in Malayalam is given. Then the Roman notation of it is given. Then query in English language, which is equivalent to it, is given next. Finally the SQL statement and result of query processing are given.

1. പാലാ നിയോജക മണ്ഡലത്തിൽ എത്ര സ്ഥാനാർത്ഥികൾ മൽസരിച്ചു ?
 pAlA niyojk mNtlwWi2l ewR szWanaRWik2L m2lsriccu ?
 (How many candidates contested from Pala constituency ?)

```
SELECT COUNT(candidate.candname)
FROM candidate, contest, cons
WHERE candidate.candcode = contest.candcode AND
      contest.conscode = constituency.conscode AND
      constituency.consname = 'pAlA'
```

candidate.candname

Database Names	No. of Tables	Table Names	Rows Per Table	No. of Attributes per table	Ratio of Size of Database to Lexicon
Election	5	Party	4	2	5 : 1
		Constituency	100	4	
		Candidate	400	4	
		Contest	400	4	
		Win	100	3	
Academic	7	Student	100	5	7 : 1
		Quarter	3	2	
		Courses	30	3	
		Department	4	4	
		Teacher	17	5	
		Entrolls	800	5	
		offers	30	4	
Library	5	Books	1500	5	3 : 1
		Borrower	100	3	
		Circulation	400	2	
		Publisher	10	3	
		Topic	10	2	

Table 5.2 Database size comparison table

2. തോമസ് എന്ന സ്ഥാനാർത്ഥിയുടെ വയസ് എത്ര ?
 wOmsz enn szWnaRWiyute vyz ewR ?
 (What is the age of the candidate Thomas ?)

```
SELECT candidate.age
FROM candidate
WHERE candidate.candname = 'wOms'
```

Candidate.age

56

3. 70 വയസിൽ കൂടുതലുള്ള സ്ഥാനാർത്ഥികൾ ആരെല്ലാം ?
 70 vySi2l kutuwluLL szWanaRWiK2l ArellAM ?
 (List the candidates with age greater than 70 .)

```
SELECT candidate.candname
FROM candidate
WHERE candidate.age > 70
```

Candidate.candname

1. ke. KruNAkrV
2. ti. PqAVsisz
3. vrcgIsz
4. ell.seviyrc
5. seviyrcaRkkIl
6. eM.pi.pHIOSz
7. eM.pi.mANi

4. കോൺഗ്രസ് പാർട്ടിയിൽ തോറ്റവർ ആരെല്ലാം ?
 KOQgrsz pA2Rtti2l wORRv2R ArellAM ?
 (List the candidates who have lost the contest in Congress party.)

```

SELECT candidate.candname
FROM candidate, win, party
WHERE candidate.candcode <> win.candcode AND
      candidate.partyname = party.partyname AND
      party.partyname = 'kOQgrsz'

```

candidate.candname

1. pi.ke.pwRosz
2. eM.pi.mANi

5. ബിജെപിക്ക് എത്ര വോട്ടു കിട്ടി ?

bijepikkz ewR Vottu kitti ?

(What is the total number of votes won by BJP party ?)

```

SELECT SUM (contest.vote)
FROM contest, party, candidate
WHERE contest.candcode = candidate.candcode AND
      candidate.partyname = party.partyname AND
      party.partyname = 'bi.je.pi'

```

contest.vote

10,11156

6. കോട്ടയം ജില്ലയിൽ ജയിച്ച കോൺഗ്രസ്കാർ ആരെല്ലാം ?

kOttyM jillyi2l jyicc koQgrszkA2R ArellaM ?

(Who are the Congress candidates who have won from kottayam district?)

```

SELECT candidate.candname
FROM win, candidate, constituency, party
WHERE candiadte.candcode = win.candcode AND
      win.conscod = constituency.conscod AND
      candidate.partyname = party.partyname AND

```

party.partyname = 'kOqgrsz' AND
constituency.district = 'kOttyM'

candidate.candname

1. eM.pi.pHIOsz
2. si.je.Zskkz

7. എവിടെല്ലാം കോൺഗ്രസിന്റെ സ്ഥാനാർത്ഥികൾ മത്സരിച്ചു ?

evitellAM kOQgrsi2nte szWAnA2Rwik2L m2lsriccu ?

(Give the names of the constituencies from which congress candidates have contested ?)

```
SELECT constituency.consname  
FROM contest, candidate, constituency, party  
WHERE candiadte.candcode = contest.candcode AND  
contest.conscore = constituency.conscore AND  
candidate.partyname = party.partyname AND  
party.partyname = 'kOQgrsz'
```

constituency. Consname

1. pAIA
2. wOtupuV
3. krinAgppLLi

8. കോൺഗ്രസിന്റെ എത്രെല്ലാം സ്ഥാനാർത്ഥികൾ ജയിച്ചു ?

kOQgrsi2nte ewellAM sWanaRWik2L jyiccu ?

(List the names of the candidates who have won with a congress ticket.)

```
SELECT candidate.candname  
FROM win, candidate, party  
WHERE candiadte.candcode = win.candcode AND  
candidate.partyname = party.partyname AND  
party.partyname = 'kOQgrsz'
```

candidate.candname

1. e.joQsz
2. sumM

9. പാലായിൽ ജയിച്ചത് ആര് ?

pAlAyi2l jyiccwz Arz ?

(Who has won from Pala ?)

```
SELECT candidate.candname
FROM candidate, win, constituency
WHERE win.candcode = candidate.candcode AND
      Win.conscode = constituency.conscode AND
      Constituency.consname = 'pAlA'
```

Candidate.candname

1. sjIv2n

10. പ്രൊഫസർ തോമസ് പഠിപ്പിച്ച വിഷയങ്ങൾ ഏവ ?

pRoPs2R wOmsz pTippicc ViYyff2L Ev ?

(Give the names of courses offered by Prof. Thomas.)

```
SELECT course.coursename
FROM teacher, offers, course
WHERE teacher.teachid = offer.teachid AND
      offers.courseid = course.courseid AND
      teacher.teachername = 'wOmsz' AND
      teacher.designation = 'pRoPs2R'
```


course.coursename

1. deRRAbesz
2. grAPikzsz

11. എല്ലാ ഇലട്രോണിക്സ് വിദ്യാർത്ഥികളുടെയും പേര് തരിക ?
ellA iLtRONiksz vidyarthikkaluteyum pErz Wrik ?
(Give the names of all students of the Electronics department ?)

```
SELECT student.name  
FROM student, department  
WHERE student.deptid = department.deptid AND  
department.deptname = 'iLtRONiksz '
```

student.name

1. e.joQsz
2. sumM

12. ഇലട്രോണിക്സിൽ എത്ര വിദ്യാർത്ഥികൾ ഉണ്ട് ?
ILtRONiksze2L ewR vidyarthikkal unndu ?
(How many students are there in the Electronics department ?)

```
SELECT COUNT(student.studname)  
FROM student, department  
WHERE student.deptid = department.deptid AND  
department.deptname = 'iLtRONiksz '
```

Student.studname

2

13. ഡേറ്റാബേസ് കോഴ്സിന്റെ ആവരേജ് മാർക്ക് എത്ര ?
dERRAbEsz kOVzsi2nte AvREjz mA2Rkkz ewR
(what is the average mark of database course ?)

```
SELECT AVG(entrolls.mark)
FROM entrolls, course
WHERE entrolls.courseid = course.courseid AND
      course.coursename = 'dERRAbEsz'
```

entrolls.mark

56.5

14. 100ൽ കൂടുതൽ വിദ്യാർത്ഥികൾ ഉള്ള ഡിപ്പാർട്ടുമെന്റ് ഏതെല്ലാം?
100Il kutuw2l vixyA2RwWikL uLL dipARRum2neRz EwellAM
(Which departments have more than 100 students ?)

```
SELECT department.deptname ,COUNT(student.studname)
FROM department, student
WHERE student.deptid = department.deptid
GROUP BY department.deptname
HAVING COUNT(student.studname) > 100
```

Department.deptname

1. mawwmaRRikzs

15. കമ്പ്യൂട്ടർഗ്രാഫിക്സിന്റെ എത്ര പുസ്തകങ്ങൾ ഉണ്ട് ?
kMpUtt2RgrAPikzsi2neR ewR puswkff2L untz ?
How many books are their for computer graphics ?

```
SELECT COUNT (book.accno)
FROM book, subcode
WHERE book.scode = subcode.scode AND
      subcode.string = 'kMpUtt2RgrAPiksz'
```

book.accno

1. 15

14. കെ.ആശ എത്ര പുസ്തകങ്ങൾ എടുത്തിട്ടുണ്ട് ?

k.AS ewR puswkff2L etuw Wittuntz ?

(How many books did K.Asha take ?)

```
SELECT COUNT (circulation.Idno)
FROM circulation, borrower
WHERE circulation.idno = borrower.idno AND
      Borrower.name = 'k.As'
```

Circulation.Idno

1. 5

5.3 Performance of the Machine Translation System

In the system for machine translation, the lexicon creator object is used for building the lexicon. The vocabulary is spread over 4 dictionaries. About 2500 root words are stored in the dictionaries. Declarative sentences in the active voice are analyzed. Both simple and complex sentences are considered. Since in a general context anaphoric references are difficult to consider, sentences are treated isolated. Each sentence is analyzed independently, with the result that a previous sentence does not affect the interpretation of the following sentence. Any ambiguity in this regard is transferred from source language to destination.

Several sentences were translated using this system. Some sample sentences are given in table 6.3.a and table 6.3.b. The sentences are coded using the Roman notation given in appendix – 4. In the tables “M” denotes the Roman notation of the sentence, “E” denotes the English Translation and “Pattern” denotes the pattern to which the Malayalam sentence is mapped. Sentences in pure Malayalam language is also given with in brackets.

The story about the Lion and the Rat translated by the system from Malayalam to English is given next. The text in Malayalam is given in table 6.4 and the text in Roman notation is given in 6.5 and the English translation obtained is given in table 6.6.

The sentences were selected in such a manner that they could be mapped to one of the 15 patterns selected in chapter 3. All the sentences generated correct translation. The analyzer's ability was tested further by letting it handle more types of sentences. During this extension stage, it was found that more verb forms should be included, more semantic tags were required for word sense disambiguation, more patterns should be added to the generator and more rules regarding karaka sharing should be included. The performance of the system increases by incorporating more knowledge to the system.

1. M. kutti av2neR PuszwkM vAjiccu (കുട്ടി അവന്റെ പുസ്തകം വായിച്ചു.)
 E. The child read his book.
 Pattern. (Subject+Verb+Object)
2. M kARRz atikkunnu (കാറ്റ് അടിക്കുന്നു.)
 E. The wind is blowing.
 Pattern. (Subject+Verb)
3. M rAmu rAjuvinz pe2nsi2l kotuwvu (രാമൻ രാജുവിന് പെൻസിൽ കൊടുത്തു.)
 E. Ramu gave Raju a pencil
 Pattern. (Subject+Verb+In-direct object+direct object)
4. M. rAmu pAwRff2L vqwweyAyi (രാമു പാത്രങ്ങൾ വൃത്തിയായി കഴുകി)
 E. Ramu washed the plates clean
 Pattern. (Subject+Verb+Object+Object complement)
5. M av2n bAgz wiryunnu (അവൻ ബാഗ് തിരയുന്നു)
 E. He is Searching for the bag
 Pattern. (Subject+Verb+preposition+Prepositional Object)
6. M rAju eyuwwz aykkuvA2n mRnnu (രാജൻ എഴുത്ത് അയക്കുവാൻ മറന്നു)
 E. Raju forgot to post the letter
 Pattern. (Subject+Verb+to-Infinitive)

Table 6.3a Some Sample Sentences Translated

- E. He told me to go there
Pattern. (Subject+Verb+Indirect object+to-infinitive)
8. M. av2n pAIM ktkkunnWz fA2n kNtu
E. I saw him crossing the bridge.
Pattern. (Subject+Verb+Present Participle+clause)
(അവൻ പാലം കടക്കുന്നത് ഞാൻ കണ്ടു.)
9. M. suXyute pe2nsi2l etuwwitz rAmu ptM vrccu
E. Raman drew a figure having taken Sudh's pencil
Pattern. (Subject+Verb+Object+Perfect Participle+clause)
(സുധയുടെ പെൻസിൽ എടുത്തിട്ട് രാമൻ പടം വരച്ചു)
10. M. e2neR pErz viLiccwz FA2n kettu.
E. I heard my name called.
Pattern. (Subject+Verb+Object+Past participle+clause)
(എന്റെ പേര് വിളിച്ചതു ഞാൻ കേട്ടു)
11. M. kutti pTikkukyANennz amm pRffu
E. Mother told that the child was
Pattern. (Subject+Verb+that-clause)
(കുട്ടി പറിക്കുകയാണെന്ന് അമ്മ പറഞ്ഞു)
12. M. kwkz wuRnppO2L pUcc cAti.
E. The cat went out when the door was opened.
Pattern. (Subject+Verb+when-clause)
(കരക് തുറന്നപ്പോൾ പുച്ചു ചാടി.)

Table 6.3b Some Sample Sentences Translated

1. ഒരിക്കൽ ഒരു സിംഹം ഒരു കാട്ടിൽ ഉറങ്ങുകയായിരുന്നു.
2. ഒരു എലി ഇതു കണ്ടു.
3. ആ എലി സിംഹത്തിന്റെ അടുത്ത് കളിക്കുവാൻ തുടങ്ങി.
4. സിംഹത്തിനു ദേഷ്യം വന്നു.
5. സിംഹം എലിയെ തിന്നുമെന്ന് പറഞ്ഞു.
6. എലി തന്നെ കൊല്ലരുതെന്ന് പറഞ്ഞു.
7. ഒരു ദിവസം സിംഹത്തെ എലി സഹായിക്കാമെന്നു പറഞ്ഞു.
8. സിംഹം എലിയെ വിട്ടയച്ചു.
9. ഒരു ദിവസം ആ സിംഹം ഒരു കെണിയിൽ വീണു.
10. അത് ഉച്ചത്തിൽ ഗർജ്ജിച്ചു.
11. അതു കേട്ടിട്ട് എലി വേഗം വന്നു.
12. എലി കെണിയുടെ കയറ് പല്ലുകൊണ്ട് അറുത്തുമാറ്റി.
13. സിംഹം കെണിയുടെ പുറത്തു വന്നു.
14. സിംഹം എലിക്ക് നന്ദിപറഞ്ഞു.
15. സിംഹം അതിന്റെ വഴിക്ക് പോയി.

Table 6.4 Malayalam text of the story “*The Lion and the Rat*”

1. orikk2l oru siMhm oru katti2l urffukyAyirunnu.
2. oru eli iwu kNtu.
3. A eli siMhwwi2neR atuwwu kLikkuvA2n wutffi.
4. siMhwwinnu xeshyM vnnu.
5. siMhM eliyē winnumennu pRffu.
6. elie wne kollruwennu pRffu.
7. oru xivsM siMhwwinne eli shAyikkAmennu pRffu.
8. simhM elie vittyccu.
9. oru xivsM A siMhM oru keNiyiIl vINu.
10. awz uccwwiIl g2Rjjiccu.
11. awz kettz eli vEgM vnnu.
12. eli kenniyute kyRz pllukoNtu aRwwumAtti.
13. siMhm kenniyute puRwwu vnnu.
14. siMhM elikkz nni pRffu.
15. simhM awi2neR vVikkuz pOyi.

Table 6.5 Story in Roman Notation

1. Once upon a time a lion was sleeping in a forest .
2. A rat saw this .
3. That rat started to play near the lion .
4. The lion grew angry .
5. The lion said that it will eat the rat .
6. The rat said not to kill it .
7. The rat told that it will help the lion one day .
8. The lion set the rat free .
9. This lion fell into a trap one day .
10. It roared loudly.
11. The rat came quickly having heard it .
12. The rat cut open the trap with its teeth .
13. The lion came out of the trap .
14. The lion thanked the rat .
15. The lion went its way .

Table 6.6 English Translation of the Story

The results show that the method comprising the usage of verb boundaries as phrase markers, the karaka based approach of extracting meaning from phrases and the frame structure for meaning representation is an ideal approach for sentence comprehension in Dravidian languages. The object-oriented methodology used, facilitates portability and scaling up of this model.

Chapter 6

Conclusion & Future work

The conventional methods of sentence analysis make use of a context-free grammar and a parser before proceeding further. Context-free grammars are capable to handle order and position elegantly. Hence they are suitable for processing positional languages like English. But Dravidian languages or other Indian languages are basically free word order languages. Hence the technique of processing these languages with CFG is actually a misfit.

In this work an alternative approach more efficient in the context of Indian languages is suggested. This approach makes use of the karaka relations for sentence comprehension. An intermediate representation of the sentence is built from the karaka relations. This representation is unambiguous and general.

This approach is based on the concept that the verbs convey the action part in a sentence and the karaka-vibakthi relations enable to find the respective places of the various components in the sentence. Instead of the conventional NP-VP grammars, verb boundaries are adopted as phrase markers and phrase-level representations are build to convey the complete meaning of a sentence.

The validity of the approach is established through studies on two application-oriented experiments. In the first experiment translation is done from a free word order language to fixed word order one. Here both source and destination are natural languages. In the second experiment the target language is an artificial language with a rigid syntax. The same meaning representation technique is used in both the cases. The difference

is in the generation of target language sentence. Anyhow both use pattern directed methods. The results obtained are encouraging.

Simple and complex sentences of declarative type and interrogative type are selected for analysis. It could be extended to handle imperative and exclamatory sentences. Complex sentences with more than one subordinate clause and compound sentences could also be explored.

In the first experiment isolated sentences were only considered. Hence the methods for resolving anaphora were not studied. In a general context this is a difficult job. However work could be extended in the direction of resolving anaphoric references in the intermediate representation. In the second experiment anaphora and ellipsis are handled satisfactorily since the domain is fixed. The possibility of improving system performance by tolerant towards user errors like spelling mistakes, sentence structures, agreement rules etc could be investigated.

Word sense disambiguation is done with semantic tags. This scheme is basically simple to implement. But it is an ad hoc scheme as no universal set of semantic primitives is yet identified. The lexicon should be built carefully after considering all the cases.

Since karaka relations are present in almost all Indian languages, an added advantage of this approach is its adaptability to all Indian languages. The general nature of the meaning representation structure and the object based design of the system helps in achieving easy transportability to a new application. This work could be extended to other free word order languages, other than Indian languages. The chances of applying this approach to fixed word order languages could also be probed. What is needed is the coding of vibakthi information in the word position.

However, as far as machine translation is considered, this approach has a serious drawback of losing surface structure in the internal representation. In fact this approach presumes a perfect understanding of the source language. For translation among languages, which are structurally close to each other, a direct lexical substitution in a language pair could lead to reasonable results in a simplistic manner.

In this work the input was given as typed text. A voice recognition system could also be included in the system so that input could be given as spoken text, making it particularly useful in information booths.

References

1. Wendy B. Rauch-Hindon, *Artificial Intelligence in Business, Science and Industry*, vol .1, Prentice-Hall, New Jersey, 1986.
2. E.C.Charniack, *Towards a Model of Children's Story Comprehension*, TR-266, MIT AI Lab, Cambridge, MA, 1972.
3. R.C.Schank, *Conceptual Information Processing*, North-Holland, Amsterdam, 1975.
4. Avron Barr, Edward A. Feigenbaum, *The handbook of Artificial Intelligence*, Vol 1, William Kaufmann Inc, California, 1981.
5. P.R.Cohen and C.R.Perrault, *Elements of a Plan-based Theory of Speech Acts*, *Cog.Sc* 3,1979.
6. B.J. Grosz, The representation and use of focus in a system for understanding dialogues, *Proceeding of the fifth International joint conference on Artificial Intelligence*, Cambridge, MA, 1977.
7. C.L. Sidner, *Towards a Computational theory of Definite Anaphora Comprehension in English Discourse*, TR-537, MIT AI lab, Cambridge MA, 1979.
8. G.G. Hendrix, Human Engineering for Applied Natural Language processing, *Proceedings of the fifth International Joint Conference on Artificial intelligence*, Cambridge MA,1977.

9. B.J. Grosz, TEAM: A Transportable Natural Language Interface System, *Proceedings of the Conference on Applied Natural Language processing*, Santa Monica, CA, February, 1983.
10. J.Kaplan, Cooperative responses from a Portable Natural Language Query System, *Artificial Intelligence*, vol 19, no.2, 1982.
11. J.G.Brown and R.R.Burton, Multiple representation of Knowledge for Tutorial Reasoning in D.G.Bobrow and A.Collins (eds), *Representation and Understanding*, Academic press, New York, 1975.
12. J.G. Carbonell, W.M.Boggs, M.L.Mauldin and P.G.Anick, The XCALIBUR Project, A natural Language Interface to Expert Systems, *Proceedings of the Eight International Joint Conference on Artificial Intelligence*, Karlsruhe, FRG , 1983.
13. P.Norvig, *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*, Kaufmann. California, 1991.
14. R.C. Parkinson, K.M.Colby and W.S. Faught, Conversational Language Comprehension using Integrated Pattern - matching and Parsing, *Artificial Intelligence* Vol. 10, No.2, 1977.
15. Mark Wallance, *Communicating with Databases in Natural language*, Ellis Horwood, England, 1984.
16. William B. Gevarter, *Intelligent Machines*, Prentice- Hall, Inc, Englewood Cliffs, New Jersey, 1985.

17. Michael A. Covington. *Natural language Processing for Prolog Programmers*, Prentice-Hall, Englewood Cliffs, New Jersey, 1994.
18. G. Gazdar, Phrase Structure Grammar and Natural Language, *Proceedings of the Eight International Joint Conference on Artificial Intelligence*, Karlsruhe, FRG, August, 1983.
19. D.G.Bobrow and J.B.Fraser, An Augmented State Transition Network analysis Procedure, *Proceedings of the first International Joint Conference on Artificial Intelligence*, Washington D.C., 1969.
20. W.A.Woods, Transition network Grammars for Natural language Analysis, *CACM* 13 (10), October,1970.
21. S.C Kwansy and N.K Sondheimer, Relaxation Techniques for Parsing Grammatically Ill-Formed Input in Natural language Understanding System, *American Journal of Computational Linguistics* 7 (2) , 1981.
22. R.M. Weischedel and J.Black, "Responding to Potentially unparseable Sentences, *Am.J. Computatal linguistics* 6, 1980.
23. W.A.Woods, W.M.Bates, G.Brown, B.Bruce, C.Cook, J.Klovstad, J.Makhoul, B.Nash-Webber, R.Schwartz, J.Wolf and V.Zue, *Speech Understanding Systems*, Final Technical Report 3438, Bolt, Bernack and Newman, Cambridge, MA, 1976.
24. Avron Barr, Edward A. Feigenbaum, *The handbook of Artificial Intelligence*, Vol II, William Kaufmann Inc, California, 1982.

25. William B. Gevarthen, *Intelligent machines*, Prentice-Hall, Inc, NJ, 1985.
26. R.M. Kaplan and Joan Bresnan, Lexical functional grammar : A formal system for grammatical representation, In Joan Bresnan (ed), *The Mental Representation of Grammatical Relation*, MIT Press, Cambridge, MA, 1982.
27. Elaine Rich, Kevin Knight, *Artificial Intelligence*, Tata McGraw – Hill Publishing Company, 1991.
28. R. Schank and C.K. Riesbeck, *Inside Computer understanding*, Lawrence, Erlbaum. Hillsdale, NJ, 1980.
29. C.K. Riesbeck Conceptual analysis, in R.C. Schank (ed), *Conceptual Information Processing*, North Holland/American Elsevier, Amsterdam, 1975.
30. R. Schank and R. Abelson, *Scripts, plans, goals and understanding*, Lawrence Erlbaum, Hillsdale, NJ, 1977.
31. L. Birnbaum and M. Selfridge, Conceptual Analysis of Natural language, in R.C Schank and C.k. Riesbeck (eds), *Inside Computer Understanding*, Lawrence Erlbaum, Hillsdale, NJ, 1981.
32. Y. Wilks, A Preferential Pattern-matching Semantics for Natural Language Understanding, *Artificial Intelligence* 6, 1975.

33. S.Small and C.Rieger, Parsing and Comprehending with Word Experts (A Theory and its Realization), in W.G.Lehnert and M.Ringle (eds), *Strategies for Natural language Processing*, Lawrence Erlbaum, Hillsdale, NJ,1982.
34. M.Lebowitz, Memory-based Passing, *Artificial Intelligence*, 21, 1983.
35. D.E. Rumelhart, J. McClelland , On Learning the Past Tense of Verbs in D.E. Rumelhart, J. McClelland (eds) *Parallel Distributed processing* ,vol. 2, MIT Press, Cambridge, MA, 1988.
36. J.L. McClelland and A.H. Kawamoto, Mechanism of Sentence processing : Assigning Roles to Constituents in D.E. Rumelhart, J. McClelland (eds) *Parallel Distributed processing* ,vol. 2, MIT Press, Cambridge, MA, 1988.
37. T.J. Sejnowski, C.R. Rosenberg, *NET talk: a parallel network that learns to read aloud*, Technical Report JHU/EECS-86/01, John Hopkins University Press, Baltimore, MD ,1986.
38. A Waibel, T. Hanazawa, G. Hinton, K.Shikano, K.J.Lang, Phoneme recognition Using Time Delay Neural Networks, *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37, 1989.
39. M.I.Jordan, Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, *Proceedings of the Eighth Annual conference of the Cognitive Science Society*, 1986.

40. J.L.Eلمان, *Representation and Structure in Connectionist Models*, Technical Report 8903, CRL, University of California, San Diego, CA 1989.
41. J.B.Pollack, Recursive Auto-Associative Memory: devising compositional distributed representations , *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Montreal, Canada, 1988.
42. M.F. St.John, J.L.McClelland, Learning and Applying Contextual Constraints in Sentence Comprehension, *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, 1988.
43. G.Cottrell, B.Bartell, C.Haupt, Grounding meaning in Perception, Marburge H. (ed) *Proceedings of the 14th German workshop on Artificial Intelligence*, 1990.
44. J.B.Pollack, Implication of recursive distributed representations in Touretzky D.S (Ed). *Advances in Neural Information Processing systems*, Morgan Kaufmann; San Mateo, CA, 1989.
45. W.G.Lehnert, C.Cardie, D.Fischer, J.McCarthy, Description of the CIRCUS system as used for MUC -4, *Proceedings of the fourth message understanding Conference*, 1992.
46. J.Hendler, Developing hybrid symbolic connectionist models. In : J.A Baurden, J.B.Pollack (eds). *Advances in Connectionist and Neural Computation Theory*, Vol.1 : High level Connectionist Models, Ablex Publishing Corporation, Norwood, NJ, 1991.

47. W.R.Deshpande, Technology development for Indian languages : Language understanding and machine translation, in. S.S.Agarwal, Subas Pani (ed.) *Proceeding of Information technology application in language, script and speech*, New Delhi, India, 1994.
48. Akshar Bharti, Vinceent Chaitanya, Rajeev Sangal, *Anusaraka or Language Accessor : A short introduction*, TRCS – 93-205, Department of Computer Science & Engineering, IIT Kanpur, 1993.
49. Akshar Bharti, Vineent Chaitanya, Rajeev Sangal, *Anusaraka, as a measuring devices for the linguists*, TRCS – 93-210, Department of Computer Science & Engineering, IIT Kanpur, 1993.
50. R.M.K. Sinha and K.Sivaraman, *ANGAL-BHARTI : A machine aided translation system from English to Indian languages – an overview –* Technical Report TRCS – 93-173, Department of Computer Science and Engineering, IIT Kanpur, 1993.
51. G. Gazdar and C. Mellish. *Natural language Processing in Lisp*, Addison-Wesley, 1989. Prolog version also available.
52. Akshar Bharathi, Vineet Chaithanya, Rajeev Sangal, *Natural Language processing- A paninian perspective*, Printice-Hall of India, New Delhi, 1996.
53. N.N.Moosath, *Dravida Baksha Syasthram*, National Bookstall, 1973.

54. Akshar Bharati, Rajeev Sanyal, *Parsing free word order languages in the Paninian frame work*, TRCS-93-171, Department of Computer Science and Engineering, IIT, Kanpur, India, 1993.
55. R.M.K.Sinha and K. Sivaraman, *Ambiguity Resolution in Anglabharati*, TRCS-93-174, Department of Computer Science and Engineering, IIT, Kanpur, India, 1993.
56. R.M.K. Sinha, Aditi Agrawal and Chinmoyee Sanyal, *Morphological analyser*, TRCS – 93-176, Department of Computer Science IIT, Kanpur, India, 1993.
57. N. Alwar, S.Raman, *"An AI based approach to machine translation in Indian languages*, ACM Communications, May 1990.
58. Akshar Bharati, Y.Krishna Bhayava, Rajeev Singal, *Reference and Ellipsis in an Indian Languages Interface to Databases* , TRCS – 92-147, IIT, Kanpur.
59. J. Kaplan. Cooperative Responses from a Portable natural language Data Base Query System, Artificial Intelligence, 1982.
60. Mimi Kao, Nick Cercone, Wo-shun Kuk, Providing quality responses with natural language interfaces : The Null value Problem, *IEEE Transactions on software engineering*, vol.14, no.7, July, 1988.
61. B.Rondand etal, A procedure for the evaluation and improvement of an MT system by the end-user, in *Machine Translation* 8, No 1-2, Special issue on Evaluation of MT systems, 1993.

62. K.Sparck Jones and J.Galliers, *Evaluating Natural Language processing system. An Analysis and Review*, Heidelberg and Berlin : Springer Verlag, 1996.
63. I. Androutsopoulos. *Interfacing a Natural Language Front-End to a RelationalDatabase (MSc thesis)*, Technical paper II, Department of Artificial Intelligence, University of Edinburgh, 1993.
64. I. Androutsopoulos, G. Ritchie, and P. Thanisch. An efficient and Portable Natural Language Query Interface for Relational Databases. In P.W. Chung, G. Lovegrove, and M. Ali, editors, *Proceedings of the 6th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, Edinburgh, U.K., Gordon and Breach Publishers Inc., Langhorne, PA, U.S.A., June 1993.
65. A.R.Rajaraja Verma, *Keralapaniniyum*, India Press, Kottayam, 1985.
66. P.C.Wren, H.Martin, *High School grammar and composition*, S.Chand & company, 1988.
67. N.R. Adam, A. Gangopadhyaya, A Form-Based Natural Language Front-end to a CIM Database, *IEEE Trans. on Knowledge and Data Engineering*, vol.9, no.2, Mar-Apr 1997.