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 Draviadian 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-Altaic 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 viewpoint. The speaker’s viewpoint 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.

<table>
<thead>
<tr>
<th>Vibakthi</th>
<th>suffix</th>
<th>word</th>
</tr>
</thead>
<tbody>
<tr>
<td>nirdesika</td>
<td>-</td>
<td>kavi</td>
</tr>
<tr>
<td>prathigrahika</td>
<td>e</td>
<td>kavie</td>
</tr>
<tr>
<td>samyojika</td>
<td>Ode</td>
<td>kaviyOde</td>
</tr>
<tr>
<td>udesika</td>
<td>ikke, ine</td>
<td>kavikke</td>
</tr>
<tr>
<td>prayojika</td>
<td>Ai</td>
<td>kaviAi</td>
</tr>
<tr>
<td>sambandhika</td>
<td>inte, ude</td>
<td>kaviude</td>
</tr>
<tr>
<td>aadharika</td>
<td>il, kl</td>
<td>kaviyil</td>
</tr>
</tbody>
</table>

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 sakannaka 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 Krishnan Odu vazhakkittu. (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 adhikarna 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.)
Ramanu 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. padikkom (shall learn)
12. padichhapol (when learnt)
13. padikkukanayennu (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.
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](image)

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

The pronoun “aval” is stored as
(aval,O:nirdeshika,ai:prathigrahika,odu:sumoyika,kkuz:udeshika,AI:prayogika,il:adharika,udei:sambandhika,female,singular,third, {human})

The verb padikkuka is stored as

The lexicon of modifiers contains information like the following.
(ethu, demonstrative),
(ventingal : 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. **Venthalingal** udichhu (The moon had risen)
3. **Orupara nellu** thannu. (One litre paddy was given)
4. **Nooru roopakku nallu pattiva** vangi. (Four dogs were bought for hundred rupees)
5. **Odunna vandikku** brake venum. (Running car needs brake)
6. **Vellutha pakshi parannupoi.** (The white bird flew away)
7. **Avan A paadam nannav 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.
<table>
<thead>
<tr>
<th>Verb</th>
<th>Root</th>
<th>Verb form</th>
<th>Tense</th>
<th>Person</th>
<th>Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kartha karakam</td>
<td>root:</td>
<td>Category</td>
<td>Modifier</td>
<td>Value</td>
<td>Category</td>
</tr>
<tr>
<td>Karma karakam</td>
<td>Root:</td>
<td>Category</td>
<td>Modifier</td>
<td>Value</td>
<td>Category</td>
</tr>
<tr>
<td>Sakshi karakam</td>
<td>Root:</td>
<td>Category</td>
<td>Modifier</td>
<td>Value</td>
<td>Category</td>
</tr>
<tr>
<td>Swami karakam</td>
<td>Root:</td>
<td>Category</td>
<td>Modifier</td>
<td>Value</td>
<td>Category</td>
</tr>
<tr>
<td>Hethu Karakam</td>
<td>Root:</td>
<td>Category</td>
<td>Modifier</td>
<td>Value</td>
<td>Category</td>
</tr>
<tr>
<td>Adhikarna karakam</td>
<td>Root:</td>
<td>Source</td>
<td>Destination</td>
<td>Location</td>
<td>Time</td>
</tr>
</tbody>
</table>

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. parayukanennu, kelkukkanennu)

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 markettilakkku 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](image-url)
S.2  Indiayila janangal narayananayi pradhanamanthriyai thiranjeduthu.

Fig. 3.4. Frame structure for sentence S.1

Fig. 3.5. Parse structure for sentence S.2
<table>
<thead>
<tr>
<th>Verb</th>
<th>Root</th>
<th>thiranjad</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Verb form</td>
<td>form-l</td>
</tr>
<tr>
<td></td>
<td>Tense</td>
<td>Past</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Plural</td>
</tr>
<tr>
<td></td>
<td>Person</td>
<td>3rd</td>
</tr>
<tr>
<td></td>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td>Kartha karakam</td>
<td>Root</td>
<td>Janam</td>
</tr>
<tr>
<td></td>
<td>Category</td>
<td>Noun</td>
</tr>
<tr>
<td></td>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>Plural</td>
</tr>
<tr>
<td></td>
<td>Gender</td>
<td>Alinga</td>
</tr>
<tr>
<td></td>
<td>Person</td>
<td>3rd</td>
</tr>
<tr>
<td>Karma karakam</td>
<td>Root</td>
<td>narayanan</td>
</tr>
<tr>
<td></td>
<td>Category</td>
<td>Noun</td>
</tr>
<tr>
<td></td>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td></td>
<td>Value :</td>
<td>prathanamanthri</td>
</tr>
<tr>
<td></td>
<td>category</td>
<td>complement</td>
</tr>
<tr>
<td></td>
<td>number</td>
<td>singular</td>
</tr>
<tr>
<td></td>
<td>gender</td>
<td>male</td>
</tr>
<tr>
<td></td>
<td>person</td>
<td>3rd</td>
</tr>
<tr>
<td>Adhikarna karakam</td>
<td>Root</td>
<td>India</td>
</tr>
<tr>
<td></td>
<td>Category</td>
<td>noun</td>
</tr>
<tr>
<td></td>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>singular</td>
</tr>
<tr>
<td></td>
<td>Gender</td>
<td>Neuter</td>
</tr>
<tr>
<td></td>
<td>Person</td>
<td>3rd</td>
</tr>
</tbody>
</table>

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
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 : 3rd
- Modifier
  - Value : vegam
  - Category : adverbial

Karthak karakam
- Root : kathakku
- Cateogory : noun
- Modifier
  - Value : E
  - Category : demonstrative
- Number : singular
- Gender : nueter
- Person : 3rd

Fig. 3.10. Frame structure of sentence S.4
S.5  
kurachhu ezhuthukal innu labichu.

Fig. 3.11. Parse structure of sentence S.5

Fig. 3.12. Frame structure of sentence S.5
S.6 nammal pavangalle sahayikkanam.

Fig. 3.13. Parse structure of sentence S.6

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Karthakarakam</td>
<td>Root : nammal</td>
<td>Category : pronoun</td>
<td>Modifier : NIL</td>
<td>Number : plural</td>
<td>Gender : alinga</td>
<td>Person : 1st</td>
</tr>
<tr>
<td>Karma karakam</td>
<td>Root : pavangall</td>
<td>Category : noun</td>
<td>Modifier : NIL</td>
<td>Number : plural</td>
<td>Gender : alinga</td>
<td>Person : 3rd</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>Subordiniate verb</th>
<th>Root</th>
<th>: Ke</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Verb form</td>
<td>: form-10</td>
</tr>
<tr>
<td></td>
<td>Tense</td>
<td>: past tense</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>: singular</td>
</tr>
<tr>
<td></td>
<td>Person</td>
<td>: 3rd</td>
</tr>
<tr>
<td>Modifier</td>
<td>: NIL</td>
<td></td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>Main verb</th>
<th>Root</th>
<th>para</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Verb form</td>
<td>form-l</td>
</tr>
<tr>
<td></td>
<td>Tense</td>
<td>past tense</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>singular</td>
</tr>
<tr>
<td></td>
<td>Person</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td>Kartha karakam</td>
<td>Root</td>
<td>teacher</td>
</tr>
<tr>
<td></td>
<td>Category</td>
<td>noun</td>
</tr>
<tr>
<td></td>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>singular</td>
</tr>
<tr>
<td></td>
<td>Gender</td>
<td>alinga</td>
</tr>
<tr>
<td></td>
<td>Person</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt;</td>
</tr>
<tr>
<td>Sakshi karakam</td>
<td>Root</td>
<td>kutti</td>
</tr>
<tr>
<td></td>
<td>Category</td>
<td>noun</td>
</tr>
<tr>
<td></td>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>singular</td>
</tr>
<tr>
<td></td>
<td>Gender</td>
<td>alinga</td>
</tr>
<tr>
<td></td>
<td>Person</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt;</td>
</tr>
<tr>
<td>Subordinate verb</td>
<td>Root</td>
<td>vayi</td>
</tr>
<tr>
<td></td>
<td>Verb form</td>
<td>form-9</td>
</tr>
<tr>
<td></td>
<td>Tense</td>
<td>future</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>singular</td>
</tr>
<tr>
<td></td>
<td>Person</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt;</td>
</tr>
<tr>
<td>Kartha karakam</td>
<td>Root</td>
<td>kutti</td>
</tr>
<tr>
<td></td>
<td>Category</td>
<td>noun</td>
</tr>
<tr>
<td></td>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>singular</td>
</tr>
<tr>
<td></td>
<td>Gender</td>
<td>alinga</td>
</tr>
<tr>
<td></td>
<td>Person</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt;</td>
</tr>
<tr>
<td>Karma karakam</td>
<td>Root</td>
<td>pusthakam</td>
</tr>
<tr>
<td></td>
<td>Category</td>
<td>noun</td>
</tr>
<tr>
<td></td>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>singular</td>
</tr>
<tr>
<td></td>
<td>Gender</td>
<td>neuter</td>
</tr>
<tr>
<td></td>
<td>Person</td>
<td>3&lt;sup&gt;rd&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

Fig. 3.18. Frame structure of the complex sentence C.2
C.3 Raman kathakku thurannappol poocha chadi.

Fig. 3.19. Parse structure for complex sentence C.3
<table>
<thead>
<tr>
<th>Main clause: verb</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>: chadi</td>
<td></td>
</tr>
<tr>
<td>Verb form</td>
<td>: form-l</td>
<td></td>
</tr>
<tr>
<td>Tense</td>
<td>: past tense</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>: singular</td>
<td></td>
</tr>
<tr>
<td>Person</td>
<td>: 3rd</td>
<td></td>
</tr>
<tr>
<td>Modifier</td>
<td>: NIL</td>
<td></td>
</tr>
</tbody>
</table>

| Kartha karakam    |  |  |
| Root              | : poocha    |  |
| Category          | : noun      |  |
| Modifier          | : NIL       |  |
| Number            | : singular  |  |
| Gender            | : neuter    |  |
| Person            | : 3rd       |  |

| Subordinate clause|  |  |
| Verb root         | : thura     |  |
| Verb form         | : form-12   |  |
| Tense             | : past tense|  |
| Number            | : singular  |  |
| Person            | : 3rd       |  |
| Modifier          | : NIL       |  |

| Kartha karakam    |  |  |
| Root              | : raman     |  |
| Category          | : noun      |  |
| Modifier          | : NIL       |  |
| Gender            | : Male      |  |
| Person            | : 3rd       |  |

| Karma karakam     |  |  |
| Root              | : kathakku  |  |
| Category          | : noun      |  |
| Modifier          | : NIL       |  |
| Number            | : singular  |  |
| Gender            | : neuter    |  |
| Person            | : 3rd       |  |

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
<table>
<thead>
<tr>
<th>Main verb</th>
<th>Root</th>
<th>para</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verb form</td>
<td>form-I</td>
<td></td>
</tr>
<tr>
<td>Tense</td>
<td>past tense</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>singular</td>
<td></td>
</tr>
<tr>
<td>Person</td>
<td>3rd</td>
<td></td>
</tr>
</tbody>
</table>

Kartha karakam

<table>
<thead>
<tr>
<th>Root</th>
<th>amma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>noun</td>
</tr>
<tr>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td>Number</td>
<td>singular</td>
</tr>
<tr>
<td>Person</td>
<td>3rd</td>
</tr>
<tr>
<td>Gender</td>
<td>female</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subordinate verb</th>
<th>Verb</th>
<th>para</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verb form</td>
<td>form-13</td>
<td></td>
</tr>
<tr>
<td>Tense</td>
<td>present tense</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>singular</td>
<td></td>
</tr>
<tr>
<td>Person</td>
<td>3rd</td>
<td></td>
</tr>
<tr>
<td>Modifier</td>
<td>NIL</td>
<td></td>
</tr>
</tbody>
</table>

Kartha karakam

<table>
<thead>
<tr>
<th>Root</th>
<th>kutti</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>noun</td>
</tr>
<tr>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td>Gender</td>
<td>neuter</td>
</tr>
<tr>
<td>Person</td>
<td>3rd</td>
</tr>
</tbody>
</table>

Karma karakam

<table>
<thead>
<tr>
<th>Root</th>
<th>pattu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>noun</td>
</tr>
<tr>
<td>Modifier</td>
<td>NIL</td>
</tr>
<tr>
<td>Number</td>
<td>singular</td>
</tr>
<tr>
<td>Gender</td>
<td>neuter</td>
</tr>
<tr>
<td>Person</td>
<td>3rd</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>Part of Speech</th>
<th>Details</th>
</tr>
</thead>
</table>
| 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**: katti  
**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 aaghrahichhu

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 edithathu njan kandu
12. Subject + verb + that + clause
   He admitted that he had written a letter
   Oru ezhuthhu 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
   Katakkku thurannappol pakshi parannupoi

15. Subject + verb + object + when – clause
   Father bought a shirt when he came yesterday
   Acchhan ennalai vannappol oru shirt konuvannu

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 proposition used in the target sentence are by or with. Since the ahikarana karakam refers to the position in time or space, the prepostions 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 “Indiayudai 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.
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
Karthaka 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.conscode = cons.conscode
(win, cand) ::- win.candcode = cand.candcode
Thus the SQL command for the query S1 is

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

Consider another query.

S2. Ettavum kuduthal vote labichha sthanarthi aare? (Candidate with the highest vote?)

The frame containing the meaning of the query is

Verb : labichha
Karthi 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
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 congresssintta sthanarthikkal aarelam? Marxistinttayo? (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: Sthanarthikal
Karma karakam
Value: Kottayam
Attribute: Jilla
Relation/quantifier: =

Value: Marxist
Attribute: party
Relation: =

This structure is equivalent to that of the query \textit{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

<table>
<thead>
<tr>
<th>Original query with deep level ellipsis</th>
<th>System rephrased query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aluvaude MLA aare?</td>
<td>Aluva mandalathil ettavum kuduthal vote labhichha sthanarthis aare?</td>
</tr>
<tr>
<td>Kollamkkar aarellam</td>
<td>Kollam mandalathila sthanarthikkal aarellam</td>
</tr>
<tr>
<td>Bjpude sthanarthikkal aarellam</td>
<td>BJP partyudai sthanarthikkal aarellam</td>
</tr>
</tbody>
</table>

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 pustakam ezhuthiyathu aare ?
   Athinte vila enthu ?
2. Computergraphicsenna pustakam 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 pustakathinnta 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 sometimes 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)
Entrolls (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.

<table>
<thead>
<tr>
<th>E - Relation</th>
<th>Final-Exam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Season</td>
<td>Date</td>
</tr>
<tr>
<td>Spring</td>
<td>20/4/96</td>
</tr>
<tr>
<td>Summer</td>
<td>22/8/96</td>
</tr>
<tr>
<td>Fall</td>
<td>10/12/96</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>SUP</th>
<th>SUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final-Exam</td>
<td>Enrolls.mark</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Concept</th>
<th>EXC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course.text</td>
<td>Thesis-course</td>
</tr>
</tbody>
</table>

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

```sql
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.

<table>
<thead>
<tr>
<th>V_Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thesis-course</td>
</tr>
</tbody>
</table>

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

```sql
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".