DATABASE ACCESS IN TELUGU

Thesis submitted to
The University of Hyderabad
for the award of the degree of

Doctor of Philosophy

by
C. Ravi Shankar

Department of Computer and Information Sciences
School of Mathematics and Computer/Information Sciences
University of Hyderabad
Hyderabad-500134
India

July 1993

CERTIFICATE

This is to certify that I, C. Ravi Shankar, have carried out the research work embodied in the present thesis under the guidance of Dr. A. Sivasankara Reddy for the full period prescribed under the Ph. D ordinances of the University.

I declare to the best of **my** knowledge **that** no **part of this thesis was** earlier submitted for the award of any degree **of** any university.

(Dr. A. Sivasankara Reddy)
SUPERVISOR

(C. Ravi Shankar)
Candidate

HEAD
Department of Computer and Information Sciences

Prof. V. Kannan
DEAN
School of Mathematics and Computer/Information Sciences

ACKNOWLEDGMENT

I am indebted to my supervisor, Dr. A. Sivasankara Reddy, for his help, guidance and encouragement throughout the period of my research. But for his friendly supervision, this thesis would not have seen the light of the day. I always enjoyed the long discussions I had with him on several topics ranging from linguistics to philosophy. Thank you, ASR, for all your patience.

The role of my father in initiating and of my wife in sustaining this research can neither be quantified nor adequately acknowledged. But for their constant help and encouragement, 1 would not have embarked on this arduous journey to those regions of Telugu computational linguistics where no one had gone before. Thank you for putting up with my negligence towards family commitments.

Dr Ajeet Parhar's help in proof reading the various versions of this thesis and in making it more readable are highly appreciated. But for his help, some of the assumptions I had left unsaid would have remained so. The help rendered by Prof. C.A. Sastry, and Dr.Doug Moore in making the thesis more presentable are highly appreciated. The help rendered by Mr. Kanada and Miss. Bharati in making me appreciate the intricacies of Telugu morphology can not be forgotten. Thank you all.

The excellent working environment and the enthusiastic people of the VG group at CMC R&D nurtured my research in its infancy. The linguistic environment at La Trobe University helped the later stages of this research. I would like to thank CMC Ltd., and La Trobe university for all the support they gave me during this research.

The sentences presented in this thesis in Telugu lipi (script) owe their existence to the Rachana word processor of Mr. K.L. Vyas who so graciously give me the package for use. He has also immensely helped me in collecting the Telugu textual corpus. Thank you Vyas for all your help.

ABSTRACT

Sage Panini belonging to the traditional linguistic school of India, called Vyakarana, propounded a theory of sentence formation centred around the ideas of *aakaamksha* (expectancy), *yogyata* (ability), *sannidhi* (proximity) and *samarthah* (equi-meaning). These theories are highly lexical in nature. In this thesis, these concepts are formalised and presented in a form suitable for representing and using on a computer. This formalization provides the basis for developing a parsing technique for Telugu, called Functional Application Parsing (FAP). FAP is characterized by its heavy reliance on lexical information expressed through structure sharing data structures and use of only a few functional application rules for parsing Telugu sentences. The knowledge representation used to specify the lexical entries and the nature of information to be encoded in the lexicon are presented in detail.

A Telugu Language Analyser (TELANGANA) program was developed based on the above mentioned traditional linguistic ideas. TELANGANA consists of four modules, a morphological analyser (MA), a syntactic and semantic analyser which espouses FAP for analysing Telugu sentences, a quantifier scoper and a question answering module. MA uses declarative morphological rules to analyse Telugu suffixation and compound word formation. A novel feature of the morphological rules is their ability to specify linear precedence constraints on suffixes. In addition, these rules can also work with or without a lexicon. By using a novel bootstrapping method, and the rules in a "no-lexicon" mode, it was possible to generate a lexicon for Telugu automatically.

The syntactic and semantic analyser uses the FAP technique for parsing and building the semantic structures corresponding to different sentential constructs of Telugu, such as copula sentences, simple sentences with one verb, and complex sentences consisting of relative clauses, comparative clauses and subordinate clauses. In the context of describing the analysis of complex Telugu sentences and anaphora, several innovations in handling the syntactic nuances of Telugu are presented.

A logical form (LF) for specifying the model theoretic semantics of Telugu sentences is presented. A novel quantifier scoping algorithm for assigning proper scopes to Telugu quantifiers, in the context of the above LF, is presented. TELANGANA uses scoped LFs to form database specific queries from user's Telugu queries.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 General

vaagarthaaviva sampruktau vaagartha pratipattaye[1]
jagatah pitarau vamde paarvatee parameswarau

Kalidasa[2], the well known Sanskrit poet, started his magnum-opus, *Raghuvamsam*, with the above invocatory verse. The above verse means, "For my words to attain their full purport, I pay my salutations to the parents of the world, Paarvatee and Parameswara, who are **inseparable like a word and its meaning".** This verse in a nutshell brings out the popular view that was held by people in olden times in India regarding the relationship between a word, and its meaning. The terms 'word' and 'meaning' can be interpreted to mean "sequence of words" (syntax) and 'semantics' (including pragmatics) respectively. This belief regarding the inseparability between syntax and semantics is largely due to the traditional schools of Indian Philosophy (Mimamsa, Nyaya, Sankhya, Yoga, Vaiseshika and Vyakarana)[3] which so often debated and theorized various aspects of language. Their views slowly seeped into literature as exemplified in Kalidasa's works.

Panini[4] is the most well known grammarian of ancient times in India. He was the originator of the Vyakarana (Grammar) school of philosophy. He analysed Vedic[5] Sanskrit and codified all the then prevailing linguistic knowledge into a book called *Ashtaadhyaayi* (AST)[6]. Since then, theories on various aspects of language like phonology, syntax, semantics, rhetorics, and pragmatics have been thoroughly studied by various scholars of

[1]In the transliteration of Telugu and Sanskrit words into English, the notational conventions of English capitalization like using a capital letter in the beginning of new sentences and proper names is not respected in this thesis, because capital letters convey special significance for transliteration as shown in appendix 1.
[2]Bom around 200AD.
[3]Called shatdrashanas, the six paths (views) to reality.
[4]Born around 600BC.
[5]Hindu religious scriptures.
[6]Since the book contained eight chapters, it was called Ashtaadhyaayi (ashta= eight, adhyaayi= containing chapters).

Sanskrit grammar. The analysis of language, as lime progressed, was further extended to non-Vedic texts as well. Grammar was a school of philosophy, that especially concentrated on the study of language both as a means and as an end, achieving the ultimate goal of life, in itself. The other schools of philosophy like Mimamsa, Nyaya, and Vedanta also made their contribution towards the analysis of language, from their own view points and perspectives. To the philosophers of these schools, however, the study of language was a means to an end which was different from the study of language. Thus language came to be studied from various points of view with diverging intentions. These various schools of philosophy devoted much thought to the problems of linguistic philosophy and general linguistics and evolved different theories to explain the manifold aspects of language. The discussions on the problems of language and its meaning by Indian thinkers like Panini, Patanjali, Jaimini, Bhartrihari, Kumarilabhatta, and Anandavardhana [Maha84] show extraordinary linguistic and philosophic acumen. The observations made by them could be of considerable help to people trying to work in the area of natural language processing (NLP) for Indian languages as well as for other languages.

Using insights from the above Indian theories and modern computational linguistics, a computer program, called TELANGANA (Telugu Language Analyser), was developed by the author to 'understand' Telugu sentences. TELANGANA and the theory underlying its construction are presented in this thesis. Telugu is a language spoken in south India, Andhra Pradesh, by over 80 million people. Telugu is believed to have its origins in Sanskrit and Dravidian languages. The theories developed in Sanskrit appear to be very well suited to the study of Telugu. To the best knowledge of the author, there is not much literature on relating Sanskrit grammatical principles to the study of Telugu syntax and semantics. In addition, the classical Telugu grammar treatises do not deal very much with the rules of sentence formation. The study of Telugu language is also interesting as Telugu syntax does not follow Sanskrit grammar, while following the general linguistic philosophy of Sanskrit.

As this thesis owes it's inspiration to some of the ideas propounded by the earlier Sanskrit grammarians, it is necessary to bring out, at least briefly, those ideas and views. It is also necessary to explain how these theories have been cast into computational terms, suitable for developing a language 'understanding' program. The meaning of the word 'understanding' is different to different people and is difficult to define. In the broadest sense, a computer may be said to 'understand' a sentence, if it can answer some questions regarding the content of the sentence or act as directed or implied in the sentence. Depending on the complexity of the questions and the appropriateness of the answers, the degree of understanding may be

gauged. Thus it can be seen that the meaning of the word 'understanding' given above is highly operational and not denotational. In this thesis, the word 'understand' is used in a more narrow sense by delimiting the domain of application. A computer is said to 'understand' a sentence if it can query or update a database as requested in the sentence.

A common man who occasionally uses a database, that too at some service point, may never want to learn the intricacies of the database access languages and would like to interact with the machine in a natural language known to him. There are three reasons, as described below, for choosing database access as the end application.

1.1.1 Potential for usage

Firstly, the advent of electronic publishing and database technology has resulted in an unprecedented and ever growing availability of text and data in machine readable form. A stage has come, wherein non-computer professional users have out numbered computer professionals in the usage of computers. Simplifying the usage of computers would go a long way in improving their productivity. The majority of computer applications are in the area of database access. Hence from the usage point of view, database access is a very good candidate for testing TELANGANA.

1.1.2 Novelty of Research

Secondly, the idea of querying databases in Indian language has not so far attracted much interest. No information of this nature has been published or available to the best knowledge of the author. Currently all the efforts in the area of Natural Language Processing for Indian languages are geared towards machine translation only [Alwa88, Bhar90, Sinh88j and very little effort is made in the other allied areas like textual information retrieval, database querying, automatic Indian language proof reading, spell checking etc. Published work on database access through Indian Languages is practically nil. Out of the above few possible application areas, database access is more demanding with respect to syntax and semantics of a language. This makes database access an ideal subject for this study.

1.1.3 Tractability of Application

Thirdly, restricting the domain of application simplifies many of the thorny knowledge representational issues associated with unrestricted natural language sentences. Sentences involving continuous actions (like water flowing into a room), tense and other aspects of verbs (The sun rises in the east) are quite easy to syntactically parse, but very difficult to

represent semantically in order to answer questions about them. There are many aspects [Moor81] of representation in relation to these phenomenon that are not adequately understood by logicians and philosophers. Hence one likes to have a domain of application which allows for rich syntactic variations, without a heavy con-commitment on the semantics front. Querying relational databases is such a domain of application. As relational databases encode certain kinds of knowledge only (snap shots of states or interrelations among entities), the linguistic phenomenon to 'talk' about such knowledge also gets limited. This limit, even though from a technical point of view lessens the scope of investigation, seems to be the only practical way to make useable models of the language. At present, any method trying to understand any natural language in all its colours and hews is likely to fail, as so little is known about languages. However, in order to impart some generality to the investigations, an attempt was made in this study to analyse and represent Telugu sentences independent of any specific database or its contents to the extent possible.

In this thesis an attempt is made to 'naturalize' the way Telugu speaking people can interact with databases (DB). Natural language, ie. Telugu, queries are allowed to be posed to the DB instead of the stylised DB query language queries. The program, TELANGANA, provides the necessary interface between the user and the DB. TELANGANA understands the user's queries in Telugu, converts them to appropriate machine oriented queries, like SQL, QUEL, or Prolog, presents these queries to the DB and finally forwards the returned answers to the user.

The work presented here can be viewed as an attempt to operationalize[7] the old theories of Sanskrit grammar. Most of this operationalization would not have been possible but for the existence of the modern linguistic theories. Some of the mathematical linguistic formalisms were borrowed from GPSG (Generalized Phrase Structure Grammar) [Gazd85], HPSG (Head-driven Phrase Structure Grammar) [Poll85] and Categorial Grammars [Vanb88] to give a formal descriptive body to the non-descriptive Sanskrit grammar sentence formation theories.

In this age of large scale computerization of even non-mathematical ideas, there are many advantages to Operationalizing any theory. Linguistics in general, and Indian language oriented linguistics in particular, can also benefit from operationalization. The foremost advantage is to the linguists themselves, as it has the great potential for improving their own research facilities. Most of the linguistic principles are studied using a small set of examples

[7]This word is used through out this thesis to mean, making a theory useable on a computer by spelling out all the assumptions and details.

(Sanskrit linguistics is no exception to this) . That is why one comes across the same examples time and again while going through various papers and books on the subject. The resilience of the theories with respect to a wider usage of language can be tested only with large corpora of sentences. Once the size of these corpora grows beyond a certain stage, it becomes necessary that some automatic means be devised to categorize and retrieve valid instances of sentences with respect to certain criteria. For example, if one needs to look up all the sentences in the corpus such that the pronoun 'his' in the sentence is used to refer to some universally quantified word in the sentence, as in 'Every person in the conference gave his visiting card to Rama', one really needs highly sophisticated automatic methods. These methods can only be developed if linguistic theories are operationalized and can be tested on computers. To operationalize a theory one needs to fill in all the descriptive elements of the theory. This makes the theory more easily understood, leaving nothing to individual assumptions. Once the theory is operationalized, it can be put to many good public uses, like accessing databases in natural language at common places like railway stations, banks etc. Thus the author's effort of computerizing Indian Linguistics is well motivated.

As the concepts dealt with in the traditional Sanskrit linguistic theories are so exhaustive, it would be difficult to summarize them in one chapter. In this introductory chapter, outlines of various theories of syntax and semantics put forth by the Vyakarana and other schools of philosophies are presented very briefly. As the design of TELANGANA is based on some of these theories, a good understanding of those concepts would go a long way in understanding this dissertation. Certain concepts introduced in this chapter (section 1.3 to 1.6) would be referred to in the subsequent chapters often. The ideas borrowed from these theories for developing the program TELANGANA will also be highlighted in the this chapter and the subsequent chapters as and when appropriate.

1. 2 Two Approaches to the study of language

The various schools of thought, Vyakarana, Mimamsa and Nyaya, have essentially taken two approaches to the explanation of linguistic phenomenon. One view is known as *akhandapaksha* (wholistic or Gestalt view) and the other is known as *khandapaksha* (segmentary or analytic view). The two views are totally at odds with one another and are interesting for their novelty and diversity. According to the followers of *akhandapaksha*, the meaning of a sentence is an indivisible atomic entity. The meaning of a sentence is not computable as a combination of the meanings of its constituent words. Further, this view purports that words by themselves do not have any meaning bearing capability. They get

such a capability only in the context of a sentence. Grammarians like Bhartrihari, and Audumbarayana advocated this view.

Followers of the second view, *khandapaksha,* held that individual words are real entities and are associated with some meaning of their own. In other words, the meaning of the whole is equal to the sum of the parts. This view is similar to the present day view of the major linguistic theories like GB-theory [Chom81], LFG [Bres84], and GPSG [Gazd85]. According to this analytical view of sentence comprehension, words are considered to be autonomous units of thought and language. A sentence is formed by the concatenation of a set of words constrained by certain well formedness criteria, and the meaning of the sentence is derived from the meaning of its constituent words. Grammarians like Panini, Patanjali, Kaatyaayana belonged to this school. In the next section these two views are further explored as they shed light on the various possible approaches for analysing sentences using Sanskrit grammatical theories.

1.2.1 Akhandapaksha (Gestalt View)

The main tenet of *akhandapaksha* is that the smallest entity for analysis is a sentence and words are not of much relevance. The particular words and their order of utterance, is just for forming a mental signature (encryption) of the sentence, which arouses in the mind of the hearer a Gestalt (overall) perception of the meaning of the sentence instantaneously. This view was held by Bhartrihari, Audumbarayana and other philosophers advocating the *sphotasidhaanta* (Gestalt theory) [Bhar66]. They held that a sentence as a whole is regularly present in the perceptive faculty of the hearer and the multifold classification of words into nouns, verbs etc does not help in analysing the import of sentences. They saw a sentence as being a single symbol *(eko na avayavah sabdah)*[8] formed out of letters, and words. The meaning conveyed by this sentence is considered an integral symbol (called *vaakya-sphota).* The meaning is conveyed as an instantaneous flash of insight or intuition *(pratibha).* The meaning is thus partless. The words have no reality of their own and are only signatures of the intuition.

This very highly abstract theory was elaborately developed by the philosopher poet Bhartrihari [Bhar66] in the book *vaakyapadiyam.* Bhartrihari's analysis envisages three aspects of the sentence cognition. The first is *vaikrita-dhvani* (explicit sound), which is an instance of the second, *prakrita-dhvani* (internal/natural sound). *Vaikrita-dhvani,* the individual specific utterance in purely phonetic terms is the one that is spoken by the

[8]An utterence is atomic and does not have parts.

speaker, and heard by the hearer. This includes all the specific aspects of the utterance namely, intonation, pitch, tempo, accent and so forth. The normalised, speaker and situation independent phonological structure, *prakrita-dhvani,* forms the other aspect of the sentence. At this stage all the personal and situation dependant information is lost in the utterance. The utterance resides in the memory of the listener in its proper time-sequence. The third aspect is the *sphota,* wherein the mental signature is present, cannot be pronounced or written. There is no *sphota* without meaning. It is the meaning-bearing nature of an expression that makes it a *sphota.*

The process of comprehending *sphota* is illustrated by grammarians by means of various analogies. A student trying to comprehend a verse by repeatedly reading presents an example. The last reading, makes him suddenly understand the verse unambiguously. Till the last reading, the sentence is present as a *prakrita-dhvani* and once the meaning is understood it becomes *sphota.* It is the cognition of the whole that is significant and thus important. The whole, taken as a mental symbol is different from the parts that constitute it. Therefore parts may be considered irrelevant or illusory. It is not to say that parts do not exist, but in the gestalt perception the parts are lost. This phenomenon of losing the parts can be seen in one's inability to reproduce exactly what one heard from a friend even though one was able to remember the content of the message. According to the protagonists of *akhandapaksha,* the function of the individual letters and words, is based on their capacity to differentiate one word from the other, and one sentence from the other. That is, the function of the letters and words, is to form unique signatures of the meaning of the sentence.

The following example is given to illustrate this abstruse point further by Seshakrishna [Seshl3] in *Sphota-tattvanirupana.* He says that when a person utters the sound *ka* with the intention to say *kamalam* (a lotus), the uttered syllable *ka* gives a cue to the word to be spoken in as much as it negates *non-ka* beginning words. The next syllable *ma* further narrows the possibilities. One is left to guess, whether the word is going to be *kamuliyah* or *kamaniyam.* Once the final syllable *lam* is uttered, the word is known fully and unambiguously. Thus the function of the letters is to build up the higher unit; the letters do not have their own meaning. Their value consists in differentiating one word from the other. Words and sentence are related similarly. Buddhists also had a theory close to this, called *apoha.*

The theory of *sphota,* explained above, being totally orthogonal to the current computational linguistic work, may appear to be intangible for computerization. But recent work on memory based translation [Kita91], and other neural network based natural language

processing systems [Berg92, Jain89, Stan86] embody this principle to some extent. It would be very interesting to delve deeper into this unexplored area and see if any new insights or directions could be found for cognitive style natural language processing. One great advantage in following this theory is that elliptical sentences and normal syntactically well formed sentences can be treated uniformly. As is well known, ellipses pose many theoretical and implementation problems in other analytical theories.

Interesting as it may be, the *akhandapaksha* does have its problems. By adopting the *akhandapaksha* view, one is left with no handles to answer "why type questions" like, Why is a certain a sentence incorrect? and, Why should the pronoun in a particular sentence refer to a certain object in the sentence and not to some other object? It can not answer questions like, what is the relationship between a causative sentence and a normal sentence?, and How can one causativize a sentence? In addition, the exposition of *akhandapaksha* in traditional linguistic literature [Bhar66] is highly philosophical and meta-physics oriented. This makes it difficult to use with confidence (with respect to the accuracy of interpretation). Further, it is too much geared towards the pragmatic and semantic aspects of language and ignores the syntactic aspects altogether.

In this thesis the second view of analysis, *khandapaksha*, is adopted. Following this view, one can gain deeper linguistic insights into the syntax and semantics of language. This approach gives equal importance to all the aspects of language and hence can answer many of the "why" and "how" type questions alluded to above. Also, the details of *khandapaksha* are very well worked out in AST, and hence one can use this approach with far greater confidence of correct interpretation.

1.2.2 Khandapaksha (Analytic View)

The *khandapaksha* view which takes an analytical view of sentences, is closer to the majority of the theories developed by the computational research work going on world wide. The linguistic study of the followers of *khandapaksha* centred around

- words,
- word meaning,
- the relationship between the word and the sentence, and
- the relationship between the word meaning and the sentence meaning.

They were mainly concerned with correct forms of words, even though they indicated sentence analysis as a top down process, i.e. starting from the sentence to be carried downwards through the sub phrases and words to the roots, stems and suffixes.

The most important contribution of this view of language understanding is the notion of *aakaamksha* (desire or mutual expectancy). The Mimamsa school enunciated the principle of *aakaamksha* to explain how syntactic and semantic unity is brought about among the various words that constitute a sentence, from the analytical and associative perspective. This concept was further modified by other schools. Two other notions, *yogyata* (ability) and *sannidhi/aasatti* (proximity), were added. The normal condition for *Saabdabodha* (the knowledge of the meaning of a sentence) is that the constituent words must be related to one another through *aakaamksha*, *yogyata* and *sannidhi*. These concepts saw their crowning in Panini's Ashtaadhyaayi [Josh68], wherein the rule *'samarthah padavidhih'* was given the status of a metarule that is applicable to the entire plethora of *padavidhih* (sentence formation from words, or syntax). As *aakaamksha, yogyata, sannidhi* and *samarthah* are central to the theory of sentence analysis in *khandapaksha*, they will be further explored in the ensuing sections. Out of these four, as the idea of *aakaamksha* is pivotal to all the Indian linguistic theories and to this thesis, it will be dealt first.

1.3 Aakaamksha (Expectancy)

Aakaamksha literally means 'desire or mutual expectancy'. It can be understood as the desire on the part of the LISTENER of a word W to hear a few more words, to comprehend the full sense/meaning of W. A word is said to have *aakaamksha* for another, if it cannot, without the later, produce knowledge of its inter-connection in an utterance. For example a verb like see has an expectancy for the object seen and the seer. Without specifying the seer and the object seen, the word 'see' does not convey its full meaning. To summarize, *aakaamksha* manifests in a word as the inability of the word to convey its complete meaning in the absence of another word.

For example in Sanskrit in the sentence,

రామః హరిం పశ్యతి

raamah harim pasyati
Rama sees Hari

the verb *pasyati* (sees) alone does not convey the meaning of the sentence. Similarly, the other two words also do not convey the full meaning of the sentence. Any combination of any two of the above three words, also does not form a sentence because a complete meaning is not formed. However, the above three words grouped in any order

rAmah pasyati harim
rAmah harim pasyati
harim rAmah pasyati,
harim pasyati rAmah
pasyati rAmah harim
pasyati harim rAmah

form a sentence in Sanskrit[9], as the resulting combination has no more expectancy for any other words to convey the full meaning of the sentence. This does not mean that no more words can be added to the sentence[10], but that the sentence is essentially formed by the above combination itself. A string of words such as ' cow dog go laugh' do not, however, form a sentence as there is no *aakaamksha* amongst the words.

A concept very closely related to the concept of *aakaamksha* is *uthita-aakaamksha* (aroused or potential expectancy). This concept was forwarded by the *advaitic* school of philosophy. There exists *aakaamksha* between words not only when one word expects the other but also when there is a possibility of expectation. For example in the sentence, "Read the book", the particular book is not specified. The word book may possibly imply adjectives like green, small or old, or complements like on the table, next to the oven or between the magazine and the cup. There is no limit to the possibilities of such potential expectancies. The word book has *uthita-aakaamksha* for the adjectives indicating qualities/attributes like green and small. On the other hand, adjectives have actual *aakaamksha* for the word that indicates the substance possessing those qualities. Similarly, verbs have *uthita-aakaamksha* for adverbs and adjuncts. The notion of *uthita-aakaamksha,* thus, brings adjectives, adverbs and adjuncts into the realm of *aakaamksha.*

According to the Mimamsa school a sentence is viewed as a group of words serving a single purpose, if on analysis the separate words are found to be wanting one another (mutual expectancy) to fulfil the total meaning of the sentence. When the sentences are independent of one another (each sentence having no requirement or expectation or *aakaamksha* of words

[9]All senetnees mean the same; Rama saw Hari.
[10]*rAmah udyAnE hariM pasyati* (Rama sees Hari in a garden)

outside itself to complete its meaning), they should be treated as different sentences. Thus *aakaamksha,* or mutual expectancy among the words is accepted as an essential condition lor sentence formation. A sentence, when complete, is *niraakaamksha* (without any more expectancy of words, opposite of *aakaamksha).* Hence *aakaamksha* exists not only amongst words, but also among partial sentences. By definition a complete sentence is *niraakaamksha* (devoid of expectancy).

The idea of *aakaamksha* as expounded in those Sanskrit theories, went further to include or imply an element of pragmatic completeness also. In the definition of *aakaamksha,* word listener was highlighted earlier. It is the listener who has an *aakaamksha* (desire) to hear the other words. The words do not per-se have *aakaamksha* (desire). When words arc said to have *aakaamksha,* it is said so in a figurative sense. This minor point becomes important when the intention, *taatparya,* has to be taken into account to understand a sentence. This brings into the realm of *aakaamksha* not only the syntactic completeness of a sentence but also the pragmatic completeness [Sear75] of the idea.

To this primary condition of *aakaamksha,* two more conditions (1) *yogyata,* meaning consistency of sense, and (2) *sannidhi,* meaning contiguity of words were added to form a self contained theory of sentence formation. These conditions were added to account for the presence of words which do not have *aakaamksha* for other words. When a word does not have any *aakaamksha,* even then it can be part of a sentence owing to its ability, *yogyata,* to satisfy some other word's *aakaamksha.* To satisfy that *aakaamksha,* the *yogyata* word should have proper *sannidhi* (proximity) to the *aakaamksha* word. Thus a word without any *aakaamksha* secures a place in a sentence. In this fashion two new notions, *yogyata* and *sannidhi,* were brought into the realm of sentence formation theories. These two notions are explained in the next two sections.

1.4 Yogyata (Ability)

Yogyata is the logical compatibility or consistency of all words in a sentence. It is judgemental in nature, when the meaning of a sentence is not contradicted by experience, there is *yogyata* or compatibility or consistency between words. For example in the sentence, ' the stone sang a nice song', the stone has no logical compatibility with singing, hence one rejects this sentence as ill-formed or meaningless. At this point it is worthwhile to mention that, it is necessary to distinguish between inconceivable combinations like 'bachelor's wife' or 'a circular square', both void by definition, and conceivable but unreal combinations like 'a

hare's horn'. In the later case, the incompatibility does not prevent sentence comprehension[11] but mitigates the validity of the knowledge gained. In the former case, it is the inconceivability of the mutual association of the word-meaning itself that renders the whole sentence or the word combination nonsensical.

In sentences like,

My car drinks gasoline

the lack of *yogyata* (ability) in the car to 'drink' may be explained by resorting to the metaphorical meaning of the word 'drink' in the sentence. In the above sentence the word 'drink' could be taken to mean 'consume'. If the incompatibility is thus removed and *yogyata* is understood, there is no difficulty in comprehending the meaning of the sentence. This function of the word, denoting a referent other than its normal and primary one, but in some way related to it, and where the speaker and the hearer are really aware of the distinction between the primary and the secondary referents, is called *lakshana* (secondary). There is an elaborate theory of *lakshana* in *Alamkara Sastra* [Anan40J which is not relevant to this discussion. It should be noted that the concept of *yogyata* is close to the concept of 'selectional restrictions' in modern computational linguistic literature [Wilk75, Boug79].

1.4.1 Karaka Theory (Case Theory)

It was believed by the ancient grammarians that a sentence, before it is uttered, is present in the mind of the speaker in some 'mental language' image *(Saabdabodha)* of its own. That image needs to be linearized when it is to be uttered or written down. When the image is being linearized, words have to be used to denote the parts of the mental image. While linearizing the sentence, proper suffixes need to be appended to words so that the inter-relationships between the words are clear.

The suffixes, also popularly known as case endings in English, are called *vibhaktis* in Sanskrit. The words appended by suitable *vibhaktis* can be written virtually in any order to make a sentence. The *karaka* theory expounded in Ashtaadhyaayi elaborates the suffixes *(vibhaktis)* that can be added to words and the effects of suffixation on the meaning of words. *Karaka* (deep case) theory is a fore runner of Fillmore's case theory [Fill68]. Unlike Fillmore's theory which dealt with only verb-noun relationships in sentences, *karaka* (deep case) theory encompasses the verb-verb relationships, verb-nonverb relationships, adjective-

[11] as in fairytales.

noun relationships, and noun-noun relationships. It was developed to explain the relationship between surface case *(vibhakti)* and deep cases *{karaka)* in a sentence. *Karakas* (deep cases) are infinite in number, whereas *vibhaktis* (surface cases) are only seven in number. Hence *vibhaktis* can imply multiple *karakas.* The *karaka,* which a *vibhakti* denotes, is a function of the *aakaamksha* between the governing verb and the *yogyata* of the noun to which the *vibhakti* is attached. Thus with a finite number of *vibhaktis* an infinite number of relationships can be expressed, just as in English with a finite number of prepositions an infinite number of relationships can be expressed. *Kaaraka* theory also indicates how the *yogyata* of a word is to be built from the root word and the suffix.

1.5 Sannidhi (Proximity)

The word *sannidhi* (and *aasatti)* means contiguity or proximity. *Sannidhi* is the other aspect of sequencing which has a bearing on the meaning of the sentence. Words uttered at long intervals or widely separated in writing with intermittent words, cannot produce the knowledge of any interrelation among them, even when *aakaamksha* and *yogyata* are present between them. The difference in meaning between the following two sentences of Telugu,

రాముడు చాలామంచి పుస్తకాలు కొన్నాడు

raamuDu caalaamaMci pustakaalu konnaaDu —1.1
Rama very good books bought
Rama bought very good books

రాముడు చాలా మంచిపుస్తకాలు కొన్నాడు

raamuDu caalaa maMcipustakaalu konnaaDu —1.2
Rama many good books bought
Rama bought many good books

is attributed to the difference in *sannidhi* between the words *caala, maMci* and *pustakaalu.* In the sentence 1.1, the words *caalaa* and *maMci* are pronounced quickly without much gap, hence they are written together. Whereas in 1.2, *maMci* and *pustakaalu* are pronounced together, and hence they are written together. The word *caalaa* means very or many depending upon context. The context is indicated by writing the words as compound words. In other words, *sannidhi* (proximity) is used to specify meaning. This kind of difference cannot be, however, brought out in English when writing. For example, in the sentence

Rama bought many more interesting books

there is no way to specify whether 'many' qualifies 'more' or 'books'. But in Sanskrit, Telugu and some other Indian languages this can be done easily, because words can be grouped together to indicate meaning related inter-relationships.

Basically *sannidhi* specifies the relationship between the sequence of words in a sentence or the temporal utterance of words that make up a sentence and its meaning. In other words *sannidhi* specifies the syntax of the language. Neither *aakaamksha* nor *yogyata* specify the physical order of the words in a sentence. This is because Sanskrit is essentially a free word order language. Hence the study of *sannidhi* (syntax) was not extensive in Sanskrit. However, in Telugu, *sannidhi* is more important as Telugu is not as free word ordered as Sanskrit. Consequently, the idea of *sannidhi* requires some minor modifications when applied to Telugu. Modelling *sannidhi,* however, is straight forward as detailed in Chapter 3.

1.6 Samarthah (Syntactico-Semantic Relatedness)

It may appear that words can be grouped arbitrarily to form compound words. This is not true. Compound word formation and sentence formation in turn are governed by a very concise and a difficult to formalize criteria called *samarthah.* While using words to form a sentence there must be *samarthah* amongst those words, otherwise the intended meaning will not be conveyed by the sentence.

The word *samarthah* literally means 'equi-meaning'. After specifying that the three conditions, *aakaamksha, yogyata,* and *sannidhi* are necessary for *Saabdabodha* (meaning comprehension), Panini tied down these conditions to word and sentence formation with his celebrated *paribhasha* (meta rule) *'samarthah pada vidhih'* (sentence formation is governed by *samarthah).* That means all word related operations like affixation, prefixation, compounding, relativization, and causitivization, can be done only if words are *samarthah.* This constraint was given by Panini as a meta-rule, because all other rules (a few hundred in number) given in *Ashtaadhyaayi* should be applied to words only if they are *samarthah.*

The notion of *samarthah* is central to this thesis. Hence considerable effort was spent in clearly understanding the meaning of the word *samarthah* as it is used in *Ashtaadhyaayi* and its commentaries. Mahavir [Maha84, pp 6-14] dealt with the meaning of *samarthah* at length, and concluded that it means "immediate syntactic relation of one word with another word in the same sentence". This definition of *samarthah* is not precise enough to be the basis of a formalization. For the sake of clarity, one could reword this definition to

"samarthah exists between two words W1 and W2, if they are related to one other directly through *aakaamksha, yogyata* and *sannidhi,* and form part of the same sentence". The above definition of *samarthah* given by Mahavir only captures one aspect of *samarthah,* the necessity for the existence of a relationship between the words, but does not say what happens if such a relationship exists. If the result accruing out of the relationship is not mentioned, the meta-rule *samarthah padavidhih* cannot be used for explaining sentence formation. Hence the full meaning of *samarthah* is more complicated than what Mahavir [Maha84] has proposed.

1.7 Technical interpretation of *samarthah*

After reviewing AST, Mahavir [Maha84, pp 18-19] observes that " ... it is not a mere syntactic relationship but an immediate syntactic relationship as an import of *samarthah,* which is more precisely applicable to all the data discussed earlier, and which only can explain the data given. ... *Samarthah* thus means an immediate relation of one word with another one in one and the same sentence which we call as 'immediate syntactic relation' ". In this thesis, it is proposed that *samarthah* be defined[12] as, "in a sentence, if a word X which has expectancy *{aakaamksha)* for a word Y, and Y occurs in the right word order (as dictated by *sannidhi),* and the interaction (functional application of one over the other, to be explained later on) of X and Y leads to some Z that is in ability *(yogyata)* or expectancy *(aakaamksha)* relation to the rest of the words in the sentence, then X and Y are said to be *samarthah".* This definition formalises the notion of "relation" used by Mahavir, and thus sharpens his definition. In addition, the proposed definition fills in the missing statement on the result accruing from the 'immediate syntactic relationship' between words and stales the means for computing the result of the immediate syntactic relationship. In this definition the result of the *samarthah* between X and Y is Z which could be a word, or a compound word or a sequence of words. The *yogyata/aakaamksha* of Z is derived by a functional application of X over Y. The main purpose of the grammar is to enumerate the properties of the derived entity Z, given the properties of X and Y. Accepting this notion of a grammar leads to a grammar which is more lexical than phrasal in nature. This also accounts for the learnability of Sanskrit by learning the lexicon.

The above definition of *samarthah* was used as the basis for developing the TELANGANA parser. This definition of *samarthah* can be justified by a closer study of the original text and commentaries of AST as was done by Mahavir [Maha84]. Mahavir took a more

[12]A complete definition is given in Chapter 5.

syntactic view of *samarthah* whereas in the thesis a Syntactico-Semantic view of *samarthah* was taken. A pure syntactic view can not account for the rich *aakaamksha/yogyata* relationships present a in natural language. This is made clearer in Chapter 3 and 5. Philosophically, the *samarthah* meta-rule says that Syntactico-Semantic relatedness more directly governs sentence formation than either purely syntax or semantic relatedness.

1.8 Basic sentence structure in Sanskrit

The entire emphasis of Ashtaadhyaayi, after propounding the *samarthah* concept and the *karaka* theory, shifts to word formation rules. There after, sentence formation is not touched upon. This approach is appropriate with respect to Sanskrit, because by and large Sanskrit is a free word order language, meaning, the words in a sentence either in prose or in poetry, can be used in any order e.g

రామః పాత్రే శ్యామాయ కూపాత్ జలం ఆనయతి

raamah paatre shyaamaaya kupaat jalam aanayati — 1.3
Rama pot-in shyaama-for well-from water-obj brings
Rama brings water for Shyam in a pot from a well

paatre raamah kupaat jalam aanayati shyaamaaya
aanayati raamah jalam shyaamaaya kupaat paatre
raamah shyaamaayapaatre jalam aanayati kupaat
. . . and so on

All the above sentences mean 'Rama brings water for Shyam in a pot from a well'. It is easy to see that there are 6! = 720 different word orderings possible for the above sentence, 1.3, in prose. Even adjectives, and the nouns they qualify, can be separated. There are a few well documented (in *Ashtaadhyaayi)* exceptions to this free word order. There are some restrictions on the usage of particles like *ca* (and), *tu* (but), *api* (also) etc. These have to be placed at the end of the words the sense of which they convey, *ca* (and) may come at the end or in between two words it is conjoining or at the end of the second word. The particle *na* (no) has no such restrictions. Interjections are used at the beginning of a sentence in general but some of them like *iti* (meaning in this way) are not so. Certain restrictions are placed on the forms of the *asmad* (I, mine) and *yushmad* (you, your) declensions. Excluding these few exceptions, Sanskrit is a free word order language. Hence in *Ashtaadhyaayi,* after stating the *samarthah* meta-rule, the focus shifts from sentence formation rules to word formation rules. Traditional Telugu grammarians like Cinnayyasuri [Cinn51], and Brown [Brow81]

also follow this trend in presenting Telugu grammar. Al this stage, it would be appropriate to see the relation between Sanskrit and Telugu.

1.9 Relationship between Sanskrit and Telugu

Due to the elaborate number of word formation rules in Sanskrit grammar, the belief that Sanskrit grammar is a word grammar rather than a sentence grammar has gained roots. This belief is subtly implied by the old grammarians of Telugu [Cinn51], who have tried to present Telugu grammar in Ashtaadhyaayi style, explaining exhaustively many word formation rules, without emphasising sentence formation rules[13]. Telugu is not so free word ordered as Sanskrit, and hence warrants more sentence formation rules. For example the Telugu sentence below corresponding to the sentence (1) of Sanskrit,

రాముడు శ్యాముకోసం బావినుంచి కుండలో నీళ్ళు తెచ్చాడు

rAmuDu shyaamukOsaM bAvinuMci kuMDalO neeLiu teccADu
Raama Shyaama-for well-from pot-in water-obj brought
Rama brought water for Shayam in a pot from a well

allows only about 96 variations in prose, inspite of many of them being only marginally acceptable. This is still very high compared to English, which would allow only 6 variations. Hence Telugu falls in between Sanskrit and English in Word order freedom. In view of this observation, Telugu grammar needs to be much more syntax oriented than Sanskrit and much less than English. However most of the traditional as well as modern Telugu grammarians have not spent much time on syntax related issues. This means that Telugu grammarians felt that traditional Sanskrit theories of *aakaamksha, yogyata, sannidhi,* and *samarthah* are adequate for explaining Telugu linguistic phenomenon.

1.10 Scope of the work

This observation has motivated the author to investigate the applicability of Sanskrit theories to Telugu. Accordingly, a detailed study of the Sanskrit linguistic literature was conducted. It was found that *samarthah* theory expounded in AST can be adopted easily to the study of Telugu even though it was mainly developed for the study of Sanskrit. The higher degree of syntax sensitivity of Telugu compared to Sanskrit does not diminish the applicability of *samarthah* theory to Telugu as shown in this thesis.

[13] Modern Telugu grammar books [Kris85] are tending to accord more respect to sentence formation rules.

In AST, the four ideas of *samarthah, aakaamksha, yogyata* and *sannidhi* have been described informally, as seen from the present day linguistic theories, in *sastrik* Sanskrit (a terse subset of spoken Sanskrit well understood by learned grammarians of those times). The idea of *samarthah* especially has been subjected to multiple interpretations [Maha84J. An attempt has been made in this study to formalise the above four ideas of AST.

Formalising *samarthah* theory and using this formalism to derive a parser for Telugu is a challenging task. As seen from the definition of *samarthah* and from the exposition of the Indian grammatical theories given in the earlier sections, it is easy to see that there is no notion of phrase structure rules in the grammar. Thus any one trying to follow the tradition of *Asthaadyaayi* is forced to develop a parsing scheme which does not rely on any notion of phrase structure. In tackling this task, the modern grammatical formalisms offer only a limited amount of help. By observing similarities and differences between *samarthah* theory and the modern grammatical theories one can develop a computationally viable technique for parsing Telugu sentences using *samarthah* theory.

The primary goal of this thesis, that of building a program to analyse and understand Telugu sentences, can be subdivided into the following four distinct objectives.

Objective 1: To formalise relevant parts of the old grammatical system of Sanskrit. The main difficulty in this task lies in sorting out the useful and useable grammatical principles from the informal sastrik Sanskrit texts and formulating those principles in modern parlance in a computationally useful form. Constructing appropriate data structures to capture the linguistic knowledge hidden in Ashtaadhyaayi also forms a part of this task.

Objective 2: To develop a parsing scheme for Telugu eschewing the traditional grammatical principles. It is clear from the exposition of the traditional grammatical theories made so far that they do not recognise the existence of phrase structure in Sanskrit or Telugu. Then the question is, how does one parse natural language sentences without using phrase structure rules? The parser is further constrained, by the author's choice, to follow the traditional grammatical principles and not to use ad hoc techniques. The parser so developed should be capable of parsing a wide variety of Telugu syntactic constructs in order to be credible. This would establish the generality and the utility of the approach.

Objective 3: To develop a declarative morphological analyser for Telugu that can be used, not only for the limited set of words that occur in database access, but also for the words that occur in arbitrary Telugu texts. The problem here is that Telugu is a highly inflection and suffixation oriented language allowing extensive compound word formations[14]. The maximum number of suffixes a word can take can be as large as 6. Morphological analysis in Telugu raises special problems that are not amenable to standard techniques put forward for morphological analysis of English and other related languages.

Objective 4: To develop an algorithm for the proper treatment of quantification in Telugu for database access. The assignment of scopes to embedded quantifiers is a problem idiosyncratic to every language. Telugu offers its own nuances in quantifier scoping.

From the formalization of the grammatical principles, a computationally viable method for parsing Telugu sentences has been developed. This method of parsing is called in this thesis "Functional Application Parsing". Using this method of parsing, the Telugu analysis program, TELANGANA, was developed which can parse and build the semantic representation for a variety of Telugu sentence structures consisting of, simple sentences, copula sentences (verb-less sentences), relative clauses, comparatives sentences (including clausal comparatives, adjectival and adverbial comparatives, superlatives), some amount of intra-sentential anaphora and limited conjunctions. TELANGANA parses all these varied types of sentences in a uniform and simple way.

1.11 Organization of the thesis

This thesis describes the program TELANGANA developed by the author to analyse Telugu sentences. To understand the theoretical basis of this program, one needs to understand both the traditional Sanskrit based grammatical theories and the modern computational and general linguistic theories. In this introductory chapter, the traditional theories were explained briefly. In the subsequent chapters, modern linguistic theories, Al techniques and details of TELANGANA are covered as follows.

Chapter 1, **Introduction,** this chapter, introduces the Indian grammatical theories. It also motivates the development of a program for database access in Telugu as a crucible for testing these theories. Then on, the main concepts of traditional linguistic theories such as

[14]Called sandhi in Sanskrit and Telugu.

aakaamksha, yogyata, karaka, sannidhi and *samarthah* are explained with suitable examples. The relationship between some of these ideas to modern linguistic ideas is alluded to where appropriate. Thus the basic ground for filling in the numerous details to make a working program out of the theory is laid in this chapter.

Chapter 2, **Approaches to Natural Language Processing,** critically reviews the literature consisting of the modern grammatical formalisms including GB theory, HPSG, CG, and the various artificial intelligence (Al) techniques that were developed for understanding natural language. Emphasis is placed on those formalisms and techniques that arc useful for processing the syntax and semantics of non-configurational languages.

Chapter 3, Knowledge representation **in TELANGANA,** initially deals with the overall organization of TELANGANA. Following this, the various stages of sentence processing are explained. The information passed between these stages is encoded in certain knowledge structures. These knowledge structures, and the knowledge representation techniques used to capture *aakaamksha, yogyata* and other related lexical knowledge are presented in this chapter in detail.

Chapter 4, Morphological Analyser, deals with the morphological analyser developed as part of TELANGANA. The problems in developing a morphological analyser for Telugu which motivated the creation of a new scheme for the morphological analysis of Telugu are brought out in this chapter. The morphological analyser developed can segment compound words and also has the capability to handle word morphology beyond the requirements imposed by database access. The rules and the algorithms that enabled this kind of extensive coverage are described. The automatic methods used to develop a lexicon for Telugu from a large textual corpus are also briefly described in this chapter.

Chapter 5, Syntactic **and** Semantic Analysis, explains in detail all the various syntactic constructs of Telugu and how TELANGANA parses them. Many examples are given to explain the functioning of the parser-cum-semantic analyser and to show the kind of linguistic coverage attempted in TELANGANA. The deep interaction between the semantics and the syntax of Telugu is brought out in this chapter. This chapter forms the foundation for the more complex analysis done in the next Chapter.

Chapter 6, **Syntactic and Semantic Analysis of Complex Sentences,** gives details of handling complex sentences with multiple verbs, relative clauses, sub-ordinate sentences, gap-filler constructions, and limited conjunctions. In this chapter the difficulties presented in

handling comparative sentences and the techniques developed in TELANGANA to overcome them are presented. Handling comparative sentences is a relatively neglected area in the NLP literature. The major differences between parsing in TELANGANA and other grammatical systems like HPSG, CG and GPSG are highlighted in this chapter.

Chapter 7, **Quantification and Question-Answering,** gives details on how the different quantifiers in Telugu like *anni, prati, aMta* are understood by TELANGANA. For database access an understanding of the interaction between these quantifiers is essential, as this mutual interaction between them leads to scoping problems. The details of assigning scopes to such embedded quantifiers are presented. Further on, some examples of sessions with TELANGANA are presented which show how TELANGANA actually accesses a database in Prolog and answers questions.

Chapter 8, **Conclusion and Future Work,** describes the contributions made by this thesis. Every natural language processing program covers a natural language to a limited extent only. Quite often the list of sentence types handled by a system is much smaller than the sentence types not handled by the system. The limitations of TELANAGA program are highlighted in this chapter. The possible future developments and areas for research are indicated .

Appendix A, **Telugu Transliteration,** gives the English equivalents of Telugu characters used in this thesis. Telugu alphabets consist of 52 characters. To transliterate Telugu characters into English characters, thus one needs to use capital letters also. The transliteration in this thesis relies on phonetic equivalence rather than ISCI1 (Indian Standard Codes for Information Interchange) character set.

Appendix **B, Morphological Rules,** gives a large sample of the morphological rules used in TELANGANA.

Appendix C, **Sample Lexicon,** gives sample lexical entries for some verbs, common nouns, proper nouns, pronouns, quantifiers, adverbs and determiners.

Chapter 2

Approaches to Natural Language Processing

2.1 Introduction

In the first chapter, traditional Indian approaches to understanding natural language sentences were dealt with. This chapter reviews the modern approaches to understanding natural language sentences. Over the past four decades researchers from various disciplines have attempted to study language from different perspectives. Some of them were concerned with using computers for handling natural languages. A significant part of their work can be termed as natural language processing (NLP). The expression 'natural language processing' (NLP) usually denotes 'using a computer to operate on the sentences of a natural language like English or Telugu, and perform tasks like retrieving, analysing, translating, understanding, and question-answering with respect to that language'. In this chapter NLP literature will be reviewed.

Most of the NLP research has been centred around English and other European languages. NLP research with respect to Indian languages is of very recent origin. Mainly Hindi has been the focus of this research. Very few papers have been published, and they are on parsing. There is no published literature on NLP for Telugu. Hence, NLP research on English and Hindi are reviewed in this chapter. Initially the history of NLP is briefly reviewed. This lays the foundation for understanding the background, motives and research directions of the different approaches to NLP. Subsequently, the major paradigms of NLP are reviewed. The Hindi oriented NLP falls into one of these paradigms.

Machine translation is the first sub-field of NLP that was investigated widely. Interest in machine translation and thus interest in NLP dates back to a memorandum entitled "Translation" circulated by Warren Weaver to his acquaintances during 1949 [Weav49]. In that memorandum some thoughts about problems arising out of multiple meanings of words (polysemy), the logical basis of language, and cryptographic techniques were expressed. Weaver believed that, the process of translation was similar to decoding military and diplomatic messages and hence amenable to mechanization. He also indicated that an intermediate "universal language" or *inter lingua* could be developed

that can be construed as an internal representation of the meaning of a sentence. When translating from language X (source language) to Y (target language), one could convert (transduce) X to the "universal language" first and then to Y.

Weaver's ideas triggered off a chain reaction, culminating in an enormous activity in machine translation. Several research teams embarked on the machine translation "band wagon" [Wilk87]. Machine translation also received wide spread public attention when a group of scientists from Georgetown University and IBM publicly demonstrated in 1964, a program which could automatically translate some carefully chosen Russian Texts into legible English texts. The vocabulary of the system consisted of only 250 words. The basic concept having been demonstrated, it was believed that full blown machine translation just required scaling up of the dictionary and the so called grammar rules.

Soon afterwards in 1966, the National Research Council of USA, after reviewing various machine translation systems available at that time, brought out a detailed report, called the ALPAC [Nati66] report, stating that "… there has been no machine translation of general scientific text, and none is in immediate prospect". The report showed that there were no significant improvements in the translation technology in the preceding 10 years of research. The techniques used in most of the translating systems were not significantly different from an earlier one developed by Oettinger in 1955. Basically, word to word translation was carried out. For polysemous words, all the meanings were included, so that the reader could pick out the correct meaning. The IBM system accomplished a little more by choosing amongst the intended meanings using some limited heuristics. The outputs of several systems were also included. Even after post-editing, the output was considered to be poor and also more expensive than manual translation. This brought to an end the automatic machine translation era. However the report suggested that more research is needed to be done in the area of computational and general linguistics before any more effort and money is spent on automatic translation. The initial euphoria having evapourated, researchers started moving into allied fields of artificial intelligence, computational linguistics, information retrieval and theoretical linguistics. The interest in the other sub-fields of natural language processing continued to exist and researchers began to appreciate the extent of their ignorance of the intricacies of natural language.

At about the same time, linguistics was also going through a major reorientation after Chomsky [Chom57] put forth his program of generative linguistics contradicting the traditional collative linguistics of Bloomfield [Bloo63]. This approach took a definite shape with the publication of a series of papers and books culminating in the 'Standard Theory' of Chomsky [Chom65]. Generative linguistics is characterized by its emphasis on the investigation of natural language through the construction of explicit descriptions of

some particular languages in some formal system like phrase structure rules and movement rules, and on cross generalisation of these descriptions over all possible languages (the so called "universal grammar"). The end of such explicit descriptions was achieving convergence towards a general theory of the structure of natural language. The underlying assumption of generative linguistics is that, there exists a unique universal grammar whose instantiations with different parameters are the different natural languages in use. The endeavour of the linguists is to discover this universal grammar. This stress on explicit descriptions has led to various formalisms amenable to strict mathematical scrutiny. Once a level of mathematical scrutiny is achieved by any discipline, computers can used for further research in that discipline.

Computer scientists, with their own individual fields of specialization like Logic, Formal Languages, Artificial Intelligence, Philosophy, Cognitive Science, Neural Biology etc, began investigating NLP from the above two routes of translation and linguistics. They started contributing to the study of languages from different perspectives. Thus, NLP being a symbiosis of numerous disparate fields, the approaches to the study of natural language (NL) differ widely. The fact that natural languages are excessively prone to ambiguity and are very difficult to formalize makes NLP an interesting field.

2.2 Dimensions of Comparison

Different schools of NLP use different approaches to understanding natural language, and are similar in certain aspects and different in others. Some major schools of NLP are described here. These schools are studied from two differing complimentary views; the Linguistic view, and the Artificial Intelligence view. The Linguistic view largely concerns itself with the 'formalization' aspects of a NL, and hence addresses itself to issues like,

(1) Specification of the syntax and the semantics of natural languages, and

(2) Learnability of natural language.

The Artificial Intelligence view, on the other hand, concerns itself with developing NL models that can be implemented on computers and consequently looks at the additional issues arising out of Operationalizing[1] a natural language theory. The issues arising while Operationalizing a NLP theory are

(3) the computational complexity of the theory, and

(4) the handling of the ambiguities arising in natural languages.

[1] Implementing a theory on computers.

There is a precise definition of the term "computational complexity" in relation to recognising the syntactic well formedness of sentences [Gazd85b], but not in relation to recognising the semantic well formedness of sentences. Hence in this review the expression "computational complexity", is restricted to mean "syntactic complexity" only. If one takes 'n' as the number of words in a sentence, the computational complexity of various NLP methods ranges from O ($n^{2.73}$) to exponential depending on the method of syntactic recognition employed. As the range is so large, one needs to know the complexity of a method.

The biggest impediment to Operationalizing NLP theories is the presence of ambiguities in NLs. Some of the schools of NLP consider, ambiguity resolution and pragmatics as the only real problems plaguing NLP. Ambiguities in NL can be broadly categorized [Wilk87] as Lexical ambiguities, Structural ambiguities, and Referential ambiguities. It is possible to further finely classify these three categories of ambiguity and also add a few more to the list [Hirs87]. However the discussion here is restricted to the above three broad categories only, as the majority of the ambiguities occurring in natural language texts belong to these categories [Hirs87, Wilk87J.

As most of the linguistic and Al oriented NLP literature is developed around English, to give an idea of the relevance of the various documented techniques to the study of Telugu, suitable Telugu oriented examples are given through out this chapter. As and where appropriate the suitability of each particular paradigm to the study of Telugu in general and to the more immediate goal of building a practical system for Telugu is also discussed. Without this kind of correlation, the author believes that it would be time consuming for the reader to perceive the relevance of the reviewed literature to the study of Telugu syntax and semantics.

A review of the various methods and paradigms of NLP with respect to the already mentioned four criteria related to the specification of

- syntax and semantics,
- learnability,
- computational complexity and
- ambiguity resolution

would give a good picture of the strengths, weaknesses, descriptive adequacy, observational adequacy and explanatory adequacy of each paradigm. Some of the terms, like syntax, semantics, pragmatics, lexical ambiguity need elaboration in order to clarify

the sense in which they will be used in the rest of the thesis. These terms will be explained in the context of Telugu presently.

2.2.1 Syntax

The notion of syntax is central to any language. Every natural language sentence when uttered or written down, requires that the constituent words are uttered in certain temporal order or written down in some spatial order (right to left, top to down, or right to left etc.). Syntax attempts to tell which word order is acceptable and which is not, on a per sentence basis. It does not include multiple sentences and the legality and illegality of their order in a discourse. For example the following Telugu and English sentences are illegal (illformed) purely on syntactic grounds, and can be made acceptable by just reordering the words.

రాముడు వేగంగా సీతకంటే పరిగెత్తాడు
* *raamuDu vEgaMgA sitakaMTE parigettADu*
Rama fast more than Sita ran

రాముడు సీతకంటే వేగంగా పరిగె _డు
raamuDu sitakaMTE vEgaMgA parigettADu
Rama ran faster than Sita.

చూసాడు రాముడు సీతని
coosADu rAmuDu sitani
* saw Rama Sita
Rama saw Sita

From here on, as is common in the linguistic community, all illegal sentences will be starred as above.

Syntax, thus, concerns itself with the correct formation of the sentences of the language. As syntax is so essential for the perceptible form of the sentence, every theory of language should have something to say about the word order in the language. The word order is given through phrase structure rules, such as

S --> NP, VP
VP --> V, NP

Many strong arguments[Newm80] have been put forward to support the notion of phrase structure in English. The notions of sentence (S), complement sentence (COMP), Noun Phrase (NP), verb phrase (VP), adjective phrase (AP), preposition phrase (PP) are well established. The above phrase structure rule, tells that a sentence consists of a NP and a VP. The VP inturn consists of a V and NP. It has been hotly debated and sufficiently

agreed upon [Newm80] with respect to English that the above two rules can not be merged into a single rule as follows,

S --> NP, V, NP

even though this new rule generates the same set of sentences as the earlier two rules. The first two rules imply a phrase structure consisting of a VP, whereas the new rule indicates the absence of such a phrase structure. In other words, the new rule implies a flat structure for the language.

The notion of phrase structure is not well established in many languages like Telugu, Hindi, Tamil and Sanskrit, and also in Japanese. For example two different views are held about the Japanese language. One view supports a flat structure [Sait85] and consequently phrase structure is absent. The other supports the presence of a hierarchical phrase structure with a clear cut VP [Gunj87]. There is no unanimous agreement as to what constitutes a VP or whether VP nodes need to be assumed to exist or not. To the best knowledge of the author, there does not seem to be any systematic study giving reasons, one way or the other, for the existence of phrase structure in Telugu. Some linguists have implicitly assumed the presence of phrase structure by couching their principles as refinements to transformational grammar or GB approach [Subb84, Wali91], without really giving strong reasons for choosing a phrase structure treatment of Telugu. Hence in the thesis, the use of the notions of NP, VP etc. with respect to Telugu are avoided. In the context of Telugu, the term PP is used for POSTPOSITION PHRASE, meaning a "noun conjoined with *vibhakti* (case indicating suffix)". The expression syntax, thus in the context of Telugu, just means surface word order and has no implication with regard to phrase structure.

2.2.2 Semantics and Pragmatics

Semantics concerns itself with the interpretation of expressions in a language. Given a legal sentence, one needs to specify and represent its meaning in some form (either formal or informal). Traditionally, semantics deals with computing the meaning representation of a sentence from its parts. In other words, semantics deals with how to represent the meaning of individual words, and how to combine these individual word meanings to get the meaning of the whole sentence. It would appear that not all aspects of word meaning are in the realm of semantics. It has been argued by many linguists that some aspects of meaning are to be explained in terms of theories of action. For example, when a glass of water is ordered in a restaurant by saying "I would like a glass of water", the sentence is intended to produce an action on the part of the waiter. It is not meant to tell the waiter ones wish, nor seek an yes or no answer from him. So the phrase

"semantics of a sentence", in the rest of thesis, is restricted to mean "the meaning of the sentence more or less invariant across contexts and situations", virtually the "literal meaning of the utterance in a situation". The author considers that, the intention of the speaker, what people do in entreating, and asserting, filling in missing information and all other such contextual and situated meaning extensions, fall under the realm of Pragmatics. Thus the sentence "Alice talked to the red flower" is ill-formed on semantic grounds whereas well formed on syntactic grounds. It could possibly be well formed on pragmatic grounds.

2.2.3 Lexical Ambiguity

Lexical ambiguity, as far as NLP is concerned, manifests in two ways: polysemy (poly = many, semy = meanings) and categorical ambiguity. Typically, words have many meanings. For instance the word *ball* (in English) could mean: a dancing party, a spherical entity, or a cannon missile. The word *tokku* in Telugu could mean: to press with the foot, to suppress, the peel of a fruit like orange, or a pickle. The word *saindhavam* in Sanskrit means : a horse or salt. The ability of words to harbour multiple meanings is the beauty of natural languages and no natural language can afford not to have it. In addition to these essentially meaning related ambiguities, words can display ambiguity related to their syntactic categories also. For example, the word *ettu* in Telugu could be a noun, in the sense of 'plan in a game' or in the sense of 'height' or could be a verb, in the sense of 'lift'. In English the word *coach* could be noun, in the sense of an instructor, or a verb, in the sense of instructing.

Categorical ambiguity can normally be resolved within a syntactic framework, whereas polysemy cannot be resolved so easily. This is because polysemy can lead to multiple parse trees, all of which would be syntactically legal. As the dictionary size and the linguistic coverage of a system grows in size, these ambiguities grow exponentially and create enormous implementation problems [Naka88: pp 57-59]. For a normal reader it is not difficult to work out the unique meaning of a sentence by using his knowledge of the world and the context of the text as a whole. In fact quite often it is very difficult for normal readers to see the other possible meanings. The human mind can retrieve the correct meaning out of the possible 'n' meanings by using some unknown mechanisms pertinent to the context. In contrast to this, present day computer programs retrieve meanings of words more or less on a context-free basis. Hence one needs to know the extent to which a theory caters to the resolution of lexical ambiguity.

2.2.4 Structural Ambiguity

A sentence is said to be structurally ambiguous if it can be analysed in more than one way without resorting to polysemy of words. The sentence "John saw a person in the garden with a telescope" could be understood many ways. It could mean, "John saw a person who was in the garden with the help of a telescope", or "John saw a person who was in the garden and who was carrying a telescope", or "John was in the garden and he saw a person from there using a telescope" and so. In the same vien, the following sentence in Telugu can be analysed in two different ways, giving rise to two different meanings.

వానరులు కొండఎక్కి లంకకు ఎగరబోతున్న హనుమంతుడిని చూశారు
vAnarulu koMDaekki laMkaku egarabOtunna hanumaMtuDini cooSAru
monkeys hill climb-past to-Lanka about-to-fly Hanuman saw

In the above sentence, the actor of *koMDaekki* (ascended the hill) could be either *vAnarulu* (monkeys) or *hanumaMtuDu* (Hanuman). To put it in a technical way, the PP, *vAnarulu*, can be an argument of either the verb *ekki* (climbed) or the verb *cooSAru* (saw), and likewise the PP *hanumaMtuDu* can be an argument of *ekki* in addition to being an argument of the inflected verb *egur* (root of *egarabOtunna).* Thus the above sentence conveys two meanings as follows.

The monkeys ascending the mountain saw Hanuman who was about to fly to Lanka.

The monkeys saw Hanuman, who having ascended the mountain, was about to fly to Lanka.

In many sentences PPs can be attached either to other PPs or to the different head verbs as arguments. This leads to structural ambiguity. Another major area where ambiguity manifests is in coordinate constructions. The following sentence has four interpretations, indicated by brackets.

రాముడికి రోడ్డుమీద అమ్మే పాత పుస్తకాలు మరియు గడియారాలు అంటే ఇష్టం
rAmuDiki rODDumeeda ammE pAta pustakAlu mariyu gaDiyArAlu aMTE ishTaM

Rama on-the-road sold old books and clocks likes
Rama likes ((old books) and (clocks)) sold on the roads.
Rama likes ((old books) and (old clocks)) sold on the roads.
Rama likes (old books) and (old clocks sold on the roads).
Rama likes (old books) and (clocks sold on the roads).

In this fashion structural ambiguity can lead to multiple meanings, quite often not resolvable even by the immediate sentential context. Structural ambiguities can arise due to adverbial phrases, adjective phrases and relative clauses also.

2.2.5 Referential Ambiguities

This refers to the fact that pronouns often refer to some nominals (nouns) in the discourse. One does not know to which one they refer to if there are multiple nominals in the discourse. Taking an example from Hobbs [Hobb77], which he quotes had appeared in *Newsweek*, one can easily see how ubiquitous the problem is.

> The FBI said that they had tentative identifications on the fugitives, but didn't know where *they* were.

The word they in italics "they" can easily refer to either the FBI agents or the fugitives or the identifications of the fugitives. Purely syntactic methods would not help in resolving the above ambiguity. In the following example from Wilks [Wilks'83] in two very similar (syntactically) sentences the word "she" refers to two different entities.

> Mary told her mother that *she* was intolerant (she normally would refer to mother)
> Mary told her mother that *she* was pregnant (she normally would refer to Mary)

Resolving these reference ambiguities is very difficult, as world knowledge or pragmatic knowledge is needed and no amount of syntactic knowledge would help.

2.3 Foundationally Different Approaches to NLP

The four criteria of analysing NLP paradigms, specified in section 2.2, thus, would give us an idea of the level and scope of the achievements of various NLP schools. The author has categorised the methods and philosophy adopted by various schools into five (slightly overlapping) categories:

(A) Principle Based Approach,

(B) Syntactico-Semantic Approach,

(C) Mathematical Approach,

(D) Pragmatic Approach

(E) Connectionist Approach.

The categorisation is based on how much stress is given to the study of the different aspects of language in research, in addition to the direction and orientation of the research. The above categorisation cuts across the obvious differences between various methods at realization (implementation) level and unifies them at the underlying philosophical and foundational level. In the ensuing sections, each of the above approaches will be described. The central concept will be presented first, and then its interesting variations will be touched upon briefly.

2.3.1 Principle Based Approach

The Universal Grammar (UG) approach of Chomsky is reviewed first. This approach is also called a principle based approach by computational linguists. Virtually every innovation in the field of theoretical linguistics, for the past thirty five years, has been either an elaboration of or a reaction to some of the notions of Chomsky. Hence in some way, a review of UG amounts to a review of 75% of the theoretical linguistics [Newm80].

The notion of UG was championed by Chomsky. The roots of UG are wide spread both in time and differing areas of research. Initially, Chomsky |Chom57] stressed the need for "observational adequacy", that is generating the correct set of strings for the natural languages, as a theme of linguistic research. Observational adequacy was paramount in the order of priorities while tackling the issues of NLP. The nature of constraints imposed on well-formedness of sentences tended to be inclined heavily towards some structural properties of the language under investigation. Until the early sixties Chomsky's work was dominated by this spirit. He put forward his theory of transformational grammar advocating the "method of rigorously stating a proposed theory and applying it strictly to linguistic material with no attempt to avoid unacceptable conclusions by ad hoc adjustments or loose formulation" [Chom57]. Meaning was deemed to be outside the realm of linguistics and top priority was given to developing a theory capable of generating all and only those sentences that were well-formed. It was fell that phrase structure grammar alone was inadequate and required an additional transformational component to achieve observational adequacy.

In the mid sixties the focus changed substantially, when Chomsky published his "Standard Theory" (ST) [Chom65]. ST gave a better status to semantics, and language was viewed as a system of connections between meaning and utterance. ST identified two levels of representation of sentences: deep and surface structure. Deep structures served as the basis for semantic representation and surface structure as the basis for phonological interpretation. These two levels were related through transformations. It

was felt that these transformations would ultimately prove to be "psychologically real and account for the ways humans process natural language sentences and the way they relate sound to meaning. However at that stage "psychological reality" marred the semantics of the language and not much was done beyond a sketchy description of the 'deep structure[1] of sentences. Along with this, some amount of emphasis was laid on the learnability aspects of languages as well. From the seventies to the eighties there was a gradual shift in the emphasis placed on explicitness and formalization. More emphasis was laid on the learnability and universality of theories. The epitome of this is the Government and Binding (GB) theory of Chomsky introduced in 1981 and improved in 1986.

The point of view of practitioners of GB theory is that the human mind comes with a lot of wired-in linguistic knowledge which is language independent. With exposure to a particular language, this wired-in knowledge gets fine tuned within a predetermined range, and attains the capability to distinguish wellformed and illformed expressions of that language. This endowment of the language faculty is known as Universal Grammar (UG). In other words UG is a grammatical system "flexible enough to account for the language variations while at the same time be, to a large extent, restricted in order to account for the relative ease of language acquisition and the impossibility of certain language types" [Trav89, p.263]. Thus the description of language follows certain "principles" rather than " rules".

As no review of linguistic approaches can be complete without an exposition of Chomsky's work, an attempt is made in the succeeding paragraphs to present a glimpse of the principles of GB theory. To make the understanding easier to computer scientists and non-linguists, an example of a parsing scheme for parsing simple sentences using GB principles is shown. For greater details regarding GB theory a standard text book such as [Reim86] or monographs such as [Chom81 and Chom86] should be consulted.

In GB theory a sentence is represented at four levels, D-structure (DS), S-structure (SS), Phonetic Structure (PS) and Logical Form (LF). To put it in the most simple form, DS captures the argument structure of lexical categories: Verb, Noun, Pre (or Postposition and Adjective, present in the sentence. The LF comes nearest to the meaning representation of the sentence in terms of quantifier scoping, anaphoric reference indexing etc. Quite often PS and SS are the same, but they could also be different as in " I wanna go" (PS) and " I want to go" (SS).

In the GB grammatical model a crucial role is played by interacting systems of principles (theories), namely X-bar theory, Thematic theory (θ–theory), Government theory, Case theory, Bounding theory, Binding theory and Control theory. These theories state general

conditions on the well formedness of the different levels of representation (DS, SS, PS and LS) of a sentence. All the above systems are associated with some parameters. The principles in each of these theories are taken to be universal and thus account for the "relative ease of language acquisition". The ranges of values on the parameters account for the "impossibility of certain language types". When one learns a language, one assigns (quite unknowingly) values to these parameters. This makes ones grammar fine tuned to the language that is being learned.

The concept of phrase structure is central to GB theory. An understanding of the phrase structure and some terminology is useful before GB theory is elaborated. A phrase structure is represented as a tree as shown in figure 2.1. (There are many different forms presented in literature. For pedagogical purposes this particular version is adopted).

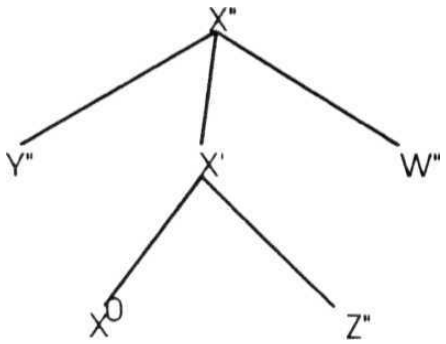

Figure 2.1, A typical Phrase Structure tree used in X-bar theory

In the Figure 2.1 X" is said to **dominate** Y", W", X', X^0, and Z", and X' is said to dominate X^0 and Z". X^0 does not dominate any thing. The terms Y", Z" in turn dominate their sub-constituents. Typically Y" is called a **specifier** and W" a modifier and Z" **argument.** The set of possible specifiers for nouns include determiners, quantifiers etc. The set of specifiers for verbs include auxiliaries, medals, verbal clitics and so on. The set of specifiers for adjectives are degree modifiers like very, most etc. Many times it is difficult to distinguish between arguments and modifiers. However this is not problematic, because the hierarchical structure is more important for GB than the order. The term X^0 refers to lexical heads: Verb (V), Noun (N), Pre (Post)position (P), Adjective (A) and INFL (inflection information like, Tense and Agreement, on the verb) . X' and X" are called the bar level one and bar level two projections of X^0. X" is called the **maximal projection** of X^0. Typically VP=V", NP=N", PP=P", AP=A", and for historical reasons S'=INFL" (there is no unanimous agreement to this notion of S'=INFL"). It may be noted that, under this scheme S=INFL'. S is not a maximal projection of INFL, while S' is so. Excepting specifiers, all else in the above phrase structure are needed to be present at maximal projection level.

their lexical heads. GB theory assigns essentially three roles [Will81] to the arguments, namely Agent, Theme and Goal. These are called θ–roles. θ–roles are assigned by a (lexical) head within its domain to its argument positions. A position which has a θ–role assigned thus, is called θ-marked. θ–theory states that each argument bears one and only one θ–role and, each θ–role is assigned to one and only one argument. This unique assignment criterion is called θ–criterion. The other role of 6-criterion is to constrain the domain of the move-α rule. θ–criterion stalls move-α from moving an argument from one θ–marked position to some other θ–marked position because if this happens the moved item gets two θ–roles, violating the θ–criterion.

The Case theory and related theories, characterize the well-formedness criteria at SS, LF, or both levels. Case theory is largely responsible for determining the distribution of NPs at SS level. It states that if a NP has a phonetic content, it must have a case. This accounts for the object movement to subject position in passive sentences as follows. During passivization, a verb looses its ability to 6-mark its governing positions and its ability to assign case. However INFL still has its capability to assign case. As a result, the subject position which hangs without a θ–role assigned to it needs a NP to fulfil the Case theory constraint (called Case filter) arising due to the INFL position. So the subject position is free (and in fact mandatory) to host the object NP. Hence an object NP is mandatorily moved to the subject position in a passive sentence. Many other phenomenon (like distribution of traces) can be similarly explained by case theory.

In the previous paragraph the word "governing" was used without giving a denotation to the word, but by appealing to the intuition of the reader. The intuitive idea behind the concept of government is that a head governs the syntactic positions occupied by its thematic dependents, provided a specific structural relationship exists between that head and such a position. To such governed positions only can a head assign 6-roles. The notion of c-command is fundamental to government and also to Binding theory.

C-command is defined as

a c-commands β iff every maximal projection dominating a dominates (3

Government is defined as

a governs β **iff** (1) a c-commands p.,
(2) a is head i.e. X^0 and
(3) every maximal projection dominating p dominates a.

In figure 2.1, X^0 is the head, and it governs Y", and Z". That is, a head X^0 has access to every syntactic position (Y" and Z") inside its maximal projection X". Y" and Z" also act like barriers that do not let the head access their internal positions. It may be

All structural relations in GB are made against the background of a phrase structure as above. In a linear form the above is written as [X" *a* [X' X^0 (3]] where a and β are the contents of Y" and Z". For Telugu the order of X^0 and Z" will be different as [X" α [X' β X^0]]. The order of X^0, Z", Y" and W" as they appear in the tree (fig 2.1) depends on what is called the **headedness** parameter. The headedness parameter for Telugu is "head-final". It is in direct contrast to English where the headedness parameter is 'head-initial'!. Due to this, in the linear form Y", W" and Z" occur before X^0 in Telugu. So the phrase structure tree of Telugu should be, as shown in figure 2.2

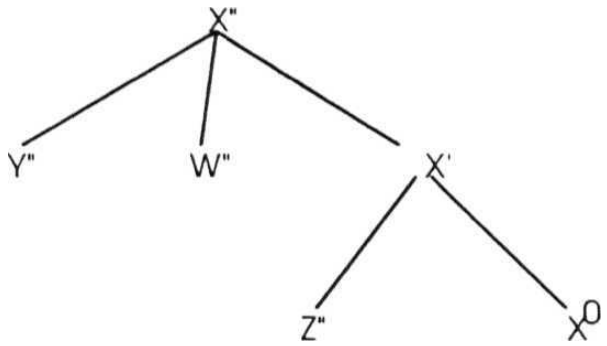

FIGURE 2.2 A possible Phrase Structure tree for Telugu

The Projection Principle states that "representations at each syntactic level are projected from the lexicon, in that they observe the Subcalegorization properties of lexical items" [Sell85, pp 33]. This principle has far reaching consequences. If one considers a lexical head like *cooD* (a Telugu verb meaning 'see'), which takes an object and a subject, the projection principle forces the presence of a syntactic position occupied by an object and a syntactic position occupied by a subject to bc present in every DS, SS and LF representation of the sentence of which *cooD* is a constituent in the capacity of a verb.

The X-bar theory addresses itself to the phrase structure of the language at DS level. It spells constraints on the kind of phrase structures that are legal at DS level. From the D-structure of a sentence, its SS and LF are realized through a transformation called move-ex. The move-α transformation can be used to move, virtually, any item in the DS to any other position, to transform it into a corresponding SS, PS or LF. The X-bar theory, together with the Projection Principle which states that all projections of representations at various syntactic levels must obey the Subcalegorization of the lexical items, curtails many of the transformations permitted by move-a. Move-α is also subject to some other constraints by other components of the GB theory like Bounding theory, Case theory, and θ–theory.

One crucial component of GB theory, namely θ–theory, spells the principles governing the relationship between the arguments and the roles they ought to play with respect to

observed that in the configuration of the government, subcategorization is satisfied, internal θ–role assignment takes place, and case assignment takes place.

Binding theory deals with the principles ruling the inter-relationships between NPs, and covers among others, the distribution of pronouns, reflexive pronouns and anaphors. Two NPs are coindexed if they refer to the same entity. For example in the following sentence,

రాముడు వాలిని తన కారులో వాడి ఇంటికి తీసుకు వెళ్ళాడు

rAmuDu$_i$ vAlini$_j$ tana$_i$ kaarulO vADi$_j$ iMTiki teesuku veLlADu

Rama$_i$ took vali$_j$ to his; home in self$_i$ (i.e. his$_i$) car.

The subscripts i and j are used to co-index the antecedents *rAmuDu* and *vAli* with the precedents *tana* and *vADi* respectively. For Coindexing to be valid, it must satisfy Binding theory constraints. Coindexing is valid only when the antecedent c-commands the precedents. Binding theory partitions NPs into four types and states binding conditions for each. Reflexive pronouns and ordinary pronouns are only considered here. Reflexive pronouns are bound only in their local domain while other pronouns are free in their local domain. Local domain of an item a is the smallest NP or S containing both a and a governor of a.

For the sake of clarity, an example is given to show how one could do parsing using the GB principles. For the sake of the example, a phrase structure, with the headedness parameter set to 'head-final', is assumed for Telugu. A hypothetical parser is assumed which can generate a parse tree according to the phrase structures of the grammar. In the course of this example it will be possible to explain what GB theory has to say regarding the four dimensions (cf. previous section) of understanding of a grammatical system. Let us take the sentence,

రాముడు తన తల్ని కలిశాడు

rAmuDu$_i$ tana$_i$ tallin$_i$ kaliSADu

rama$_i$ his$_i$ mother met

Rama$_i$ met his$_i$ mother

A parser following the GB principles should give the tree in figure 2.3 as the d-structure for the above sentence. Note the configurational position of the object, *tana tallini.* It is to the left of the governing verb *kaluv*. This is so because the headedness parameter for Telugu is 'head-final'.

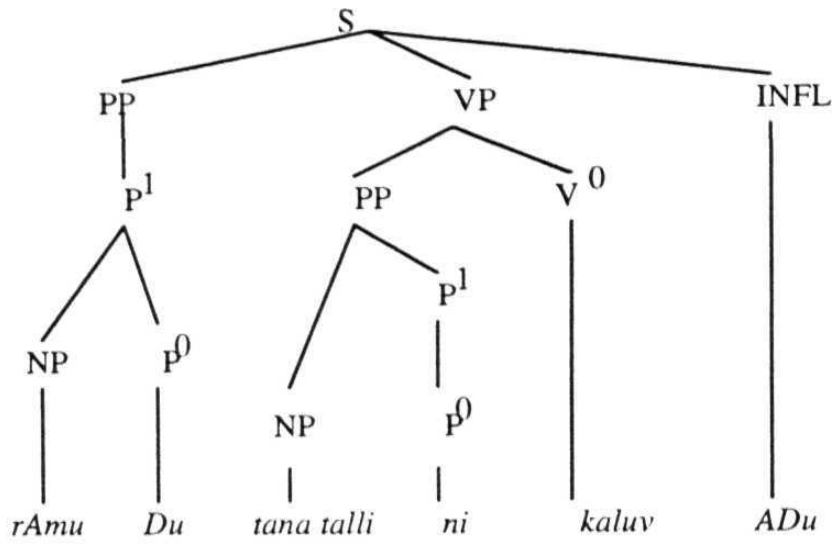

FIGURE 2.3 Phrase structure tree for the *sentenoAmuDu tana tallini kaliSADu*

The parser processing the sentence (left to right) would first encounter the word *rAmuDu.* It realizes that this is a noun with a postposition *Du. rAmu* would be found in the lexicon as a proper noun, thus can be a lexical head around which a phrase structure can be built. It does not have any complements or specifiers, hence its maximal projection NP, is formed from *rAmu.* From this NP and the postposition *Du,* the maximal projection PP is formed, as dictated by the X-bar theory. Then the word *tana* is analysed and treated as a specifier to the next in coming word and hence is kept in store. Then the word *tallini* is analysed. Using the word *tana* in store and the recently processed words *talli* and the postposition *ni,* the parser forms a PP (postpositional phrase), the maximal projection of the postposition *ni.* Then the word *kaliSADu* is analysed. The lexical head is the word *kaluv* with INFL *ADu* indicating 3rd person singular. The word *kaluv* being a verb, the maximal projection of it can be formed, if a suitable theta-marked NP (satisfying syntactic and semantic sub-categorization of *kaluv),* can be found. From the lexical information associated (as seen from the lexicon) with the word *kaluv,* it will be possible to ascertain that *kaluv* requires an argument in accusative case. Hence the *PP tallini* can satisfy the subcategorization requirements. Thus a maximal projection of the verb *kaluv* can be obtained. Let us assume that the subject *rAmuDu* is also similarly accepted. Now the entire sentence is accepted. However due to the presence of the reflexive pronoun *tana,* one has to invoke the Binding theory module and see if this structural position for the pronoun is allowed. According to the Binding theory, it can be easily verified that the possessive pronoun *tana* is c-commanded by *rAmu,* and hence can be co-indexed with and bound to *rAmu.* Thus the principles of X-bar theory, Theta-theory, Case-theory and Binding-theory working independently (modularly) can assign a d-structure to the entire sentence. This shows briefly how the syntax of the sentence is

taken care of. Once the d-structure is obtained, the following LF-structure of the above sentence is obtained as granted by theta-criterion and move-α rules.

[pp rAmuDu$_i$] [pp tana tallini$_j$] [$_S$ kaliSADu e_i e_j]

In the above e_i and e_j are logical referents to *rAmuDu* and *tana tallini* respectively. The LF-structure of GB is supposed to encode the information relevant to the semantic information of the sentence. Such questions as quantifier scope and the scope of question words etc., are dealt with at this LF-structure level. The representations used are intended to bear a similarity to first order logic, in which left-right order indicates the relative scope of quantifiers.

In GB theory, historically, the tools of the theory have been developed primarily to explain the various syntactic facets of the language better. Semantic aspects have not been studied to the same extent. As far as possible, answers to every problem are first sought out through configurational or syntactic criteria, before any semantic explanation is attempted (for example, binding theory and case theory rely in some fundamental ways on the structural notions such as c-command or government). Hence GB theory inspite of having contributed immensely to the study of language, is essentially syntactic in nature and does not throw much light on the semantic and pragmatic aspects of language. The question of ambiguities is not addressed by any of the GB theoreticians. Researchers who have developed parsers based on GB theory |Wehr88, pp 2(X)] have brushed aside the issue saying "The fact that these nouns can receive various interpretations is a matter of semantic and pragmatic interpretation that should not concern syntactic processing". Finding out the computational complexity of approaches based on GB theory is a difficult problem. Hence not much can be said about it, except that certain move-a type rules can lead to exponential computational complexity. This problem is inherited by GB from its ancestor transformational grammar. The entire GB theory is in a state of flux and many concepts and parameters in it are not yet well understood. As the focus of Chomsky style linguists has changed from observational adequacy to explanatory adequacy, standards of explicitness (compared to say GPSG, next section), rigour and attention to empirical detail have declined (probably, rightly so, as larger the questions addressed, the harder it is to give complete answers). This makes GB theory difficult to use with confidence.

One key observation can be made with respect to the current linguistics as exemplified by research in GB theory. In GB theory, θ–criterion stipulates that the meaning of a predicate determines the grammatical arguments it will have. The projection principle assures that the structure determined by the lexical head's meaning cannot be altered in essential ways. The problem of acquiring a language, then, reduces largely to learning the

meanings of words. Of course, in addition one needs to learn, how to compose complex sentences from simpler ones. In the canonical cases, however, sentence structure is a projection of the meaning/semantics of the constituent words. Every theory docs seem to converge towards this **lexicalist** view. It is interesting to know how close this view is to the traditional Indian grammarians' views, as exemplified in Asthaadhyaayi [Josh68] and elaborated in a lexical way in Sidhantakaumudi, and Praudhamanorama [Dixi68], which were reviewed in the first chapter.

2.3.2 Syntactico-Semantic Approach

The other major paradigm of NLP is what is referred to in this thesis as Syntactico-Semantic approach. This approach gave equal importance to both the syntax and the semantics of natural languages right from its inception. To the extent possible, syntax and semantics are studied together without even remotely implying the autonomy of any one of them. In this approach, illegality of sentences can also be described through purely semantic notions. Notions based on configuration such as c-command, and transformations like move-α are not utilized to explain the syntactic phenomenon of a language. Semantic considerations are taken into account when ever it leads to a simpler theory.

Two variations can arise in this syntatico-semantic approach depending on where stress is placed, syntax or semantics. The approach taken by linguists [Gazd85a, Poll87] inclines towards syntax and the approach taken by AI researchers [Wilk75, Boug79] inclines towards semantics even though neither system underestimates the importance of syntax or semantics.

Generalized Phrase Structure Grammar (GPSG) [Gazd85a] is a Syntactico-Semantic theory that gives equal importance to both syntax and semantics. The GPSG literature exhibits a keen interest in the mathematical properties of grammatical formalisms. It has revived interest in traditional questions like "How context sensitive are natural languages?", and "What is the generative capacity of a formalism?" [Gazd85b]. In order to investigate such questions, it is necessary that grammars are formulated with considerable precision. The book by Gazdar et al [Gazd85a] is a testimony to the precision achieved by the practitioners of this approach. GPSG has skilfully interwoven augmented phrase structure grammars with the Montagovian semantics [Mont74b, Dowt81] to such an extent that sematic analysis is an integral part inseparable from the syntactic proposals. In GPSG theory, multiple levels (d-structure, S-structure, etc.) of sentential representation as in GB theory and transformational grammars do not exist. GPSG is a homogenous theory and posits only one level of syntactic representation. Augmentation to phrase structure (PS) is done in order to give weak context sensitivity to

the underlying phrase structure grammar so that well known phenomenon like wh-movemenl, long distance phenomenon, relationships between gaps and fillers, that were deemed to require transformations, can be handled through the augmented PS rules. Consequently, syntactic analysis of a sentence is shown as a tree (see for example figure 2.7) with nodes made up of augmented symbols called categories or feature bundles.

A category in GPSG may be considered as a partial function from features to their values. Intuitively, a feature-value pair can be thought of as a piece of linguistically significant information. Features such as NUMBER, TENSE, AGR, V, and N are either *atomic* valued as <NUMBER, SINGULAR>[2], <TENSE, PAST>, <N,+>, <V,->, <BAR, 2> or are category valued as <AGR, [<NUMBER, PLURAL>, <PERSON, 3RD>, <GENDER, NEUTER>]>. All syntactic theories use features but only few of them have put features to a principled use and built a formalism around them with the thoroughness of GPSG [Gazd85a]. Categories being functions, categories of the following kind are not possible (because functions by definition cannot deliver two values for the same argument).

[<N,+>, <V,->, <GENDER, FEM1NINE>, <GENDER, NEUTER>]

An abbreviatory convention is used in GPSG literature wherein syntactic categories are abbreviated up to the point they are not ambiguous. Thus a verb phrase that consists of a passive verb is abbreviated and written as VP [PAS] instead of its verbose description as [<N,->, <V,+>, <VFORM, PAS>). A noun phrase with the feature specifications <CASE, NOM> and <POSS, +> is written as NP [NOM, +POSS]. The category-valued SLASH feature is abbreviated as '/'. VP [VFORM PAS, SLASH NP] is written as VP [PAS]/NP. This kind of abbreviatory notation will be used henceforth. A numerical value appearing inside square brackets denotes a SUBCAT value; H [2] means H [SUBCAT 2], while a numeric value used as superscript over a symbol abbreviates the BAR value; H^2 means H [BAR 2].

In the current version of the GPSG theory, categories encode subcategorization, agreement, unbounded dependency, predication and other syntactically significant information. GPSG contains five language specific components:

- feature co-occurrence restrictions (FCRs),
- feature specification defaults (FSDs),
- immediate dominance (ID) rules,

[2]The pair <F,V> means, feature F has value V.

- linear precedence (LP) rules and
- meta rules,

to form a system of grammar from the aforementioned notion of categories. Apart from these language specific components GPSG also provides three language-universal components: a theory of syntactic features, principles of universal feature instantiation and principles of semantic interpretation, together with a specification of the formal relationships among the various components of the grammar.

2.3.2.1 Feature Specification

As one might expect, not every feature specification is valid. Feature co-occurrence restrictions (FCRs) and feature specification defaults (FSDs) constrain the formation of the categories. In English, for example, an interrogative sentence must have an auxiliary and a finite verb. This condition on the co-occurrence of some features is encoded in the following FCR.

[+INV] ==> [+AUX.FIN] -2.5

The above indicates that when a category has the +INV feature, it must also have the <AUX,+> and <VFORM,FIN> feature specifications. Consequently, a feature bundle (category) of the following type is invalid.

f+INV,ED]

In the above feature specification, the feature +INV implies [<AUX,+>, <VFORM,F1N>] due to the FCR given in 2.5 and the feature ED implies <VFORM,ED>. The above feature specification is invalid as the feature VFORM is assigned two values. The following are a few more examples of the FCRs useful for English.

[VFORM] ==> [-N,+V] meaning only verbs have VFORM feature

[BAR 1] ==> ~ [SUBCAT] non-lexical heads [ie BAR =1 or 2] cannot possess SUBCAT feature

[COMP *for]* ==> ([FIN] V [BSE]) A sentential compliment must be untensed

The FSD gives the assumptions/default values that can be assigned to features in case feature values are not indicated in some feature bundles. The following FSD,

[-INV]

encodes the information that, if no information about INV feature is given then the feature specification for INV is assumed to be [-INV]. The following are a few more FSDs useful for English.

[+ADV] ==> [BAR 0]	meaning, if ADV = +, then assume BAR=0
[INF, +SUBJ] ==>[COMP *for]*	meaning, if for a verb phrase VFORM=INF and SUBJ=+, then assume COMP=*for*
[NFORM] ==>[NFORM NORM]	meaning, the default NFORM value = NORM

2.3.2.2 Immediate Dominance/Linear Precedence (ID/LP) rules

The ID/LP rules of GPSG factor out two important and independent relations that are normally specified in context free phrase structure rules: immediate dominance and linear precedence. The following phrase structure rule in some context free grammar C,

$$C_0 \rightarrow C_1 C_2 \ldots C_n$$

specifies that the non-terminal symbol C_0 can be expanded into (or replaced by) C_1 $C_2 \ldots C_n$ string while generating (or parsing) a sentence using the grammar G. The constituents of the string $C_1 C_2 \ldots C_n$ should occur in the sentence in that order, only then they can be substituted for CQ.

In GPSG the constituent order information is removed from the phrase structure rule. Consequently information in the phrase structure rules are encoded using two different set of rules: immediate dominance (ID) rules and linear precedence (LP) rules. ID rules are context free production rules specified as

$$C_0 \rightarrow C_1 , C_2 , \ldots , C_n$$

wherein the left-hand side (LHS) C_0 is the mother category and the right-hand side (RHS) is an unordered multiset of daughter categories, some of which may be designated as *head categories.* Head categories are written using a special symbol H. One of the major uses of the special symbol H in an ID rule is to highlight the categories that determine the information flow between the LHS and the RHS of an ID rule. The mother category C_0 is said to *dominate* the daughter categories C_1 to C_n. The above ID rule can be pictorially depicted as a tree of depth one (a *local tree)* as shown in figure 2.4.

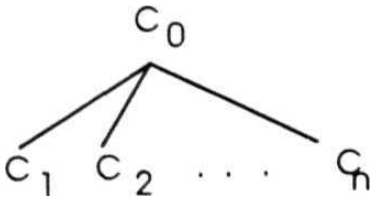

FIGURE 2.4 : Local Tree

The following arc examples of typical ID rules applicable to English

ID1: S —> NP,H [-SUBJ]	Rama went home, Sita saw Rama in a garden
ID2: NP —> Det,H[1]	The cat, some dog
ID3: N[1] — > H [30]	Rama, Sita
ID4: N[1] —> H [35],PP [*of*]	story of Rama, Battle of Bulge
ID5: PP—>H[38],NP	with a pen, on going home
ID6: VP — >H[1]	sleep, run,die
ID7: VP—>H[2],NP	saw the person, chased the dog
1D8: VP — >H [3],NP,PP	gave the book to Rama, met Sita at home
ID9: VP —> H [5],NP,NP	gave Rama a book, bid him farewell
ID10: VP —> H [8],NP,S [FIN]	asked Rama to go home, persuaded Rama to watch sec a movie

The LP rules are specified as

$$P_2 < P_3$$

meaning that the category P_2 must precede the category P3 in any local tree containing P_2 and P3 as daughters. For example, the LP statement

$$V < NP$$

states that the category having the feature V (ie [+V,-N]) must precede the category NP (ie [-V,+N] in every local tree containing both of them as daughters.

The major advantages of decoupling the ID and LP relations in a phrase structure rule are that GPSG can express free word order related facts and head parameter related information independently and succinctly.

2.3.2.3 Metarules

Metarules are rule templates that are used to generate new ID rules from an existing base ' set of ID rules. Metarules help in relating rules which are closely related to one another

through some syntactic mutations. For example the phrase structure rules for active voice and passive voice sentences are minor variants of one another. Similarly, the interrogative sentences are minor variants of simple assertive sentences. Formally, metarules are functions that take a single ID rule with a lexical head, called a *lexical* ID rule, to sets of lexical ID rules. Every metarule has an input pattern and an output pattern. The input pattern of a metarule consists of a mother category, at most one daughter category and a distinguished multiset variable W. W ranges over multisets of daughter categories. If an ID rule matches the input pattern specification of a metarule under some assignment of the multiset variable W, then the metarule generates an ID rule corresponding to its output pattern. For example the metarule in figure 2.5 takes as input the base rules given below and generates the corresponding new ID rules.

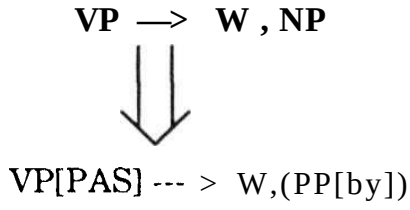

FIGURE 2.5 : A metarule

Base rule	Generated rule
VP —> H [2],NP	VP [PAS] —> H [2], (PP [*by*])
Rama *liked sita*	Sita was *liked by Rama*
VP —> H [6],NP,PP	VP [PAS] —> H [6],PP, (PP *[by])*
Rama *found the toy in a box*	The toy was *found in a box (by rama)*

2.3.2.4 Projecting ID rules into Local Trees

The notion of projection is very important to GPSG as this helps in establishing a connection between the relations encoded in the ID rules, such as domination, subcategorization and case, and the non local linguistic relations such as agreement and filler-gap relations. The base ID rules and the ID rules generated after the application of metarules are projected to local trees. A Local tree is projected from an ID rule by mapping the categories in the ID rule into their legal projections which are allowed by the *universal feature instantiation (*UFI) constraints. In the example in Figure 2.6,

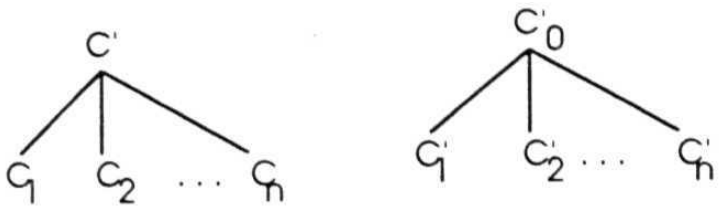

FIGURE 2.6 : A Local Tree and its projected Tree

the category C'_i, for all i from 1 to n, projects C_i A category X is said to be a projection of the category Y, if

- every feature in Y is present in X,
- all FCRs are true of X, and
- X is compatible with all the FSDs of the grammar.

After projection, the newly generated tree must satisfy all the LP-statements and the following UFl principles.

- Head Feature Convention (HFC). Informally, HFC stipulates that certain features, called head features, on the mother category must be the same as those on the head daughter category (H), if possible. In effect the HFC expresses the X-Bar Theory (section 2.2.1) in part, requiring a phrase to be the projection of its head. Head features in English are {AGR, ADV, AUX, BAR, INV, LOC, N, PAST, PER, PFORM, PLU, PRD, SLASH, SUBCAT, SUBJ, V, VFORM}.
- Foot Feature Convention (FFC). Informally, FFC stipulates that certain features, called Foot features, instantiated on the mother should also be instantiated on at least one of the daughter categories. This constraint provides a partial account of the gap-filler relations in GPSG including parasitic gaps, and the binding facts of reflexives and reciprocal pronouns. WH, RE and SLASH are the foot features.
- Control Agreement Principle (CAP). CAP is a semantically based principle, in that the underlying intuition is that predicate categories agree with their argument categories. For example, VPs must agree with their subject NPs in English. Agreement is encoded in the AGR feature which is category-valued.

As an illustration, the ID rule

VP—>H[1] ,NP

which is equivalent to

[-N,+V,BAR 2]—>H [BAR 0], [+N,-V,BAR 2]

allows the local tree shown in figure 2.7 to be projected.

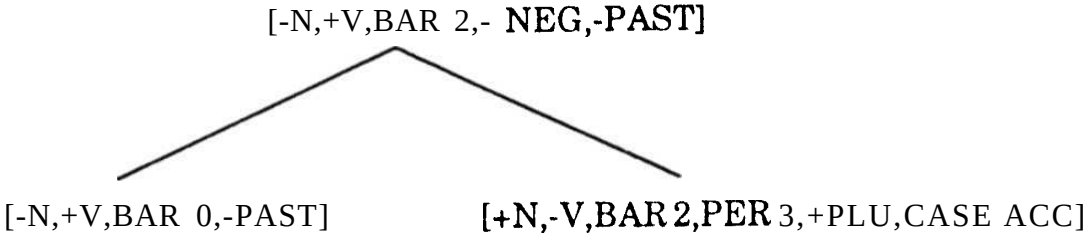

FIGURE 2.7 Local projection of a tree

The FFC and CAP together impart the right amount of context sensitivity into otherwise context free ID rules. The agreement related facts were thought to be strong cases in favour of proving natural languages to be context sensitive. Similarly, unbounded dependencies and coordination were thought to be beyond the power of context free grammars and required transformations. GPSG, by extending the non-terminal symbols of context free grammars (CFGs) with the above notion of categories and with the stipulation of the UFI constraints over these categories, has enhanced the expressive power of CFGs to tackle the above mentioned context dependencies in natural languages.

Unbounded dependencies appear in many constructions like,

Who do you think that Rama saw yesterday?	(question)	--2.6
The table on which we left the map	(relative clause)	--2.7
A movie like this, nobody ever directed	(topicalization)	—2.8

All of the above constructions have the property that there is an extra phrase outside the main clause, while within that main clause a phrase is correspondingly missing. In (2.6), the VP *saw* has no object, in (2.7) *left* is missing a locative argument and in (2.8) *directed* has no object. In GB analysis one has to generate the phrase in question within the clause, and then move it to a clause external position by some move-cx transformation, thereby ensuring a one-to-one match between the extra phrase and the missing phrase. The relative distance between these two related entities may be potentially unbounded. The GPSG analysis, on the other hand, does not use move-α kind of transformations to characterize the unbounded dependencies. GPSG proposes the following rule for dealing with many cases of unbounded dependencies.

S —> X^2, H/X^2

This rule admits a tree with any (BAR 2] category sister to an S carrying the information that it is missing a [BAR 2] category. The information of the missing constituent is carried down the tree and is ultimately associated with an appropriate null constituent *(trace).* GPSG uses a feature NULL to encode that a constituent is phonologically null or empty. An FCR in GPSG further stipulates that the feature SLASH co-occurs with the feature NULL. Consequently an empty string *e* is listed in the lexicon as belonging to the category X [+NULL]/X . Armed with this notion of empty string together with the ID rule given above, one can analyse sentences like (2.8) as follows.

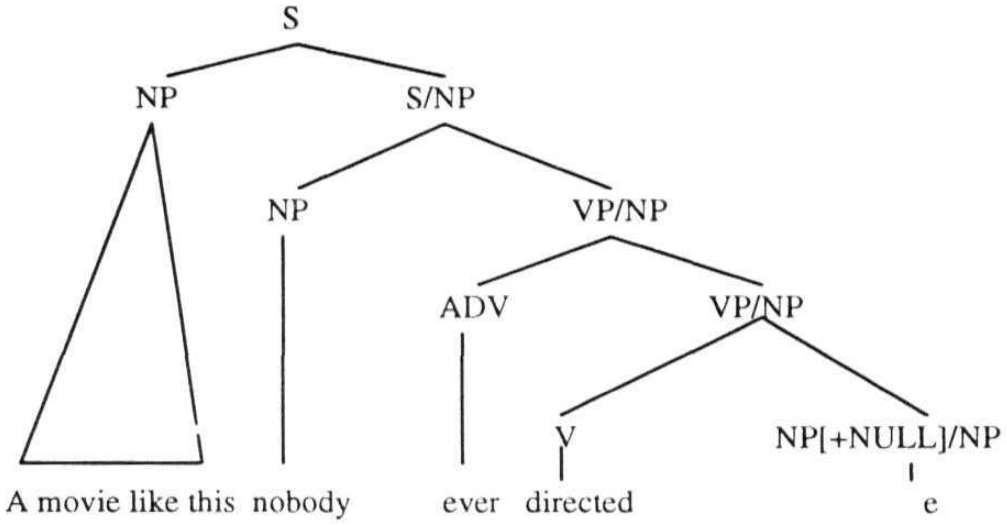

FIGURE 2.8 Use of the SLASH feature in GPSG

Semantic analysis in GPSG follows Montague's [Mont74b] method of compositional semantics. Montague used a model-theoretic approach to formalising natural language semantics. The semantic analysis in GPSG consists of (i) specifying in the lexicon the possible denotations of each syntactically determined category, and (ii) specifying along with each ID rule the manner in which the denotation of the mother category is produced as a function of the denotations of the daughter categories. For example, in the simplest form, the denotations of various words is as follows:

the has the denotation $\lambda P \lambda Q.\ 3X\ [\ P\ (X)\ \&\ Q\ (X)]$,

boy has the denotation $\lambda X.$ boy' (X) and

sleeps has the denotation *XP.* P *(XX.* sleeps' (X)).

The ID rules for S and NP have associated denotation combining rules as follows,

S --> NP, VP VP^1 (NP'), meaning apply the denotation of NP (NP') to the

denotation of VP (VP')

NP --> DET, H DET' (H'), meaning apply the denotation of H (H') to the denotation of DET (DET).

The result of applying the NP ID rule is the generation of a sentence fragment

the boy

which has the denotation

$$\lambda P.\ \exists X\ [\ P(X)\ \&\ Q(X)]\ \{\lambda X.\ \text{boy}(X)\}$$

After carrying out the functional application in the above formula, one gets

$$\lambda Q.\ \exists X\ [\ \text{boy}'(X)\ \&\ Q(X)]\ .$$

The result of applying the S ID rule is to generate the sentence

the boy sleeps

whose denotation is

$$\lambda P.\ P(\lambda X.\ \text{sleeps}(X)).\{\lambda Q.\ \exists X\ [\ \text{boy}'(X)\ \&\ Q(X)]\ \}$$

After carrying out the functional applications in the above formula, one gets

$$\exists X\ [\ \text{boy}^1(X)\ \&\ \text{sleeps}'(X)]$$

Thus specifying a close a link between the ID rules, their denotations and the denotations of the lexical entities, GPSG handles the semantics of natural languages very elegantly in a mathematically precise form.

GPSG theory thus accounts for many phenomenon like unbounded dependency, agreement, coordination using enhanced phrase structured rules and proves that transformations are not necessary for providing a satisfactory theory for the syntax of natural languages. From a linguist's point of view GPSG provides an excellent tool for the investigation of the syntax and semantics of a language. However, from the Computer Scientist's point of view, it poses many computational and implementation problems. The meta rules and the feature system of GPSG can lead to computational intractability. It has been proved by Ristad [Rist86] that computational complexity of

the universal recognition problem for GPSG is EXPOLY[3] Modifications have been indicated by Ristad [Rist86J for reducing this complexity of GPSG by altering it marginally.

The two features, SLASH and SUBCAT, used in GPSG are marginally useful for formulating a grammar for Telugu. The SLASH feature of GPSG theory is not very well suited for languages exhibiting multiple gaps in clauses. Telugu is such a language. The SUBCAT feature of GPSG takes numerical values as shown in examples ID1 to ID 10 above. By using numerical values, GPSG unnecessarily hides many of the common features among verbs and also makes the feature SUBCAT atomic and unanalysable. Thus *Akaamksha* as presented in Chapter 1, cannot be modelled using the SUBCAT feature of GPSG. In addition, as may be seen from the same examples, the SUBCAT feature of GPSG does not include subcategorization for the subject of the verb. This omission renders the SUBCAT specification of GPSG less useable for Telugu, and subtlely imposes, a requirement for the existence of phrase structure in the language. As described in Chapter 1, the notion of phrase structure is not well suited to Telugu. The free-word orderdness of Telugu can not also be expressed adequately if the subject is not subcategorized for. In addition the information captured in the ID rules of GPSG is more syntactic in nature than semantic. Well formedness criteria in GPSG are dependent on syntactic subcategorization. While in Telugu, as seen in Chapter 5, semantic criteria are more preferable. A better candidate for analysing Telugu is a closely related formalism called Head-driven Phrase Structure Grammar that exploits the structure present in the SUBCAT feature and the semantics of the language.

2.3.2.5 Head-driven Phrase Structure Grammar (HPSG)

A variant of GPSG is Head-driven Phrase Structure Grammar (HPSG)[Poll84, Poll87, Sag89]. It embodies the lexicalist view to a greater extent than GPSG and answers some of the outstanding issues in GPSG through this heavy lexicalisation of the grammar. Another system of grammar called Lexical Functional Grammar (LFG) can also be categorized into this Syntactico-Semantic approach albeit to a much lesser extent than GPSG. LFG stresses the functional relations in a grammar such as subject, object and indirect object. It considers them primitive relations, in terms of which a great many rules and conditions (constraints) are stated. As HPSG borrows a lot of ideas from LFG as well, a good review of HPSG would cover both LFG and the allied area of Unification Grammars.

[3]That is the fastest recognition algorithm for GPSG can take more than exponential time.

HPSG tries to integrate a number of ideas from the diverse traditions of GPSG [Gazd85a], Categorial Grammar [Vanb87, Stee88], Lexical Functional Grammar (LFG) [Bres84], GB theory [Chom81, Chom86], and Situation Semantics [Barw83]. It embodies a mathematically precise formulation of certain principles of universal grammar whereby properties of phrases are "projected" from lexical heads and generalizations about constituent order are succinctly expressed. As lexical structures bear much of the syntactic burden in this approach, considerable attention is given to the nature of lexical information, in particular to subcategorization information and to its role in the specification of the syntax of the language.

HPSG exemplifies the family of grammatical theories popularly known as *unification-based* grammars [Shie86]. The name *unification-based* grammars arises from the algebra that governs partial information structures. The fundamental operation upon them is the *unification* operation, which yields from a set of compatible structures a structure which contains all the information present in the members of the set. Underlying all such theories is the assumption that in actual language-use situations, specification of linguistic objects (words, clauses, and their syntax , semantics etc.) comes about in a cumulative monotonic fashion via the interactions of the constraints arising from several sources, including lexical entries (which contain phonological, syntactic and semantic information), the grammar rules that specify how the lexical information should be combined, language-specific and language universal principles of well-formedness and the particular context of the utterance. The final result of the interaction is obtained by unifying [Shie86, Sag86] the information from all these sources. The theories are declarative in the sense that they characterize what constraints are brought to bear during language use independently of what order the constraints are applied.

The partial information pertaining to linguistic objects is encoded in feature structures or *signs.* A typical feature structure in HPSG has the following fields (features) PHON(ology), SYN(tax) and SEM(antics) and is written in a matrix form shown in figure 2.9.

PHON	rAmuDu
SYN	MAJ N
	AGR 3S
SEM	*rama*

FIGURE 2.9 A typical Feature Bundle in HPSG

To specify that the value of the feature MAJ which is part of the feature SYN is N, one customarily writes SYN [MAJ N. The PHON feature is used to specify the phonology

or the surface form of the word or the sentence under study. The SYN feature is used to specify various syntactic properties of the entity in question and the way in which it combines with other entities. The typical features in SYN are LOC (al), and BINDING. LOC contains HEAD, LEX and SUBCAT information and encodes information pertaining to local trees in the GPSG sense. BINDING is used to encode non-local information. The SEM feature captures the semantic content on the entity. Feature structure in addition to encoding information of lexical entities as above, can also encode information about phrase rules and other related constraints. This makes features structures a knowledge representation language. An important feature field that is used to describe phrase structure rules is the DTRS (daughters) feature which encodes the daughter categories (RHS of an ID rule) that constitute a mother category (LHS of an ID rule). The DTRS category in turn consists of a number of categories including HEAD-DTRS (head daughters), COMP-DTRS (compliment daughters), FILLER_DTRS (filler daughters), and CONJ-DTRS (conjunct daughters) categories. Using these SYN, SEM and DTRS categories all the basic properties of a language are modelled.

Like GPSG, HPSG uses feature structures as partial functions from features to values. Unlike GPSG, features in HPSG can encode disjunctive, implicative and negative information [Poll87]. Another distinctive speciality of the features in HPSG is that they can *share values.* Feature value sharing is used to encode the flow of information amongst various information bearing entities like SYN and SEM features, head daughters and the complement daughters, anaphors and their referents, entities related to one another through agreement relations and so forth. It is the shareablity of feature values that gives the expressive power to HPSG and other unification based formalisms [kay85, Sag86, Shie86]. Feature structures that have this property of sharing are called as *structure sharing* features. An example of a *structure sharing* feature structure is given in figure 2.10. HPSG uses matrix notation to depict features.

PHON	*tin*	
SYN	HEAD	[MAJ V]
	SUBCAT	<NP[1],NP[2]>
SEM	REL	EAT
	EATER	[1]
	EATEN	[2]

FIGURE 2.10 An FB which has structure sharing

In the above feature structure, the two noun phrases NP [1] and NP [2] in the SUBCAT feature are shared. Consequently the value of the feature SEM I EATER is same as the value of the first NP in the feature SYN | SUBCAT. The value of the feature SEM | EATEN is the same as the value of the second NP in the feature SYN | SUBCAT. The sharing of structure values is indicated by the numerals [1] and [2]. The sharing of information is achieved through the *unification* [Sag'86] of the feature values.

In HPSG, verbs and other lexical items that head phrases, bear a lexical specification for the feature SUBCAT which is category-valued. The SUBCAT feature takes as its value a list specification corresponding to the various complements (including subjects, and NPs within possessive phrases) that the word in question combines with in order to form a grammatically complete (or saturated) phrasal projection such as a sentence, NP, PP or AP. The order of elements in the SUBCAT list does not necessarily correspond to surface order, but rather to the order of relative obliqueness, with more oblique elements appearing later than less oblique elements (i.e. to the right). The correspondence between the obliqueness order and the surface order of the complements is achieved through some language specific rules/principles. PP complements are treated as more oblique than NP objects, and NP objects are treated as more oblique than subjects. Thus the SUBCAT list for a transitive verb contains two NPs, the left most corresponding to the object NP and the right most corresponds to the subject. The SUBCAT list for an intransitive verb contains exactly one NP, corresponding to the verb's subject. The SUBCAT feature a ditransitive verb like "give" contains two NPs and one PP. For example,

sleep	SUBCAT	<NP>
eat	SUBCAT	<NP,NP>
give	SUBCAT	<PP(to), NP,NP>

In the above examples, the preposition of the PP is indicated along with the PP.

The satisfaction of the SUBCAT specification replaces X-bar theory as the fundamental principle underlying the construction of headed phrases. More precisely, the Subcategorization Principle requires that heads combine with complements in such a way that the SUBCAT value of a given phrase is obtained by cancelling one member from the end of the head daughter's SUBCAT list for each complement actually appearing in the phrase. The sub-categorization principle ensures that a simple sentence like "Rama ate food" has a syntactic tree diagram shown in figure 2.11 (wherein all other information other than SUBCAT has not been shown).

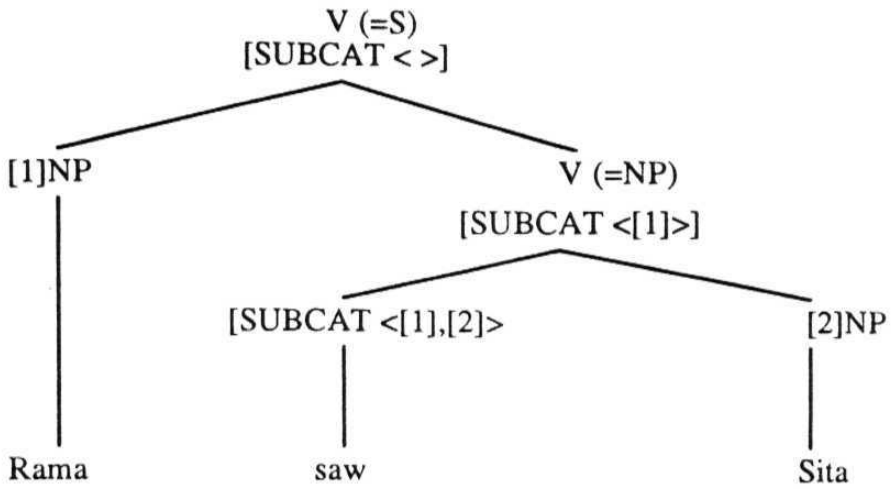

FIGURE 2.11 Syntactic tree for the sentence "Rama ate food".

As in traditional tree diagrams, each non-terminal node represents a constituent of the phrase in question with pre-terminals corresponding to lexical constituents. The numerals in square brackets, as before, indicate pieces of information required to be structure-shared owing to some linguistic constraints, in the present instance, the Subcategorization Principle (SCP). Thus individual lexical items may impose conditions of various sorts on their subcategorized complements, including their subject, and these conditions are enforced by the SCP. The SCP can be coded in feature structure, very succinctly, as shown in figure 2.12.

SYN | LOC | SUBCAT [2]

HEAD-DTRS | SYN | LOC | SUBCAT append([l], [2])

DTRS

COMP-DTRS [1]

FIGURE 2.12 Subcategorization Principle

In figure 2.12, it may be noted that for two lists L1 and L2, the formula appcnd(Ll, L2) indicates the list obtained by concatenating the two lists in the given order. Thus the above feature structure entails that in any *sign* consisting of a headed structure, the SUBCAT value is the list obtained by removing from the SUBCAT value of the head those specifications that are satisfied by one of the complement daughters. In addition the structure sharing indicated by [1] and [2], entails that the information from each compliment daughter is actually unified with the corresponding subcategorization specification on the head. Thus a *sign* can satisfy a subcategorization specification on

some head only if it is consistent with that specification; otherwise a unification failure will result in violation of the Subcategorization Principle.

The information subcategorized for in HPSG is not limited to the syntactic categories only. Each constituent in addition to having the SYN feature also has information relevant to determining the phrase's semantic interpretation as already indicated before. The information encoded in the SEM feature, in case of verbs and predicative adjectives, contains their predicative structure. The contents of SEM may be viewed as a rough analogy of G-theory's LFs, where argument positions are indicated by labelling the semantic roles rather than positionally. The Semantics Principle and the Head Feature Principle (HFP) ensure the identity between a lexical head and its phrasal projection. Role assignment is achieved by making the contents of the subcategorized NPs identical to (structure-shared with) the fillers of the corresponding roles in the contents in the verb. In Figure 2.10 the predicative structure of the verb *tin* (eat) was given. The interpretation of the SEM feature in Figure 2.10 is that (due to structure sharing) the 'eater' slot of the *tin* (eat) verb is filled by the subject of the verb and the 'eaten' slot is filled by the object of the verb. In the case of a control verb (eg. persuade, ask, want) the content of a VP or predicative complement is associated with an argument position in the control verb's semantic content.

The information pertaining to the agreement features number (NUM), person (PER) and gender (GEND) of nouns, called INDEX in HPSG formalism, is encoded as part of the SEM information. INDEX information is useful for referential purposes. If a NP is referential, then any NP coindexed with it must have the same INDEX. Since the agreement features of this nature belong to the internal structure of the indices, it follows immediately that coindexed NPs (such as anaphors and its antecedents) necessarily bear identical specifications for person, number and gender. By contrast CASE is treated as part of the category, and not as part of INDEX. Thereby it follows that case concord is not required by Coindexing.

As has been seen in the preceding paragraphs, the SUBCAT feature is essentially *local* in nature. In a sense its specification is determined by lexical features and transmitted by HFP. The information it contains does not flow beyond the maximal projection of the lexical sign containing it. The unbounded dependencies, as explained in the previous section on GPSG, are non-local in nature. In order to handle such long-distance dependencies, HPSG uses the category valued feature BINDING in its SYN feature. BINDING consists of three categories, SLASH, REL and QUE. The SLASH category in HPSG, as in GPSG, is used to mediate the flow of information between a gap and its filler. The REL feature transmits information about relative pronouns up to the point in

the structure where the relative clause is combined with the antecedent noun. The QUE feature is used to ensure that information about interrogative elements is propagated up to the interrogative clause or phrase that the element in question scopes over. To see the SLASH feature in action, consult the figure 2.12 which gives the syntactic analysis of the example sentence (2.8),

A movie like this, nobody ever directed,

discussed in the previous section on GPSG.

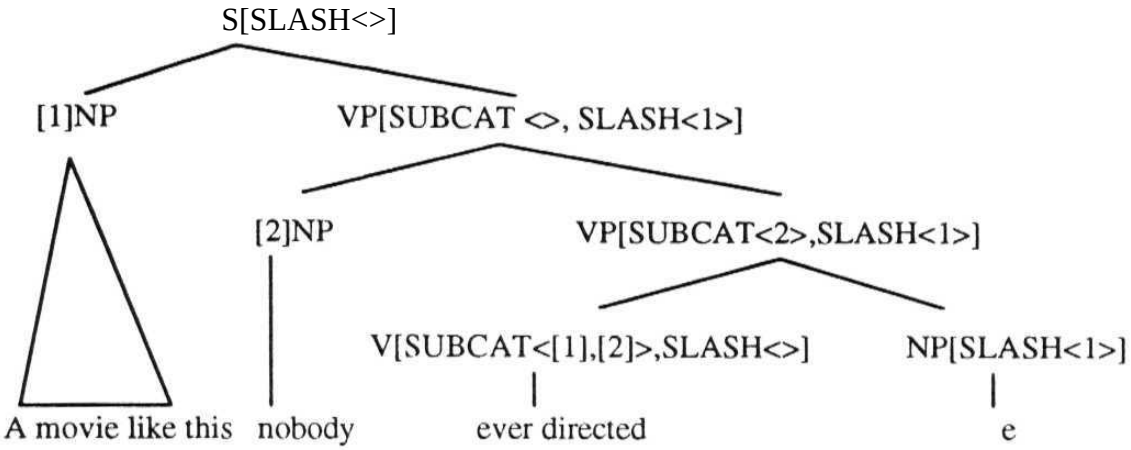

FIGURE 2.13 Usage of the SLASH feature in HPSG

The HPSG is currently under development and many issues regarding adjuncts, and quantification are not fully resolved. Handling comparatives and superlatives is also not properly developed in HPSG. Computationally viable methods to handle co-ordinate structures also needs to be developed. Data base access in natural language cannot do away without handling such complexities of language usage. Hence to use HPSG theory for developing a natural language understanding program one needs to fill in all the above gaps.

2.3.3 Mathematical Approach

The mathematical approach to NLP is totally different from all the other approaches in both its origin and emphasis. The epitome of this approach is Categorial Grammar. The term 'categorial grammar' was introduced by Bar-Hillel [Barh64, pp 99] as a handy way of grouping together some of his own and others' research work [Ajdu35] towards a particular method of linguistic analysis based on phrase structure grammar. This method is based on type theory which Russell had introduced to fend off some foundational problems in set theory. In this model every element of the vocabulary of a natural language belongs to one or more categories, and each category is either a basic category or a derived category defined in terms of simpler categories in a way which fixes the

combinatorial properties of complex categories. Let us take the subsets A, B, C of a semigroup M that are subject to the following three operations: ., / and \

(i) A.B ={ x.y E M | x ∈ A A y ∈ B }

(ii) A/B ={ ZE Ml ∀ x ∈ B, Z.X e A} and

(iii) A\B ={ z ∈ M | ∀x ∈ B, x.ze A}

Then it can be seen that , for all A, B and C c M,

(iv) A.B E C iff A ∈ C/B

(v) A.B E C iff B ∈ C\A

Let us relate the above to natural language. If the operator "." is considered as a concatenation operator, then one can look at the semigroup of the freely generated words of the language under concatenation. Sets of strings of natural language words are called *types* or *categories.* There are two primitive categories *e* and *t.* Semantically *e* stands for entities and *t* stands for truth values. Syntactically *e* stands for noun phrases (and nouns) and *t* stands for sentences[4]. The sentence *Rama sleeps* has type *t. Rama* has type *e* being a noun, then *e.{sleeps}* must be equal to *t.* Therefore *sleeps* has the category *t/e.*

There are certain operations possible on categories to generate new categories. The operations possible on categories are 7 ' and '\ '. These operators, also called **combinators,** '/' and '\' compose functions out of categories 'a' and 'b' : a/b and a\b[5] whose domain is 'b' and range is 'a'. With these operators, an infinitum of such categories including *t/e, (t\e)/e, e/e, e/ (t/e), (t/e)\(t/e), (e\e)/tcsin* be generated. If *e* is assumed to denote a noun, *t* denotes a sentence, / denotes "taking an argument to the right of" and \ denotes "taking an argument to the left of, then with respect to English, *t/e* refers to an intransitive verb (ie. a function which if supplied on its right with a noun *(e),* gives a sentence (t)), *e\t/e* denotes a transitive verb[6], *e/e* refers to a pronoun and so on. A homomorphism between the concatenation of well formed substrings (strings of the regular expression {e,f,/,\}*)[7] and sentences of natural languages is, thus, established.

[4]In Russell's type theory there is only one primitive type, i.e. set

[5]In older categorical grammar literature these would have been written as a/b and b\a

[6] This implies that a verb is treated as a function whose domain is {e x e} and range is t. It takes two arguments, one occuring to its right and another to its left, and gives a sentence.

[7] is the Kleene operator

Using the above mapping, one can visualise how a typical sentence can be parsed. An example is shown in figure 2.14

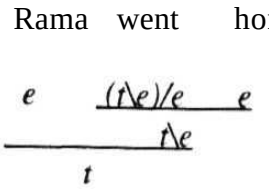

FIGURE 2.14 Sentence analysis by CG

In figure 2.14, the function word 'went' which is of category *(t\e)/e* combines with the argument word on its right, 'home', of category *e* to give the new category t/e. This function *t\e* in turn combines with the argument word to its left, 'Rama', to yield the category *t.* Thus a sentence is parsed.

It can be easily seen that certain combinations like *e\e* have no corresponding standard grammatical categories. However, they do play a role in parsing, as shown in figure 2.15.

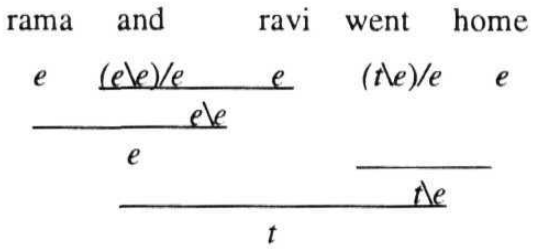

FIGURE 2.15 Handling co-ordination CG

The notion of category here is very much different from the notion of categories in GB theory, GPSG and HPSG. In this notion of category, complex and infinite categories can be formed out of simpler categories. Only a handful of the categories in CG can be mapped into the categories of the X-bar system (including major and minor categories), but after that no correspondence can be seen. If residual categories of HPSG (those categories whose SUBCAT is non-empty) are interpreted as categories, one sees a closer link between CG and HPSG.

In the earlier paragraphs the syntactic interpretation of the categories *e* and *t* and the operators "/" and "\" was explained. There are a few more operators such as type-raising operators, and combinators [Stee88, Stee91] that expand the syntactic arsenal of CG theory. All these extra operators were invented to explain various phenomenon like fronting of NPs, filler gap dependencies, and coordination [Stee85]. The important

contribution of CG theory lies in its non-transformational, and non-SLASH oriented approach to handling unbounded dependencies. The notion of a variable is totally absent in CG theory.

Semantics in CG is handled through semantic composition schemes tailor made to each operator. Semantic interpretations can be given to the CG operators and the categories *e* and *t* in the same vein as it was shown in the case of GPSG theory. As operators are applied to analyse a sentence, so are the semantic compositions applied to generate the meaning representation of the sentence.

The main problem with CG is that due to type raising, and due to the presence of multiple choices in combining operators, much back-tracking is needed to parse sentences [Witt91]. The computational complexity of syntactic recognition in CG is O (n^6) [Vija91]. There does not seem to be much work done in the area of applying CG type of analysis to non-configurational languages. Non-configurational languages like Sanskrit, Telugu, and Japanese show a wide variation in word order, and hence would unnecessarily entail too many type raising-ambiguities such as choosing between, to type-raise or not, to use a combine rule or to use an apply rule and so on. Thus CG grammar is unsuitable for analysing non-configurational languages efficiently. But, if CG grammar is combined with the notion of unification, and *samarthah* theory, then this combination can handle free-word order phenomenon better.

There are a few other variants of the mathematical approach which take an entirely different route to the analysis of natural language syntax. They do not deal with the semantics of natural languages. Notable among these is the use of word and category co-occurrence probabilities gathered from textual corpora for the analysis of NL sentences [Chur88, Mage90]. Some others have tried to formulate the problem of parsing NL sentences as solving sets of integer equations. The approach of Bharati [Bhar90a, Sinh88] tries to blend the *karaka* theory described in the first chapter with solving sets of integer equations to parse simple sentences of Hindi. Most of these approaches have been tried on a small scale and not enough information is provided in literature on their suitability for analysing complex sentences such sentences with relative clauses, coordinate clauses, and comparative complements.

2.3.4 Pragmatics Based Approach

The basic tenet of this approach is that pragmatic factors like, context, discourse structure, intention and common knowledge between the speaker and listener are more important than the mere syntactic knowledge of a language in comprehending the meaning of a sentence. This approach relies more on the semantic and pragmatic contents

of words to understand a sentence and uses syntax, if al all, in a peripheral manner. This approach has been advocated by Artificial Intelligence researchers like Wilks [Wilk75] and Schank [Scha75]. Their work was extended by many other workers in the later years, without essentially altering the spirit of the approach.

Wilks's preference semantics system [Wilk75] takes an Al approach to both the syntax and semantics of natural languages. Wilks mostly concerns himself with word sense and structural disambiguation. To achieve this, semantics is used heavily. Wilk's preference semantics parser [Wilk75] uses *semantic primitives, formulae* and *templates* for both the analysis of the text and for the representation of the text after analysis.

Semantic primitives are concepts into which other more complicated concepts can be decomposed. Typical examples of semantic primitives in Wilks' system are MAN (human being), STUFF (substance), CAUSE (causes to happen), and THING (an entity); qualifiers like WELL, and MANY; and cases like SUBJ (subject of an action), OBJE (object of an action), and INST (instrument of an action). The advantage of using primitives is that many constraints on the well formedness of sentences can be expressed in terms of these primitives. These constraints are the constraints that can not be expressed through syntax.

Formulae consist of primitives enclosed in left and right brackets. They represent a word sense. The following is a formula for the word sense of the word 'eating':

```
((ANI SUBJ)
((SOLID STUFF) OBJE)
(SELF IN)
((THIS (ANI (THRU PART))) TO)
(MOVE CAUSE))
```

This reads as "causing to move a solid object (SOLID STUFF) by an ANImate agent, into that same agent (containment case indicated by IN, and SELF refers to the agent) via (direction case) an aperture (THRU PART) of the agent". Formulae of this sort are associated with every content word of the language.

Templates are sequences of formulae. They are based on the structure of the typical messages that occur in normal texts. For example the template actor-action-object, indicates that "some actor did some action on some object". Thus sequences, templates, encode a possible message in the text. The purpose of a parser is to uncover such templates in text, and from them build the overall semantic structure of the entire textual passage. Due to this reliance on semantics, word sense disambiguation becomes easier. For example, in the sentence

The old man the boats.

when represented by a string of formulae, will give two sequences owing to the two senses of the word man (verb sense and noun sense).

The old man the boats

MAN MAN THING

MAN ACT THING

If the basic template in the system was MAN-ACT-THING then automatically the second sequence would be chosen by the system, and hence the word man would be disambiguated to its verb sense.

Templates are bound together by *paraplates* to form larger chunks of knowledge. These *paraplates* represent the meaning of a segment of text. Common sense inferences and reasoning are often used to guess the correct paraplate suiting a sequence of sentences or clauses. For example, the word 'salt' may mean 'an old soldier' or ' the substance Sodium Chloride'. MAN will be the category marker for the 'an old soldier' sense of salt and STUFF will be category for the alternate meaning 'Sodium Chloride'. By specifying some simple common sense rules to relate templates together to form paraplates, the number of templates attached to some textual fragment can be reduced to one. For example, the sentence, " The old salt is damp, but the cake is still dry", may have the following two template sequences.

(PI) MAN + BE + KIND and STUFF + BE + KIND

(P2) STUFF + BE + KIND and STUFF + BE + KIND

In the absence of any overriding considerations, the common sense rule that in a textual passage topic shifts occur only at text boundary points like paragraphs and sentences, and under the assumption that repetition of the category marker STUFF is more probable, the second template sequence (the correct one) is chosen.

Wilks's system distinguished itself from the earlier selection-restriction based systems [Katz63, Katz64] in two ways. Firstly in disambiguating word senses, Wilks's system weighed the "semantic density", or in other words, the number of satisfied preferences in the reading of a sentence. Secondly, by viewing selectional restrictions as preferences rather than as absolute conditions for match, Wilk's system was able to handle metaphors. An example handled by Wilks's system was,

My car drinks gasoline

This sentence would be rejected by Katz-Fodor-Postal [Katz63, Katz64] analysis because of the violation of the selection restrictions, the action of drinking is performed by only animate agents. In Wilks's system, on the other hand, while an animate agent is preferred, the only available subject (machine) is accepted anyway so that a reading is produced for the sentence.

The major problem with Wilks's system is that it has no checks against combinatorial explosion. This stems from the two principles mentioned above. Firstly, the system bases its final choice for a preferred interpretation on weighing the "semantic densities" assigned to competing interpretations produced by the system during analysis. This enforces a breadth first parsing strategy to be employed by the analyser, as all the alternate parses need to be evaluated before one of them is chosen. Secondly, to cater to the extended usage of language (like metaphor), no interpretation of any fragments should in principle be discarded before a final decision is made, even though these interpretations may contain semantic preference violations. As a result, the system inevitably runs into the problem of combinatorial explosion.

An interesting contrast to this breadth-first approach is the one employed in TEAM [Gros87] and in PROGRAMMAR [Wino72]. Winograd uses a simple strategy of backtracking. In his own words, the way his program treated word sense ambiguity "is not through listing all possible interpretations of a sentence, but in being intelligent in looking for the first one, and being even more intelligent in looking for the next one if that fails" ! [Wino72, p 22].

The other variant to the approach of Wilks is that of Conceptual Dependency (CD) theory of Schank [Scha75]. CD theory is different from Wilks's approach due its emphasis on "expectation and memory based parsing" of text as against "template" based parsing. According to the CD protagonists, the pragmatic content of an entire episode or story dictates what should be ultimately contained in the individual sentences of the story or the episode. The purpose of a parser is to guess the pieces of the episode to which words are pointers. From these pointers, and the episode fragments guessed, expectancy information about remaining text and episode is generated. This expectancy and the rest of the words in the text are matched to see the well-formedness of the sentences and paragraphs. At a higher level, that is at the paragraph/episode/story level, expectancy is expressed in terms of various memory resident data structures called Memory Organization Packages (MOPs), and Thematic Abstraction Units in Memory (TAUM)s [Dyer83]. At a lower level, that is word and sentence level, expectancy is represented

through Frames, the knowledge representation proposed by Minsky [Minsky75]. Frames represent the Conceptual Dependency graphs.

A CD graph is a relation between primitive objects that are either actions, states, or noun-like "picture producers". CD theory is based on two principles [Rich83]:

- CD representation should allow effective inference, by associating a fixed set of inference rules with each CD primitive
- CD representations should be independent of any particular human language.

To illustrate the first principle, if given the following sentence

Rama shot an arrow with his bow.

one would like to tell from the representation that before the event took place, the arrow was in the hands of Rama, and afterwards it was far away, whereas Rama was where he was.

To fulfil the second principle Schank [Scha75] outlined a short list of primitive actions and showed how to represent sentences as graphs built from these primitives. For example, the CD primitive PTRANS, which stands for "physical transfer/translation", indicates motion of a physical object from one place to the other place. The various components of the CD graph are related using the following predefined cases or "slots"

actor	The initiating agent of an action
object (O)	The thing affected by an action
inst (I)	The instrument or means of the action
from (D)	The source location of action
to (D)	The destination location of action

So the above sample sentence would be represented by the CD graph in figure 2.16. The different kinds of arrows and their corresponding superscripts, indicate the above five conceptual dependencies. The double arrow indicates the actor of the PTRANS.

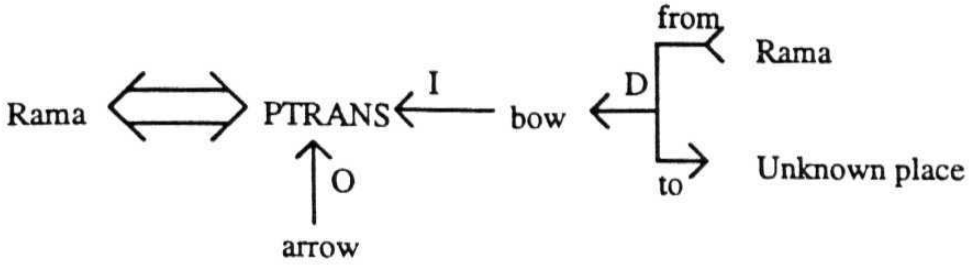

FIGURE 2.16 A typical Conceptual Dependency diagram

This literally means " Rama physically transferred the arrow from his proximity to some unknown place using his bow as an instrument". The graph notation is cumbersome to use and many authors use the following frame like representation using brackets:

```
(CD    (ACTOR      (rama))
       (<=>        (PTRANS))
       (ACTOR      (rama))
       (INST       (bow))
    .  (FROM       (rama))
       (TO         (unknown place)))
```

CD graphs are not only independent of the surface form of a sentence, but also canonical. Thus no matter how a sentence is phrased or what language is used or what words are used, it should be represented in the same CD graph. For instance all of the following sentences should be represented with the same CD graph given above.

An arrow was shot by Rama from a bow
Rama launched an arrow using a bow
Rama propelled an arrow using a bow

రాముడు విల్లునుంచి బాణాన్ని వదిలాడు
rAmuDu villunuMci bANAnni vadilADu
Rama shot an arrow from the bow

The canonical representation achieved through CD graphs makes CD representation useful for machine translation. CD graphs can be thought of as Inter Lingua. One of the early applications of CD theory was, in fact, in machine translation. However, subsequent research using CD theory was aimed at in-depth understanding of texts. The first convincing demonstration that real texts covering diverse topics could be understood automatically was by Gerald DeJong's FRUMP program [Dejo79]. FRUMP could read and correctly understand about 10% of the stories on the UPI news wire and generate short, one or two line summaries of them in five different languages.

FRUMP uses data structure called Scripts for understanding texts. Scripts encode information about what happens typically in some situation. For example the Script for visiting a restaurant would describe what people typically do when they eat out; go to the restaurant, order food, cook prepares food, waiter brings food, people eat food, people pay to the waiter for the food, and finally people leave the restaurant. This script is represented as follows,

(RESTAURANT SCRIPT
(PERSON goes to RESTAURANT)
(PERSON orders FOOD)
(COOK prepares FOOD)
(WAITER serves FOOD)
(PERSON eats FOOD)
(PERSON pays WAITER for FOOD)
(PERSON leaves RESTAURANT))

This script allows one to handle two difficulties in language understanding. The first is anaphora; how can one tell what do pronouns (in a set of sentences) refer to. In the following sentences,

Rama went into a restaurant.
He ordered idles.
The waiter served Rama hot idles.
Rama paid him 10 rupees.

The pronoun 'he' in the second sentence refers to Rama. Whereas the pronoun him in the last sentence refers to the waiter. Resolving pronoun references is made easy due to the presence of the script. The script tells the role played by each word in the sentences. Thus pronouns can be related to the 'actors' of the script easily.

The second benefit, accruing out of scripts, is that there is implicit communication and inferencing when scripts are filled. The four sentences in the situation match the first, second, fourth and the fifth lines of the restaurant script. Thus one can say that the food was prepared after Rama ordered it, in response to the question "when was food prepared?". Similarly, it can also be inferred that Rama ate the food in the absence of contradicting information.

Thus using scripts one understands natural language texts in depth with out really using expensive common sense reasoning or inferencing. The first parser based on this idea, SAM (for Script Applier mechanism) [Scha77, Scha81] was able to parse very complex

sentences using scripts. SAM used CPU time proportional to the number of scripts. Thus it got slower while it got smarter.

The FRUMP system used clever data structures to index scripts and was able to process input texts in time logarithmically proportional to the number of scripts in the data base. A description of the FRUMP's [Dejo82] parsing strategies illustrate the basic approach adopted by Schank's CD theory based parsers [Dyer83, Cam89]. The main feature of all these text analysis programs is to skim the input text to determine its main themes without examining in detail the relationships between each and every word. FRUMP was able to successfully analyse 10% of the new stories on the UPI news wire, requiring on an average 20 seconds per story. DeJong [Dejo82] claimed that FRUMP was theoretically capable of understanding 50% of the UPI wire. The word "successfully" should be understood to mean that FRUMP would be able to understand the main theme.

FRUMP uses a top-down, tree-based approach to analyse input text. The resulting parser is very fast, because it does not try to find every possible analysis of the text. Instead, the parser always seeks to confirm a top-down expectation based on the scripts available to the system. FRUMP splits parsing into two interacting units, the *predictor* and the *substantiator*. The predictor uses a discrimination net [Rich83], called a Sketchy Script Initiator Discrimination Tree (SSIDT) to propose a slot filler for a frame and the substantiator uses inference and textual analysis to confirm or disconfirm the proposed slot filler.

The parser scans a sentence left to right trying to find some structure building word, typically a verb, for which it has a CD frame. Once such a cue word is found, the predictor builds a partial conceptual dependency graph indicating the word found. Once a partial structure is built, the top-down component directs the rest of the processing. The next step is to locate a role filler such as actor, instrument and recipient, for the CD frame. The predictor generates a request to fill the slot with one of a long list of suitable slot fillers such as HUMAN, VEHICLE, and EDIBLE. Such a list is generated by looking at all the CD frames that are potentially admissible using the cue word. For example, if the cue word is 'drop', the CDs could correspond to the following senses

(1) somebody dropping an object from a height,

(2) a company's share dropping on some share market index,

(3) somebody dropping somebody from a list

(4) somebody dropping his candidature in some election

and so on. So a typical request filler list for the ACTOR slot of dropping could be, HUMAN, SHARE. The request to fill the OBJECT slot could contain the list HUMAN, PHYSICAL-OBJECT, SHARE, and BODY-PART. The tree data structure, SSIDT, contains all such lists indexed on the slot names of the CD frames. The structure of the SSIDT can be thought of as shown in figure 2.17.

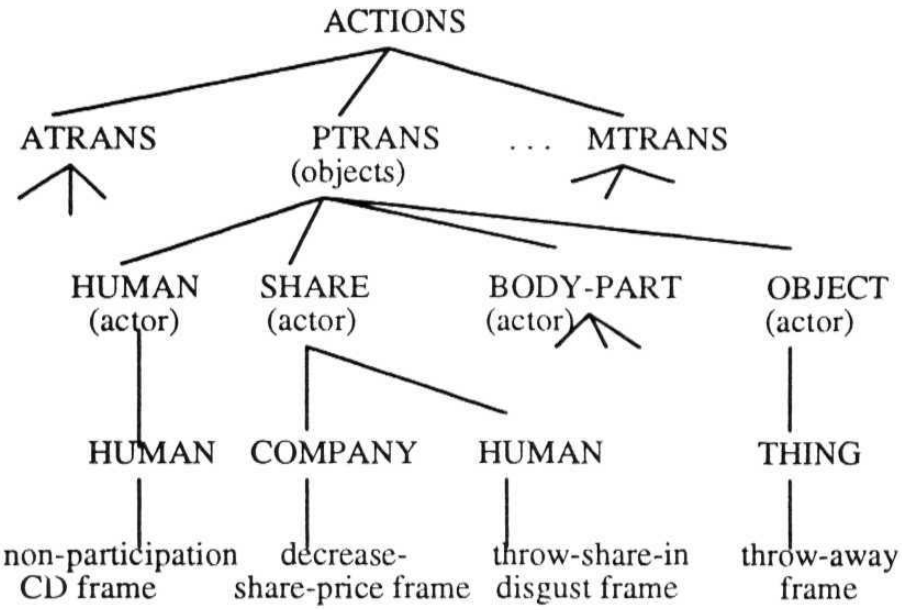

FIGURE 2.17 A graphical representation of the SSIDT

The SSIDT classifies the key portions of each script by its CD primitive action ACTION, ACTOR, OBJECT and so on. A portion pertaining to the ACTOR, and OBJECT discrimination is shown in figure 2.17. For example, if the verb is 'drop', then the PTRANS branch is taken. Then the predictor can determine that the object of the action must be a HUMAN, SHARE, BODY-PART or an OBJECT. Assuming that the object is a SHARE then the predictor has two choices for the ACTOR.

The predictor passes on the request list to the substantiator along with the slot name. The substantiator has the onus of finding the filler satisfying the requested type of entity in the rest of the text. Looking at the voice (active voice or passive voice) of the verb, the substantiator looks for the filler either to the left or to the right of the cue word. Once such a word satisfying the filler requirements is found, it is used to disambiguate the possible CD frame to go with the two cue words. The interaction between the predictor and the substantiator proceeds till all the slots are exhausted. Even after reaching such a point, if still the CD frame is not uniquely identified, then the text segment is supposed to be inherently ambiguous. For the narrow domains of usage for which many systems are put to, the CD frames are almost always resolved uniquely.

FRUMP also has a higher level of episodic knowledge called "issue skeletons" to connect scripts together and to allow one script to trigger another script for further processing of the text. In this fashion the conceptual representation grows from the kernel built from one verb to the episodic representation of the entire text.

FRUMP uses knowledge implicit in the scripts to assist in text analysis, so that ambiguous word senses are never even considered unless they are relevant. Thus FRUMP circumvents much of the work needed for understanding polysemous words. FRUMP has some limitations. Lack of a necessary script can bring FRUMP to a grinding halt because FRUMP does not rely on generic linguistic principles for comprehension. Unknown or Complex sentence structure can also blunt the capabilities of FRUMP. Parsers based on CD theory tend to ignore many relational words like adjectives, quantifiers, determiners and quite often prepositions while processing the text. This makes CD parsers unsuitable for data base access which depends on the content of each such relational word. However, systems based on some form of CD theory can be used to develop intelligent assistants for the users of complex packages/systems like Operating Systems [Wile86] and Expert Systems

2.3.5 Connectionist Approaches

This approach is completely different from the "explicit rule formation" approaches described so far. The major thrust of this approach is on learning natural languages. The learning models proposed by Connectionist researchers are totally orthogonal to those proposed by linguists like Chomsky [Chom86] and Bresenen [Bres84J. The two major differences cited with respect to learning in Rumelhart, McCelland and the PDP research group [Rume86] are as follows:

- The goal of connectionism is not the formation of explicit rules. It is the acquisition of connection strength which allows networks of units to act as though they new rules.

- No powerful computational capacities are attributed to the learning mechanism. Rather, very simple connection strength modulation mechanisms are assumed which adjust the strength of connections between units based on locally available information at the connections.

Waltz and Pollack [Walt85] presented a model based on the integration of independent syntactic, semantic and contextual knowledge sources via spreading activation and lateral inhibition links. Their work is representative of the Connectionist effort in NLP. They considered a graph with weighted nodes and links and an iterative operation which

recomputed each node's activation level, its weight, based on some function of its current value and the inner product of the strengths of its links and the weights of the connected nodes. An activation (positive) link between a pair of nodes would cause them to support each other while an inhibition (negative) link would attempt to allow only one of the pair to remain active at any given time. The net effect was that over several iterations (spreading activations), a coalition of well-connected nodes dominated, while the ones connected to the activated nodes by inhibition links remained suppressed. This behaviour of networks was utilized in several ways to build a parser. Nodes were used to represent phrases like NP, VP and PP, and lexical items like cat, dog, and mat. A phrase structure rule like S --> NP VP was represented as a graph with three nodes, S, NP and VP, connected by two activation links, S-NP, and S-VP. Similarly activation links were established between words and their different meanings, between roles and their fillers and between the corresponding syntactic and semantic interpretations. Inhibitory links were established between nodes representing different lexical categories such as noun, adjective or verb, for the same word, between concept nodes representing the different senses of the same word, and between nodes representing conflicting case role interpretations. In short, if a node could be assigned to disjunctive categories , then inhibitory links amongst the disjunctive categories were established.

Building the activation and inhibitory links was automated. The syntactic portions of the network were constructed by a parser modelled after Kay's parser [Kay73]. The semantic and contextual information was permanently resident in memory. By including references to the context in which sentences were uttered the system was able to bias its interpretation of the sentences appropriately. The net effect of the program was that over several iterations, a coalition of well-connected nodes dominated while the less fortunate nodes, those that were negatively connected to the winners, were suppressed.

Waltz and Pollack [Walt85] illustrated their work with an interesting example. In the following sentence,

John shot some bucks

the words "shot" and "bucks" are highly polysemous and belong to multiple grammatical categories. The word "shot" could be a noun, a verb or an adjective. In the context of shooting, "shot" could be a verb meaning fire, an adjective meaning "tired" (worn out), or a noun meaning a bullet. In the context of Gambling, "shot" could mean "waste". Similarly the word "bucks" also has multiples senses. Figure 2.18, depicts the various activation and inhibitory links that can be present in the Connectionist network representing the sentence "John shot some bucks" in the context of shooting.

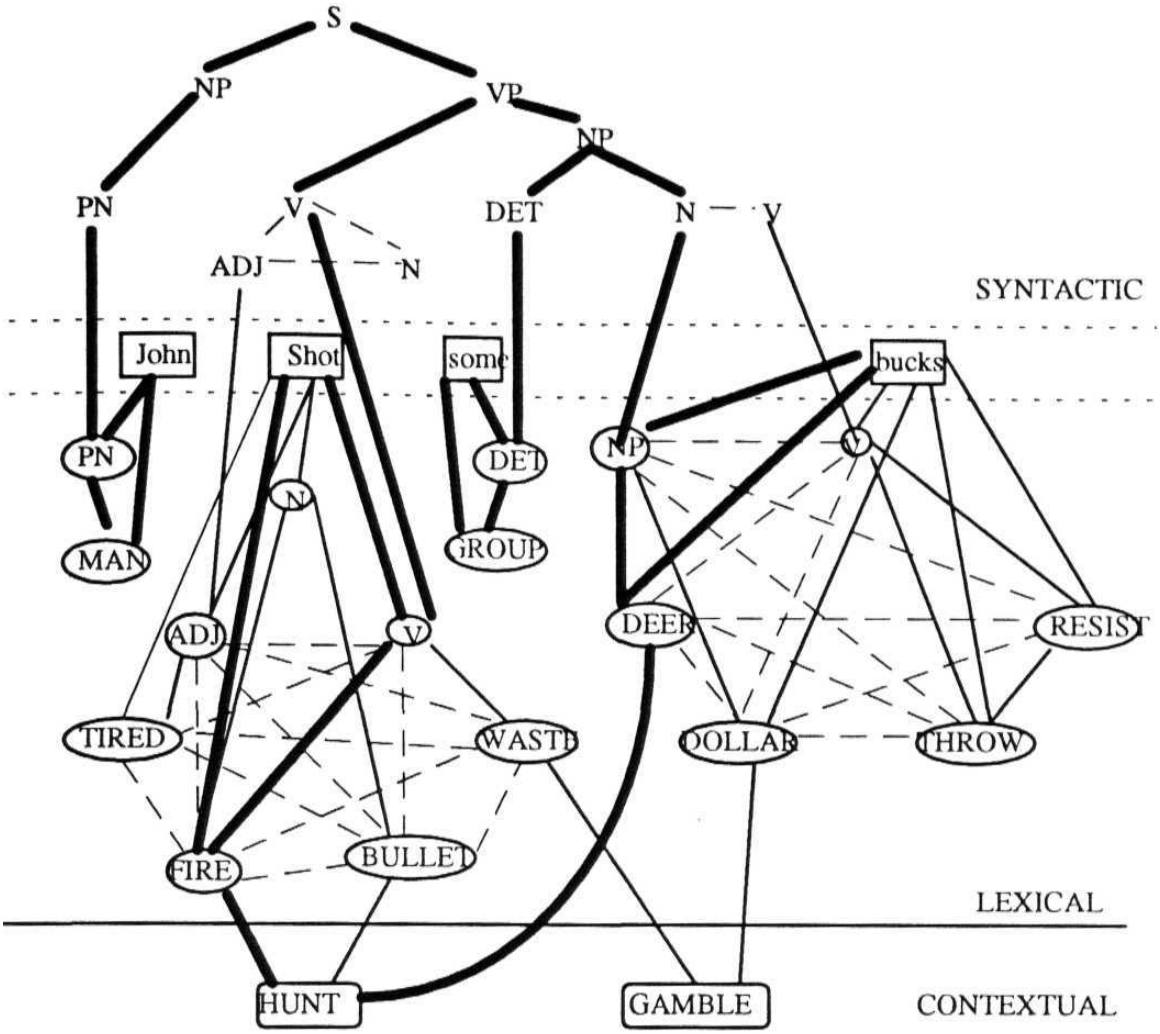

FIGURE 2.18 Connectionist representation for the sentence "John shot some buks"

Darker lines in figure 2.18 represent higher activation levels. Activation links are shown as complete lines and inhibitory links are shown as dashed lines. The figure shows the network in steady state after all the iterations from an initial state are performed.

In the context of Telugu, Waltz and Pollack's work can be interpreted as follows. The sentence " *rAmuDu sitani mulaga ceTTu ekkiMcADu"* has two meanings. Rama made Sita climb drumstick tree or Rama praised Sita, depending on the context. So, while processing the above Telugu sentence, if the context in which this sentence occurs is included as a reference, the system can bias itself to the proper meaning. If the context is given as staff appraisal the system will bias itself towards the "praised" meaning for *"mulga ceTTu ekkiMcADu",* where if the context was given to be "botanical study", the drumstick meaning would be taken.

The strength of the Waltz and Pollack's system is that it is a bold attempt in applying massively parallel processing to NLP. The model succeeds by using fairly simple mechanisms in providing an account of the semantic interpretation of the sentences. It

needs no extraneous control mechanism: its processing is completely defined by the principles of activation spreading and lateral inhibition which have been thoroughly studied by psychologists [Coll75]. It requires no additional structures as working memory or context pointers for backtracking. The constraints imposed by the sentence are taken into account globally and simultaneously by the model in its search for an interpretation. The model is also capable of graded responses, being able to represent shades of meaning by varying degrees of activation of its output.

The weaknesses of the model are as follows.

- The use of "context setting" nodes such as "employee appraisal", and "botanical study" to bias particular word and phrase senses in order to force an appropriate interpretation has obvious major problems. Many times a context setting word may never explicitly appear earlier in a piece of text even though it may be easily inferred by a reader or hearer.

- To overcome the above problem, Waltz and Pollack [Walt85] proposed the use of "microfeatures" associated with each concept node of the network. Microfeatures are features that correspond to the major distinctions humans make about situations in the world, that is, distinctions one makes regarding needs, goals, intentions, history, geography, topic, likes and dislikes, etc. The need to develop such a complete set of features to give as input bias is another bottleneck in the development of Connectionist parsers

- The Cartesian product problem in linking every word sense with every other word sense in the system is very difficult to solve.

- A proper way to handle recursions in natural language (multiple adjectives, a complement inside another complement, a sentence embedding another sentence etc) was also not developed. Other major problems with regards to gaps, co-ordination, quantification etc are also not well understood.

Many researchers [Berg92, Jain89] have tried to overcome the above four problems with the findings of Waltz and Pollack. The current effort is also making it unnecessary to explicitly assume the phrase structure, for example S--> NP VP. The Connectionist networks, after being trained over a large number of sentences, are able to extract the phrase structure from texts [Berg92]. It is interesting to note the similarities of current Connectionist approaches to the *akhandapaksha* of Indian Linguistic theories described in Chapter 1.

None of the Connectionist systems developed to date compete with systems developed using the traditional approaches. The level of syntactic and semantic complexity handled in these systems is still at toy application level only and cannot be presently used for applications such as data base access. Slightly lower end applications of NLP like information retrieval are likely to be possible with the present day Connectionist approaches.

2.4 Conclusion

The purpose of this chapter has been to both review the NLP literature which can probably have an impact on the processing of Telugu, and also to introduce the linguistic paraphernalia that will be used in subsequent chapters. Many of the ideas that work well for English and those which have relevance to Telugu have been reviewed. The strengths and weaknesses of each approach was highlighted. As and where appropriate each concept was explained with reference to Telugu also.

In the program TELANGANA developed by the author, ideas from many of the approaches described in this chapter have been borrowed freely. In the subsequent chapters, which describe TELANGANA in full detail, all the borrowed concepts used in TELANGANA are back referred to this chapter.

Chapter 3

Knowledge Representation in TELANGANA

3.1 Introduction

Every NLP system has to represent, use and maintain several kinds of knowledge like lexical knowledge, grammatical knowledge, and world knowledge, to understand natural language sentence. Linguistically motivated NLP systems [Bres84, Chom81, Gazd85a] rely heavily on grammatical knowledge, and to a much lesser extent on other kinds of knowledge. On the other hand, TELANGANA, even though linguistically motivated, relies heavily on lexical knowledge and to a lesser extent on other kinds of knowledge. This is because it is based on the *aakaamksha* (expectancy) and *yogyata* (ability) of words. As *aakaamksha* and *yogyata* encode word related knowledge only, and because such knowledge is best captured in the lexicon, TELANGANA is lexicon driven. TELANGANA also uses the notions of *sannidhi* and *samartha* which basically specify the constraints on sentence formation. These two concepts are the nearest counterparts in TELANGANA to the grammatical knowledge traditionally encoded as phrase structure rules. Parsing of Telugu sentences in TELANGANA is, thus, dependent on these concepts and their operationalization[1]. Consequently, to understand the working of the TELANGANA program and its underlying theory of *samarthah,* one needs to understand and represent *aakaamksha, yogyata,* and *sannidhi* related information.

In this chapter the representation of *aakaamksha, yogyata* and *sannidhi* is presented. The kind of knowledge that needs to be encoded in the lexicon to represent these facets of the words is explained with several examples. In the fifth chapter, the notion of *samarthah* is formalised. A viable operationalization of *samarthah* is presented there and in the sixth chapter. However, before these concepts are expounded further, to better appreciate their role in **TELANGANA,** the structure of TELANGANA program itself needs to be understood first. In **the next** few paragraphs the rationale for the present design and structure of TELANGANA is presented. After that, the knowledge representation used in TELANGANA

[1]Making a theory usable by a computer.

to represent lexical and other knowledge is presented. An understanding of the knowledge representation introduced here provides the context for understanding the scope of syntactic and semantic analysis done by TELANGANA.

The primary goal in the design and development of TELANGANA program was to achieve substantial coverage of Telugu sentences both at the level of syntax and semantics. From the beginning, the development of TELANGANA was influenced by the author's desire to make it capable of dealing with day to day Telugu as used in common news papers and journals. Newspaper Telugu is the most commonly spoken form of Telugu. It can be viewed as a colloquial form of *graanthika* (text book/literary) Telugu, and hence poses many difficult problems for NLP.

The analyses of Telugu sentence constructions provided by TELANGANA are intended to be as linguistically motivated and independent of the domain of discourse as possible, in order to ensure that TELANGANA is extensible and suitable for a range of language processing applications. Given the requirements of some of these applications, it is important that the representation of sentence meaning produced by TELANGANA should be capable of supporting machine reasoning. The general approach of analysis taken in TELANGANA is to follow the principle of compositionality whereby the literal meaning of a phrase is obtained by combining the literal meaning of its constituent phrases. The notion of constituency in TELANGANA is different from that in English, and is based on the notion of *aakaamksha* (expectancy), *yogyata* (ability), and *sannidhi* (proximity) explained in the first Chapter.

The designers of natural language interfaces to application programs, such as databases, face the problem of mapping sentences in the natural language to the database specific commands. The semantic gap is quite often so wide that it is not possible to generate the desired **end** result in one step. Hence NLP researchers have found it convenient to process the input sentences in several stages. Following this principle of "division of labour", a modular staged design was adopted for TELANGANA in which explicit intermediate levels of linguistic representation are used as the interface between successive stages of analysis. The final result of such analysis is a set of fully specified *logical forms* that represent the possible literal meaning (semantics) of the input sentence. In addition to the usual scientific and engineering benefits of modularity, the above approach allows the intermediate stage representations and outputs in TELANGANA to be useful for other applications such as machine translation, automatic proof reading and style checking.

3.2 Structure of TELANGANA

In TELANGANA, there are three kinds of sentence representation and four stages of processing. Each stage of processing takes as input one kind of representation of the sentence and produces another kind of representation as output.. The last stage produces the output to the user. The four stages of sentence processing in TELANGANA are:

- Morphological Analysis,
- Syntactic/Semantic Analysis,
- Quantifier Scoping and
- Question-Answering.

The three kinds of sentence representation are:

- Morpheme-Sequences,
- Feature Bundles and
- Logical Form.

The various stages of processing and their inputs, and the relationship between the stages and the different kinds of sentence representation are summarized in Figure 3.1.

The Morphological Analyser (MA) in TELANGANA uses morphological rules and a lexicon for analysing the input sentences. The lexicon contains root words and their categories. The MA takes as input the sentence typed in by the user, analyses morphologically each word of the sentence, segments it into root word and suffixes and finally outputs a list of words in the form of root word + suffixes. This morpheme sequence in the list form constitutes the first kind of representation of the sentence. The suffixes and root words are chosen to ease further processing of the input sentences. For example, when the input sentence

రాముడు సీతకి వనంలో చాలా పువ్వులిచ్చాడా ?
rAmuDu seetaki vanaMlO cAlA puwuliccADA ?
Did Rama gave many flowers to Sita in a garden ?

is analysed morphologically, the following output is generated.

[rAma+none, seeta+ki, vanaM+10, cAlA, puvvu+lu, iw+ADu+A]

The character strings occurring to the right of + symbols are the suffixes with which the root words were combined in the original sentence. The suffix "none" indicates a null suffix like *Du* in *rAmuDu* whose presence is an idiosyncratic feature of Telugu. If instead of the word

rAmuDu the word *rAdha* were present in the input sentence, then the output would have consisted of rAdha+none instead of rAma+none. The word *cAlA* can never have a suffix, hence no suffix is indicated for it.

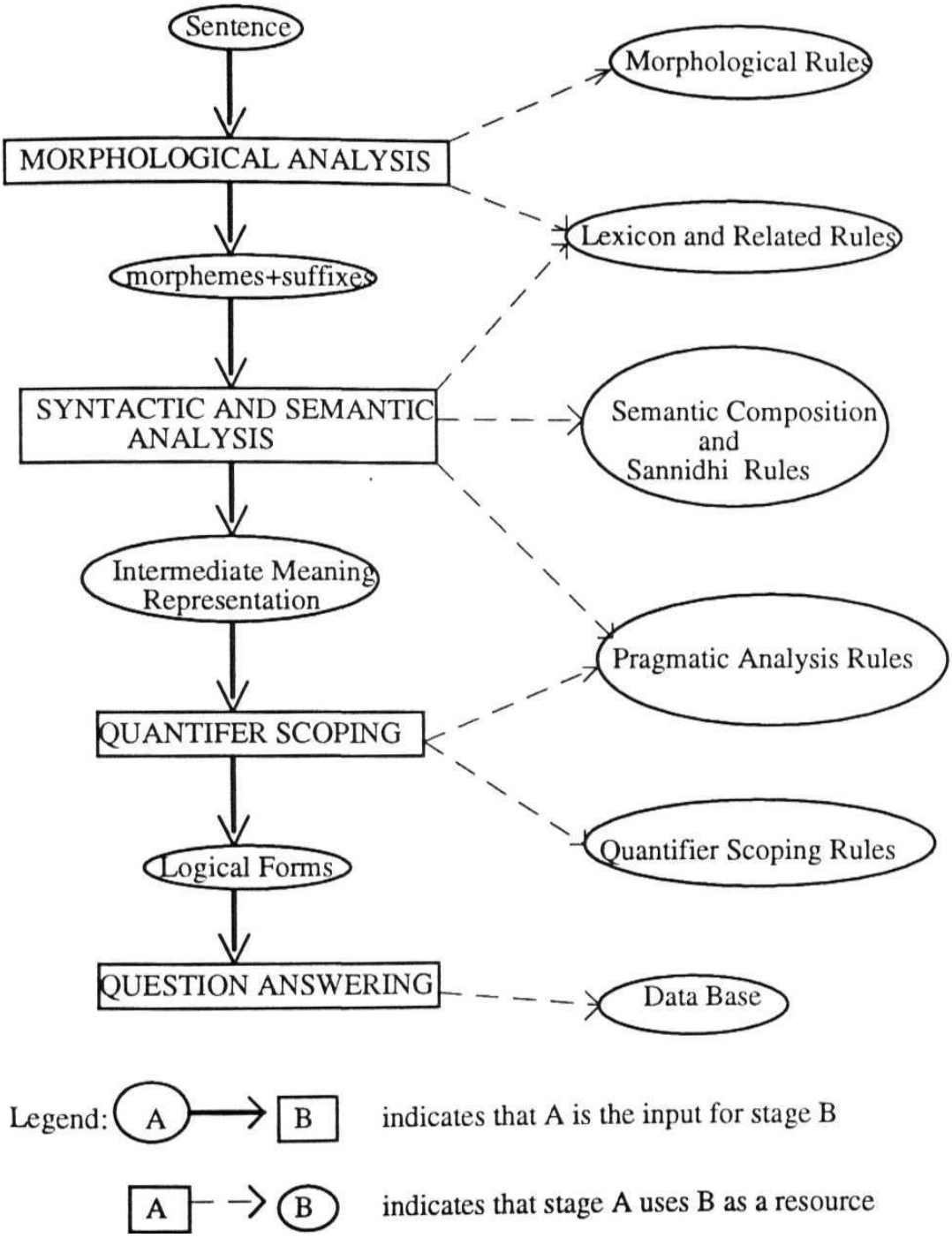

FIGURE 3.1 TELANGANA System Diagram

The next stage of processing, the syntactic and semantic analysis, forms the heart of TELANGANA and is crucially dependent on the *aakaamksha*, *yogyata* and *sannidhi*

information encoded in the lexicon. This stage takes as input the morpheme sequence generated by the first stage and analyses it for syntactic and semantic correctness based on *samarthah* theory. The meaning of the input sentence is output by this stage in the form of an attribute-value (feature-value) matrix like data structure commonly used in Artificial Intelligence [Mins75] and Computational Linguistic [Gazd85a] research. One key feature of this attribute-value matrix is structure sharing [Kay85]. Here, the unit of this intermediate level of representation is called a Feature Bundle (FB). FBs are specified in a language called Meaning Representation Language (MRL) unique to TELANGANA. FBs represent the meaning of a sentence in a variant of the situation calculus framework [Barw83]. To generate the FB specific to a sentence, this stage makes use of the lexicon which contains *aakaaMksha* and *yogyata* information, *sannidhi* (proximity) rules and pragmatics related rules. Pragmatics related rules exploit the details specific to the domain of the database in order to disambiguate vague combinations like noun-noun, and possessive-noun words. The output of this stage of processing, for the above sentence, is as follows (only the important portions of the FB are shown for pedagogical reasons)

```
[ aak: [],
  word_loc:6,
 sentype: [none,yn],
 var:X0,
 yog: [pos:v, isa:kriya, agr: [ [m,s,31,_]],
 sem: [      reln:      ivv
        saMdarbham: X0,
        dAta:            [aak: [],
                          word_loc:l,
                          var:Xl,
                          yog: [ pos:n,isa:human,agr: [ [m,s,3],_]],
                          sem: [ reln: name,id: X2,string:rAma]],
        graheeta:        [aak: [],
                          var:X2,
                          word_loc:2,
                          yog: [pos:n,isa:human, agr: [ [f,s,3],_]],
                          sem: [ reln: name,id:X2,string: seeta]],
        padArthaM:       [aak: [],
                          word_loc:5,
                          var:X3,
                          yog: [pos:n,isa:physical_object, agr: [ [n,pl,_], [indef,cnt]]],
                          sem: [ reln:puvvu,id:X3]],
                          quant: [aak: [],
                                  word_loc:4,
                                  var:X3,
                                  yog: [pos:quant,isa:_,agr: [_. [indef,cnt]],
```

```
                    sem: [rcln:number,id:X3,quant:cAlA]]],
sthalam:        [aak: [],
                 word_loc:3,
                 var:X4,
                 yog: [pos:n,isa:location,agr: [ [n,s,J,J],
                 sem: [ reln:vanaM,id:X4]],
]]
```

FIGURE 3.2 The FB for the sentence *rAmuDu seetaki vanaMlO cAlApuvvuliccADu*

The above FB captures the linguistically significant literal meaning of a sentence and is unbiased towards any specific application. The various features such as aak, yog and sem and their corresponding values in the above FB represent different aspects of the syntax and semantics of the example sentence. For example, in the above FB the feature "aak" has a null value. It signifies that this sentence docs not have any unsatisfied *aakaamksha.* The feature "word_loc" indicates the position of the head word of the FB[2] in the sentence as six. The full significance and the generic use of various features, such as aak, yog, and sem are detailed in the next section.

The representation of a sentence as a FB, however, is not directly useful for database applications. Database applications follow Model-Theoretic semantics [Gall84, Ullm88, pp 78-79] and hence are deeply dependent on the use of linguistic quantifiers like "every", "some", "many", "a" and "the". The third stage of processing is incorporated to assign proper scopes to the quantifiers and to get the denotation of sentences in terms of the database and other auxiliary relationships. It should be noted that FBs can be used for translation purposes rather easily. A machine translation system, for translating Telugu sentences into other languages can be built by stripping off the third and fourth stages of TELANGANA and adding a suitable transfer phase as in the XTRA [Huan85] and MU [Naga86] machine translation systems.

The third stage of processing (described in Chapter 6), quantifier scope assignment brings out the relative significance of quantifiers in the input sentence. This phase of TELANGANA is what makes it database specific. Prior to this stage (except for the pragmatic rules) all the stages of processing are application neutral. The output of this stage consists of a logical form corresponding to the input sentence. The logical form corresponding to the example sentence is

```
[yn,
  [past,
      q (ex,  X,  saMdarbhaM (X),
```

[2]In the given example sentence, it is the word ivv.

```
q(ex, Z, vanaM (Z),
  q (caala, puvvu (Y), ivv (X, rAma, seeta, Z, Y))))]]
```

meaning "does there exist, in past, a samdarbham (event) X and a garden (vanaM) Z, such that for many (caala) Y such that Y is a flower (puvvu), Rama (rAma) gave Sita (seeta) Y in Z". The symbols yn, past, ex, and caala stand for the sentence type (yes-no), the tense of the sentence (past), the quantifier (there exists), and the generalized quantifier caala (many) respectively.

Typically the logical form related to a sentence is subsequently fed into some filters [Boug85, Gros87] to generate a corresponding data base query. The final module of TELANGANA, the Question Answering (QA) module, converts the queries in logical form to the actual database oriented queries; Prolog queries here. The Prolog equivalent of the input sentence is

```
number_of (X, puvvu(X), N),
number_of (Y, (puvvu(Y),ivv (rAma,seeta, vanaM, Y)),N2),
N2>=N1/2.
```

which *asks* the underlying Prolog system "if there exist two numbers N1 and N2 such that NI is the number of flowers in the garden and N2 is the number of flowers given to seeta by rAma, such that N2 is more than half (i.e meaning of *cAlA*, many) of N1". The Prolog predicate "number_of" is predefined in the QA module.

In the ensuing sections MRL, FBs and LFs are described in detail. In the subsequent chapters the different stages of processing are described in detail.

3.2 Meaning Representation Language (MRL)

Entries in the lexicon and the FBs are encoded in MRL. MRL is a variant of Prolog lists, and captures the intuition behind the situation semantic characterization of knowledge representation [Barw83]. A situation (an event or a state of affairs) is represented by a list of features and fillers (values) written as feature:value. Features represent the type/kind or property of a concept, whereas the filler represents the "value" of that kind/type information or the value of the property.

For example, the information that

Rama is a six feet tall human being

is represented as the following FB

[isa:human, name:rama, length: [measure:6, units:feet]].

Let X stand for a concept/entity represented by the word "Rama". Then the "isa" feature, in the above FB, captures the kind of entity X is. The feature "name" captures the name property of the entity X. The feature "length" captures the information about the length of the entity X. The length information itself is represented as a FB where the feature "unit" indicates the units of measurement of length, and the feature "measure" indicates the number of such units that make up the length of X. This frame like representation [Mins75] allows many types of linguistic knowledge, including lexical information, phrase structure constraints, co-occurrence restrictions, and linear precedence to be represented easily. The main difference between Frames and MRL is the addition of *unification* [Sag86] in MRL. Unification lets linguistic constraints to be placed declaratively on the values of features. MRL is implemented in Prolog by suitably declaring the corresponding "op" (Cloc81] for the symbol ":".

The FBs convey an intermediate level of representation between the surface form of a sentence (syntax) and its deeper internal representation, logical form, which is used here for question answering. A FB cannot be a rewording or a rehashing of the original text itself, otherwise generation of the corresponding LF would require a detailed knowledge of the syntax of the source language and the associated parsing techniques. Further, FBs should be compositionally generatable. That is, if in a sentence two words, X and Y, are present and they have *samartha,* then the FB of X and Y put together must be a composition (either functional or relational) of the FB of X and FB of Y. This constraint makes it computationally plausible to generate the logical form of very complex sentences. This is the well known compositionality principle followed by Montague [Mont74a, Mont74b]. Montague, however, insisted on functional composition only.

Central to these constraints on the generation of the FBs is the idea that human beings can understand both partial sentences and totally new sentences once they know the meaning of smaller portions of the sentences. These constraints also simplify the computational model by localising the interaction between FBs. In Chapter 5 and 6, the methods employed in TELANGANA for generating the FBs in an incremental fashion from the syntax of Telugu are presented.

3.3 Representing Lexical Information

The lexicon is the repository of word knowledge in the system. It is the most important part of TELANGANA. The parser in TELANGANA depends on the lexicon and a few constraints of sentence formation for analysing sentences. The lexicon contains information regarding the syntactic properties and semantics of words in addition to the information pertaining to their *aakaamksha* (desire/expectancy) and *yogyata* ('ability). Words are used in many senses in any natural language. Therefore a FB is associated with each sense of a word in the lexicon. In TELANGANA, Telugu words are categorized in terms of their parts of speech (pos): nouns, verbs, pronouns, quantifiers and adjectives. The FB of each category is different. There are also some common features of the FBs, specifically, "aak", "lex", "sem", "var", and "yog".

The "var" feature of an FB represents a 'rigid identifier' or a variable/token signifying the concept represented by the FB. The "var" value is used in lieu of the actual concept much like a pronoun used in lieu of a noun. The "word_loc" feature value gives the position of the word in the sentence. This value is useful in assigning scopes to quantifiers (sec Chapter 7). The "sem" feature represents the meaning of the word in a relational style. The other features like "aak", "yog", "sentype" and "comp" are explained in detail in the ensuing sections.

3.3.1 Representing Akaamksha, Yogyata, and Sannidhi

Akaamksha (desire/expectancy) and the *yogyata* (ability) information is represented in FBs using the features "aak" and "yog". For some word W, the feature "aak" in its FB typically contains information about the *yogyata* (ability/nature) of the words expected to co-occur with it in a sentence. The "aak" feature also indicates the orthographic order in which those words should be present in the sentence. For example the verb *cooD* (see) has the following (incomplete) FB associated with it in the Lexicon.

```
[aak: (1: [subj: (Kl:hum+man),
           10: (Ll:loc+opt),
           ni: (Ol:obj+man)]),
yog: [isa:mental_natural,pos:v],
sem: [ rel: cooD, event:E,seer:Kl,seen:Ol,location:Ll]
]
```

Capital letters are used in MRL to indicate variables of Prolog. In the above FB, the "aak" feature has the value,

```
1: [subj: (Kl:hum+man),
```

1O: (Ll:loc+opt),
ni: (Ol:obj+man)]

where the symbol "1" (left) indicates the surface order of the words. The symbol "1" indicates that the *yogyata* words, corresponding to the three concepts K1, L1 and 01 representing a human, a location and an object respectively, should occur to the immediate left of the *aakaamksha* word *(cooD* in this example). Three other symbols, "r", "rl", and "n", can be used in place of "1". The symbol "r" (right) indicates that the *yogyata* words should occur to the right of the *aakaamksha* word. When "1" and "r" are specified, no extraneous words are allowed to intervene in between the *aakaamksha* word and the *yogyata* words. The symbol "rl" (remote left) is used to indicate that the *yogyata* words should occur to the left of the *aakaamksha* word but with possible intervening *non-yogyata* words. The symbol "rl" is seldom used in TELANGANA. The symbol "n" is used to represent lack of *aakaamksha.* The symbols "1", "r", "rl", and "n" represent the *sannidhi* (proximity) constraints.

The "sem" feature in the above frame relates the concepts K1, L1 and Ol to the 'seer', 'location' and the 'seen' roles of the "seeing" event represented by the word *cooD.* By using Prolog variables K1, L1 and Ol as values and due to the capability of Prolog to express unification implicitly, the correspondence between the surface case and deep case (seer, and seen here) is achieved. In other words, structure sharing (see section 2.2.3.5) is used to achieve the correspondence between the surface representation and the deep representation of a sentence. In the case of adjuncts, such a correspondence is not easily achieved. Hence adjunction rules are utilized (Chapter 5). It is not possible to express the interaction between verbs/nouns and their adjuncts through structure sharing alone.

The yog feature encodes *yogyata* (ability) information and indicates that the part of speech (pos) of *cooD* is v (verb) and that *cooD* is a 'mental_natural' kind of event. The "isa" feature indicates the "sort" of the entity in a sortal hierarchy/semantic net (sec Section 3.6). The feature yog also contains agreement related information indicated through a feature by name "agr". In the case of verbs, the value of the "agr" feature is set when number, gender, and person related suffixes are added to the word. In the case of nouns, such information is present as an intrinsic property of the concept indicated by the noun.

The symbols "subj", "10", and "ni" represent the surface case markers, postpositions, that need to be present on the words representing K1, L1 and 01 respectively. Usually "subj" is realized in Telugu by a phonologically null postposition. The symbol "10" is realized by the postposition *l0.* The symbol "ni" is realized by a phonologically null postposition in the case of non-human objects and by the postposition *ni* in other cases.

The symbols "man" and "opt" indicate whether the presence of the corresponding *yogyata* words in a sentence is mandatory or optional. In the above example, the "+opt" in the "10" feature of the FB indicates that a word representing the location of *cooDaTaM (*seeing) may be optionally present in a sentence in which *cooD* occurs.

It may be observed that *aakaamksha* (expectancy), *yogyata* (ability), *sannidhi* (proximity) *and karaka* (deep case) theory can be naturally expressed in terms of our FBs which have structure sharing properties. Another possible way to express *aakaamksha, yogyata* and *sannidhi* is to represent them as a system of linear equations as done in Bharati's work [Bhar90a, Bhar90b]. This way of formalising *aakaamksha, yogyata* and *sannidhi* seems to betray the naturalness and simplicity of these concepts and also does not follow the traditional linguistic principles like *samarthah.* Bharati et al have tried their approach on simple sentences only. The complexity of solving these equations grows rapidly as the syntactic complexity of sentences increases. It is difficult to visualise how their approach scales up for processing complex sentences. Thus the above presented method of formalising *aakaamksha* (expectancy), *yogyata* (ability), *sannidhi* (proximity) and *karaka* (deep case) theory as discrete data structures having structure sharing properties appears to be simpler, more natural and linguistically better motivated.

3.3.2 Defining Verb related FBs

The FB of a verb consists of the standard features like "aak", "yog", "var" and "sem" as explained above. The "var" feature of a verb represents the event or state variable, in Davidson's [Davi67] sense, associated with the verb. In TELANGANA, the specification of a verb's "sem" feature value, uses two special features "sthiti" (Stative) and "samdarbham" (event). Stative verbs have a "sthiti" feature and event verbs have a "samdarbham" feature. These features are useful in specifying the FBs of sentential modifiers like adverbs and modals. The verb *un* (exist) in the sentence,

రాముడు హాయిగా ఉన్నాడు
rAmuDu hAyigA unnADu,
Rama is happy

indicates a state of affairs and hence is a 'sthiti[1] kind of verb. The verb *cooD* in the sentence,

రాముడు సీతని నిన్న చూశాడు
rAmuDu seetani ninna cooSADu
Rama saw Sita yesterday

is a "samdarbham" verb. The "var" and "sem" features for the verbs *un* (exist) and *cooD* (see) are given below.

[var:S,
yog: [isa:natural_nonmental_suitive, pos:v],
sem: [rel:un, sthiti:S, entity:E, state:A]]

[var:S,
yog: [isa:natural_nongoal_activity,pos:v],
sem: [rel:cooD, samdarbham:S, seer:A, seen:B, location:C]]

The "isa" feature in the "yog" feature of the verb encodes the "sortal type" of the action (see section 3.6) the word denotes. The "isa" feature captures a much finer grain of differentiation in the types of actions denoted by verbs than is possible by the simple Stative/active categorization. Pragmatic analysis also relies heavily on this kind of sortal information [Dahl86].

The "agr" feature in the "yog" feature is filled at run time when a suffix indicating number, gender and person like *ADu, iMdi, AMu, Anu* or *Aru* is added to the verb. Suffixes are considered to be functions with root words as arguments. For example, the word *cooSAmu* (we have seen) is morphologically analysed to be *cooD+Amu.* Its corresponding FB, with the added "agr" feature would be,

[var:S,
yog: [isa:natural_nongoal_activity,pos:v, agr: [[_,pl,1],_]],
sem: [rel:cooD, samdarbham:S, seer:A, seen:B, location:C]]

3.3.3 Defining Noun and Pronoun related FBs

Many common nouns and proper nouns typically have their "aak" feature value set to null, i.e aak:n: []. Relational nouns like *taMDri* (father), *AdAyAm* (income/salary), and *vayassu* (age) have a more complicated structure than other nouns. They have *aakaamksha* (subcategorize/ expectancy) for another noun which in a sentence should precede them (occur to their left) immediately. For example, the underlined words (nouns) in *RAmuDitaMDri* (Rama's father), *seeta AdAyaM* (Sita's income). *rAdhavinnapaM* (Radha's submission). *prajalakOrika* (the desire of the people), *evaresT ettu* (the height of Everest), illustrate the subcategorization requirements of such nouns. The subcategorized word should have a possessive *vibhakti* (case marking) on it. The possessive case marker in Telugu can be realized at times by a phonologically null suffix. In case of *Du* ending words it is realized by the suffix i. This is indicated by assigning the value "nn_i" to the feature "post" (postposition

suffix) of the subcategorized noun. The feature value "nn_i" is interpreted accordingly, as default *i,* by TELANGANA. For example, the incomplete FB for *taMDri is*

[aak:l: [aak:n: [],var:X,post:nn_i],
var:V,
yog: [isa:human,pos:n,agr: [[m,s,3], L.cnt]]],
sem:[reln:taMdri,id: X, of:V]]

Unlike verbs, the "agr" feature for nouns can be assigned statically at the time of generating the lexicon. The "agr" feature contains two values which are Prolog lists. The first list refers to the number, gender and person information. The second list helps in establishing restrictions on the quantifiers the word can take. A typical noun like *kaaru* (car) has the following agr feature structure.

agr: [[n,s,3], [_,cnt]]

wherein the value 'cnt' indicates that only count oriented quantifiers are allowed to quantify this word. Quantifiers can act as *mass* quantifiers, such as *caalaa neeLiu* (much water,); or as *count* quantifiers, such as *caalaa kaarulu* (many cars); or as *degree* quantifiers, such as *caalaa vEDigA* (Very hot). Accordingly, the other possible values in place of "cnt" are "mas" and "deg". The symbol "mas" indicates that mass quantifiers are allowed, and the symbol "deg" indicates that degree quantifiers are allowed. Certain quantifiers like *caalaa* are ambiguous and by context could mean a mass, count or degree quantifier. When such ambiguous quantifiers are used in a sentence the "agr" feature of the quantified word is used to resolve the quantifier ambiguity.

The "_" in the above "agr" feature indicates the referential nature and the mutual combination properties of the quantifiers, determiners and adjectives allowed to be used with the word. The other possible values in place of "_" are "def, "indef" and "none". The symbol "none" is used to indicate that definite determiners, like *aa* (that/those), *ee* (this/these), can be used with this word. The symbol "indef is used to indicate that indefinite quantifiers, like *konni* (some number of), *koMta* (some amount of), can be used with this word. The symbol "def is used to indicate that definite quantifiers like *anni* (all), cardinal numbers like one, and ten and ordinal numbers like *okaTava* (first), and *aidava* (fifth) can be used to quantify the word. It also allows for the possible degenerate use of quantifiers like *anni* such as

రాముడు అన్ని పూలు కోశాడా !

rAmuDu annipoolu kOSADA!
Did Rama pluck so many flowers ?!

For most of the nouns this value is either "_" or "none" indicating that either any thing is allowed or that only definite determiners, like *aa* (that/those) and *ee* (this/these), are allowed. The real use of this feature becomes apparent when quantifier combinations are dealt with in Chapter 5.

3.3.4 Defining Adjective and Quantifier related FBs

Adjectives subcategorize for a noun occurring to their right hand side, as in *tella poolu* (white flowers) and *pedda illu* (big house). Adjectives are also predicative in nature, meaning that their contribution to the logical form a sentence is a predicate such as white(flower), big(house), and tall(person). Since they are predicative in nature, and can not occur independent of the noun they qualify, it is appropriate to view them as words with *aakaamksha* for nouns. The approach taken here is consistent with the one given in the first chapter when presenting the notion of *uthita-aakaamksha* (aroused expectancy). Traditional schools have maintained that nouns have *uthita-aakaamksha* (aroused expectancy/desire) for the adjective and that adjectives have ordinary *aakaamksha* for nouns. The notion of *uthita-aakaamksha* is similar to the notion of adjuncts. Adjectives are treated as adjuncts. The contribution of adjuncts to the overall semantics of a sentence is a predicate whose one argument is the variable representing the entity (noun or verb) they modify.

The "aak" feature of adjectives is set to subcategorize for generic nouns. The "pos" feature is set to the value "adj". As explained above, in the case of nouns, the "agr" feature in adjectives is used to limit their applicability. An additional feature, "attr", indicates the attribute of the nominal that the adjective qualifies. For example, the adjective *tella* (white) is capable of specifying the colour of an object (indicated by the noun) but not the size. Only objects which have the attribute colour may take this adjective as a qualifier. Thus, for the word *tella*, the FB contains attr:colour. This ensures that during pragmatic analysis TELANGANA rejects a sentence wherein an adjective whose "attr" feature is set to "colour" qualifies a nominal, like thought in " Sita was engrossed in white thoughts". At the same time the "attr" feature must be set so that a sentence like " Sita was engrossed in pleasant thoughts" is not rejected.

The quantifiers behave very much like adjectives and hence share many features of adjectives. However, they are different from adjectives in that they are less selective in qualifying the nominal. In Telugu, unlike in English, quantifiers are highly polysemous. The quantifiers like *caalaa* could mean many, much or very depending on the context of use, as in *caalaa kadhalu* (many stories), *caalaa baMgArAM* (much gold) and *caalaa vEDi* (very

hot). In addition, quantifiers do not relate the objects and their attributes like adjectives. To cater to the polysemous nature of quantifiers, the "agr" and "sem" features of quantifiers are allowed to contain disjunctive values. Disjunction in feature values is indicated by the "/" symbol. For example the lexical FB for *caalaa* is

[var:X,
yog:[pos:quant,
agr: ([[_,pl,_], [indef,cnt]]/ [[_,s,_], [indef,mas]]/ [_, [indef,deg]])],
sem: (number (X,caalaa)/quantity (X,caalaa)/degree (X,caalaa))]

The interpretation of the disjunction is that if the quantifier *caalaa* is quantifying a countable noun like *poolu* (flowers), then its semantics is "number (X.caalaa)". If the quantifier is used in the sense of a degree adjective like *caalaa vEDi* (very hot), then its semantics is "degree (X,caalaa)". In all these cases, the variable X is ultimately unified with the "var" variable of the noun is qualified by the adjective/quantifier.

Quite often more than one determiner/quantifier can qualify a noun. Typical examples of such multiple qualifiers are *caalaa konni* (very few), *caalaa ekkuva* (lot more), *kAsta ekkuva* (little more), *caalaa kAsta* (very little), *aa konni* (those few), and *ee kAsta* (this little). Many other combinations like *konni kAsta, caalaa anni, caalaa aMta,* and *aa caalaa* are not well formed. The "agr" feature's second list value helps in specifying the constraints on the well formedness of these combinations. For example the "agr" feature of *koMta* (some amount) is

agr: [[_,s,], [indef,mas]]

The symbol "indef encodes the nature of this quantifier *koMta* in terms of is combinability with other quantifiers. The possible values in place of "indef are 'def, "none" and "_". The use of these feature values is explained in Chapter 5.

Regular adjectives like *yerra* (red), and *poTTi* (short) and comparative adjectives like *ekkuva* (more), *and takkuva* (less) allow for a degree of comparison to be specified in sentences. For example,

రాముడు సీతకంటే పొట్టివాడు
rAmuDu seetakaMTEpoTTivADu
Rama is shorter than Sita

రాముడు సీతకంటే ఎక్కువ పూలు కోశాడు
rAmuDu seetakaMTE ekkuva poolu kOSADu
Rama plucked more flowers than Sita

In the above sentences, due to the presence of the word *kaMTE,* the adjectives *poTTi* (short) and *ekkuva* (more) allude to the notion of degrees of "shortness" and "more-ness". Consequently, the first sentence is understood as "the degree of shortness of Rama is more than the degree of shortness of Sita". In the sentence

రాముడు పొట్టివాడు
rAmuDu poTTivADu
Rama is short

such a connotation appears to be missing. From a close look at the above sentence, it appears as if it means " Rama is short" compared to some standard not mentioned in the context. Hence in TELANGANA all adjectives and adverbs have an extra feature namely "comp". The "comp" feature helps in handling the semantics of comparative sentences (as explained later in chapter 6). It assumes the measurands of comparison to be numbers like height in number of feet, weight in number of kilo grams, and goodness as some number of imaginary units of measurement and specifies the relationship between those numbers. For example, for the word *ekkuva* (more), the "comp" feature is assigned the value [Y1^Y2^ekkuva (Y,Y1,Y2)]. The variables Y, Y1, and Y2 are best explained using the following example sentence,

రాముడు సీతకంటే 10 లెక్కలు ఎక్కువ చేశాడు
rAmuDu seetakaMTE 10 lekkalu ekkuva cESADu
Rama solved 10 problems more than Sita.

With respect to the above sentence, Y refers to the number of problems solved by Rama, Yl refers to the number of problems solved by Sita and Y3 refers to 10.

3.3.5 Defining Adverb FBs

The FBs of adverbs look very much like FBs of adjectives. Unlike adjectives, adverbs do not subcategorize for any thing and hence have their "aak" feature set to "n: []". The "isa" feature of adverbs contains values like "manner", and "time" which describe the type of predicative adjunct the adverb forms. Adverbs, like comparative adjectives, can be an aid in forming comparative sentences like

రాముడు సీతకంటే త్వరగా ఇల్లు చేరాడు
rAmuDu seetakaMTE tvaragaa illu cErADu
Rama reached home earlier than Sita

and hence have the appropriate "comp" feature. The FB for *tvaragaa* is

```
[aak:n: [],
var:X,
yog:[isa:manner,
      pos:adv],
comp: [Y1^Y2^tvaraga(X,Y1,Y2)],
sem:time_duration (X,E)]
```

wherein the Prolog variable E is unified with the event variable corresponding to the verb this adverb modifies. This unification is achieved at the time of adjunct analysis while parsing.

3.4 Representing Sentences

The FB for a verb and the FB of a sentence containing that verb as the main verb are almost identical. The FB for a sentence contains many features (see figure 3.2) like "var", "sentype", and "sem", in addition to "aak" and "yog" features. Usually the "aak" feature of a sentential FB is null, []. If the "aak" feature of a sentence is not null, the sentence is not well formed and is taken to be wanting some complement to fully convey its meaning. This captures the notion of a sentence as *niraakaamkshah* (one without any more expectancy, Chapter 1). The "yog" feature of a sentence is not null as sentences can have the ability to satisfy the *aakaamksha* (expectancy) of other words. The "agr" feature in the "yog" feature is used to achieve the number, person and gender agreement between the main verb of the sentence and the subject of the sentence.

The "sentype" feature is added to the FB of a verb while parsing. The "sentype" feature indicates the type of the sentence through two elements of a list, the first element indicates whether the sentence is comparative or not, and the second element indicates whether the sentence is interrogative, imperative or assertive. Interrogative sentences are further divided into

(i) yes-no type question sentences (yn),

(ii) which/who/what/how question sentences (wh) and

(iii) how many type question sentences (whn).

The sem feature encodes the meaning of the entire sentence.

Another important feature associated with a sentential FB is the "adjun" feature. This encodes the information about the adjuncts of the sentence. Adjuncts usually lake the value of the "var" feature of the sentence as an argument [Davi67]. The other features that may appear in sentential FBs are miscellaneous features like "causes", "if", "before" and "after", which relate one sentence to the other sentence.

3.5 Logical Form (LF)

LF mediates between the way an end user thinks about the information in a database, as revealed in queries to the database, and the way the information can be retrieved from the database through quires in a formal language. The LF corresponding to a sentence represents the meaning of the sentence in a model theoretic frame work and thus helps in database access. Generating a LF corresponding to a sentence from its FBs helps in clearly demarking the scopes of the constituent quantifiers and thus takes the user's natural language query a step closer to the corresponding database access query.

In literature, several logical forms have been proposed corresponding to different levels of coverage of English [Gros87, McCo82, Moor81, Schu84J. Frequently, some of the decisions taken to cover a certain variety of syntactic and semantic constructs of the input natural language become inadequate as the coverage of natural language is increased. The various LFs proposed in literature (mostly for English) have spanned the English language subsets ranging from simple sentences (not containing adverbials, tensed verbs, and mass terms) to complex sentences, containing such semantically difficult things as mass terms, tense and aspect. The LF for Telugu was developed with roughly the same coverage as for the English in the TEAM [Gros87] system. This LF is used by the question-answering module to generate the corresponding database query and then to query the database. In this section the LF used in TELANGANA is explained.

The LF employed in TELANGANA is first-order logic extended by some intensional and higher order operators, and augmented with special quantifiers for definite determiners, comparative adjectives, and question determiners like *evaru* (who) and *enni* (how many). The LF corresponding to a query is generated after resolving quantifier scope related issues and by further domain dependant massaging of the FB corresponding to the query sentence. The predicates and the terms in the LF for a particular query are derived from the information in the FB of the query and the conceptual schema of the database.

The LF does not contain syntactic information like part of speech, agreement, *aakaamksha*, and "sentype". However, it contains enough information necessary to extract the final database oriented meaning of the sentence without necessitating reconsultation of the original source text. The construction of the final database oriented query could frequently involve pragmatics and recourse to the data model (in the case of Data Bases) or world knowledge (in the case of text understanding systems). The LF thus captures 'the intent of the sentence in context' [Moor81]. If ambiguity is present at the sentence level, it would not be preserved at the LF level. Different readings of the sentence should be captured through different LFs.

In the following sections different kinds of Telugu sentences are considered and their LFs are described. These sections make it easy to understand the input-output relations in TELANGANA from the Morphological analysis to the Question-Answering stages. Some differences in sentence formation in Telugu and English are also described to give a better picture of Telugu syntax.

3.5.1 Nouns and Verbs

In the following, common nouns behave as predicates with one argument. For example, cat is represented as cat (X). Proper names are represented as constants. For example, the name *rama* is represented as rama. Verbs form the heart of a sentence in Telugu. A verb introduces a predicate whose arguments are contributed by noun phrases and postposition phrases. In the following examples, logical forms for verbs are given without noun phrases and preposition phrases expanded. They will be dealt with in the next section.

శివ ఇంటికి వెళ్తాడు
siva iMTiki veLltADu
Siva goes home
go(siva, home)

రాముడు సీతని చూచును
rAmuDu seetani coocunu.
Rama sees Sita
see(rama, sita)

రాముడు సీతకి బొమ్మ ఇస్తాడు
rAmuDu seetaki bomma istADu.
give(rama, sita, toy).

The denotation of *cAla* (many) as used in the above quantified formula, is given below in a general setting . In the following formula

q (many, X, r (X), b (X))

the quantifier, "many", is interpreted as equivalent to

cardinality of { r (X) & b (X) } > cardinality of {r (X)}/ 2.

3.5.4 Stative Sentences

Stative sentences containing plural nouns form an interesting subset of sentences. Their main predication would be "is" as in is (S, X, Y). This "is" predicate, in fact functions as a meta-rule/meta-predicate while assigning scopes to quantifiers and expanding FBs into LFs. The following simple sentence brings out the role of *is* meta-predicate clearly.

రాముడు మంచి బాలుడు
rAmuDu maMci bAluDu.
Rama good boy.
Rama is a good boy.

The LF for the above sentence is simply,

q (ex, S, state (S), q (ex,Y, boy (Y), goodjiuman (Y)&is(S,rama,Y)))

This LF is transformed into the following due to the presence of "is" meta-predicate.

q (ex, S, state (S), boy (rama)&good_human (rama))

The denotation of the above LF is simply

3X. {rama (X)& boy (X)&good_human (X)}

Plural nouns occurring in nominative case in stative sentences are treated as universally quantified terms. However, plural nouns occurring in accusative case in stative sentences are treated as if they are quantified by the quantifier "some".

కుక్కలు జంతువులు
kukkalu jaMtuvulu.
dogs animals,
dogs are animals. .

In the above logical forms, complications arising from tense, voice, and modalities have not been shown. Tense and voice related information can be encoded as operators over the predications of the verbs. For example,

రాముడు సీతని చూశాడు
rAmuDu seetani cooSADu
Rama saw Sita
[past, see(rama, sita)]

వాలి రామునిచే చంపబడెను
vAli rAmunicE caMpabaDenu.
Vali was killed by Rama
[passive, kill(rama, vali)]

In ordinary Database query languages, having a separate passive predicate may not be of much use, but in discourse understanding systems, a passive construct can trigger a change in focus. Logical forms could be nested as follows.

సీతకి కధలు వినటం ఇష్టం
seetaki kadhalu vinaTaM ishTaM.
Sita likes to hear stories
[present, like (sita, hear (sita, stories))]

సీత శివని ఇంటికి వెళ్లమన్నది
sita sivani iMTiki veLlamannadi,
Sita asked siva to go home.
[past, ask(sita, siva, go(siva, home))]

Logical forms which have other LFs as arguments are known as intentional LFs. In the domain of databases it is uncommon to have intensional LFs.

LFs, in TELANGANA, being close in spirit to first-order logic formulas, are designed so as to highlight the scopes of the variable they contain. A LF containing a variable V is represented as

q(quant, V, restr, body)

where quant is a universal (all), existential (ex) or generalized quantifier like *cAlA* (many), *konni* (some), and ten. The symbol "restr" is a predicate containing V as an argument. It indicates the restriction on the type of entity the variable V stands for. The symbol "body" indicates the predicate that is true about V. In the subsequent sections, at some places, the

bodies of the quantifiers are not given. It is done to indicate that the LF is in some intermediate form and needs transformations before getting into its final scoped form.

Variables and thus quantifiers enter into LFs either explicitly through the use of nouns (as explained in the section 3.5.3) or implicitly as follows. Verbs in Telugu can be *sthiti* (state) or *samdarbham* (event) oriented and describe a state of affairs or an activity/event respectively. In both cases extra variables are introduced into the LF in order to indicate the nature of the verb. Since a variable is introduced, its quantifier also needs to be introduced. For example, the LF of the following sentence

రాముడు అందగాడు
rAmuDu aMdagADu.
Rama is handsome

is

q (ex, S, sthiti (S), is (S, rama, handsome))

where, "ex" stands for the existential operator. This LF can be understood as

There exists a *sthiti* (state) S in which Rama is handsome.

For the following future tensed sentence,

రాముడు అన్నం తింటాడు
rAmuDu annaM tiMTADu.
Rama will eat food

the LF is

[future, q (ex ,E, samdarbham (E), eat (E, rama, food))]

This LF can be read as

There will be an *samdarbham* (event) E of Rama eating food.

The ability of the logical form to distinguish Stative and active verbs is useful in handling adverbs, prepositions complementing verbs and the subtle interactions between quantifiers and prepositions. A sentence of the following kind, containing the preposition *lo* (into), can refer to the event variable, to express the intent of *lo* .

రాముడు ఇంటిలో అన్నం తింటాడు

rAmuDu iMTilO annam tiMTADu.

Rama eats food at home.

q (ex, E, samdarbham (E), eat (E, rama, food) & at (E, home))

The use of the state/event variable becomes apparent if one tries to encode the following sentence.

రాముడు ఇంటిలో కానీ హాస్టల్లో కానీ అన్నం తింటాడు

rAmuDu iMTilO kAnee hAsTallO kanee annam tiMTADu

Rama eats food at home or in hostel.

q (ex, E, samdarbham (E), eats (E, rama, food) & or ([at (E, home), in (E, hostel)]))

In subsequent sections the event and Stative variables are not explicitly given, in order to improve the readability of LFs. Suitable event/state variables should be assumed by the reader.

3.5.2 Pronouns

Pronouns are replaced by their referents, if possible. Otherwise an indication, that an unresolved pronoun is left over in the LF is placed in the argument position of the pronoun. In the following example, the reflexive pronoun is replaced by its antecedent *rAmuDu.*

రాముడు తన ఇంట్లో ఒక పువ్వు చూశాడు

rAmuDu tana iMTIO oka puvvu cooSADu

Rama saw a flower in his house

[past, q(ex, H, house(H) & belongs(H, rama),
q(ex, E, samdarbham(E) &in(E, H),
q(ex, F, flower(F), see(E, rama, F))))]

3.5.3 Quantifiers

Quantifiers and common nouns introduce variables into LFs explicitly. In Telugu, the equivalent of the indefinite article "a" of English, can occur both covertly as

రాముడు చాలా మంచి బాలుడు

rAmuDu cAlA maMci bAluDu.

Rama very good boy.

Rama is a very good boy.

శివ చాక్లెట్ కొన్నాడు

siva cAklET konnADu.
Siva chocolate bought.
Siva bought a chocolate.

and **overtly** as

శివ ఒక చాక్లెట్ కొన్నాడు

siva *oka cAklET konnADu.*
Siva one chocolate bought.
Siva bought a chocolate.

In the above sentence the word "oka" literally means one, but is used in the sense of "a". The word "oka" in conjunction with the noun it qualifies, introduces an existential quantifier into an LF. The value of the "sem" feature in the FB of the above sentence, after stripping out all syntactic information, would be

[reln:kon, samdarbham:E, karta:siva, padartham:[sem:cAkleT(X),
quant:[sem:oka(X)]l]

The above partail FB is initially converted into the following formula.

kon(q(ex, E, samdabham(E)), siva, q(ex, X, cAkleT(X)))

The above formula after undergoing scoping related transformations, yields the LF

[past, q (ex, E, samdarbham (E), q(ex, X, chocolate (X), buy (E, siva, X)))l

The logical form could be paraphrased as "In past, there exits an event E and a chocolate X, such that Siva buys X". Thus Telugu quantifiers like *oka, aMdaru* (all) and *padi* (ten) quantified variables into the LFs.

The Telugu words *prati* and *aMdaru/anni* are equivalent to the universal quantifiers "each", "and", and "all" of English respectively. In the following sentences they are used to indicate universal quantification.

ఈ కొట్టులో ప్రతి బొమ్మ ఖరీదైనది
ee koTTulO prati bomma khareedainadi.
In this shop every toy is expensive.

ఈ కొట్టులో అన్ని బొమ్మలూ ఖరీదైనవి
ee koTTulO anni bommaloo khareedainavi.
In this shop all toys are expensive.

The logical form for such expressions thus, naturally involves the universal quantifier, all.

శివ అన్ని బొమ్మలూ కొన్నాడు

siva anni bommaloo konnADu.
Siva bought all the toys.
[past, q (ex, E, samdarbham (E), q (all, T, toy (T), buy (S, T)))]

There are other amorphous quantifiers (/determiners) like *caala* (many or much) *koddi, konni,* and *koMta* (few or some). Such quantifiers are treated as predicates with three arguments. The second argument is typically called a restriction and the third body. The restriction captures the kind of entity which is being quantified by the determiner and the body signifies the predicate(s), the entity is an argument of.

కొంతమంది ఇంటికి వెళ్లారు

kontamaMdi iMTiki veLlAru.
some people went home.
[past q (some, X, people (X), go (X, Y) & home (Y))]

కొద్దిమంది ఇంటికి వెళ్లారు

koddimaMdi iMTiki vtLlAru
A few people went home.
[past q (few, X, people (X), go (X, Y)& home (Y))]

పదిమంది ఇంటికి వెళ్లారు

padimaMdi iMTiki veLlAru.
ten people went home.
[past q (ten X people (X), go (X, Y)& home (Y))]

The Question-Answering (QA) module interprets the following formula

q (few, X, r (X), b (X))

as

cardinality of {r (X) & b (X)} « cardinality of r (X)

Similar denotations could be ascribed to some, many and ten. A similar LF can be given for noun phrases in objective case as well

రవి చాలా మందికి బిళ్లలు ఇచ్చాడు

ravi cAlA maMdiki biLlalu iccADu.
Ravi many people to chocolate gave.
Ravi gave chocolates to many people.
[past, q (many, X, people (X), q (some, C chocolate (C), give (ravi, C, X)))]

The denotation of *cAla* (many) as used in the above quantified formula, is given below in a general setting . In the following formula

q (many, X, r (X), b (X))

the quantifier, "many", is interpreted as equivalent to

cardinality of { r (X) & b (X) } > cardinality of {r (X)}/ 2.

3.5.4 Stative Sentences

Stative sentences containing plural nouns form an interesting subset of sentences. Their main predication would be "is" as in is (S, X, Y). This "is" predicate, in fact functions as a meta-rule/meta-predicate while assigning scopes to quantifiers and expanding FBs into LFs. The following simple sentence brings out the role of *is* meta-predicate clearly.

రాముడు మంచి బాలుడు
rAmuDu maMci bAluDu.
Rama good boy.
Rama is a good boy.

The LF for the above sentence is simply,

q (ex, S, state (S), q (ex,Y, boy (Y), goodjiuman (Y)&is(S,rama,Y)))

This LF is transformed into the following due to the presence of "is" meta-predicate.

q (ex, S, state (S), boy (rama)&good_human (rama))

The denotation of the above LF is simply

3X. {rama (X)& boy (X)&good_human (X)}

Plural nouns occurring in nominative case in stative sentences are treated as universally quantified terms. However, plural nouns occurring in accusative case in stative sentences are treated as if they are quantified by the quantifier "some".

కుక్కలు జంతువులు
kukkalu jaMtuvulu.
dogs animals,
dogs are animals. .

The LF for the above sentence, after quantifier scope readjustment ("all" outscopes "some"), becomes

q (all, X, dog (X), q (some, Y, animal (Y), is(S,X,Y)))

The above LF, after interpreting the "is" meta-rule becomes

q (all, X, dog (X), animal (X))

It is interesting to observe a minor nuance in Telugu compared to English in the formation of Stative sentences. Stative sentences in Telugu are of two types. Those in which the Stative verb (equivalent of "is" in English) is omitted in a sentence (as below),

రాముడు మంచి బాలుడు
rAmuDu maMci baaluDu
Rama good boy
Rama is a good boy

and those in which an explicit Stative verb unduta is used as below.

రాముడు ఫ్రాన్సులో ఉన్నాడు
rAmuDu phrAMsulO unnADu
Rama France-in exists
Rama is in France

In both the above cases, translation into English necessitates the usage of is, even though semantically both the usages are different. On the other hand, this difference in semantic usage of "is" is accompanied by a suitable syntactic difference in Telugu. This syntactic clue simplifies the semantic analysis of Stative sentences. It is also interesting to note that the is-meta-rule works well for Telugu Stative sentences while it may not work satisfactorily for English. This is borne by the following example sentence in English.

Rama is in France

In Telugu, the above sentence would be,

రాముడు ఫ్రాన్సులో ఉన్నాడు
rAmuDu phrAMsulO unnADu
Rama France-in exists
Rama is in France
[present-cont isin (rama, france)]

obviating the necessity of is-meta-rule as the verb *un* takes care of "is". Hence some of the techniques used in TELANGANA for interpreting FBs are tuned to Telugu and consequently are more efficient than those using straightforwardly borrowed LFs from English oriented NLP literature.

3.5.5 Comparatives and Superlatives

This class of adjectives bring out an interesting difference between English and Telugu. In Telugu a superlative is expressed as a special case of a comparative. In Telugu, thus, one cannot in a single word express superlatives like fastest, longest and sharpest. The way to express "fastest" in Telugu would be to say "faster than all". In most of the logical forms for English used by researchers [Gros87, Wood78], nobody seems to have utilized this fact. All have used LFs of the form,

[comparative-operator predicate object1 object2] --3.1

[superlative-operator predicate object1] --3.2

Catering to both degrees of adjectives separately. For English the above type of superlative LF is useful, as it captures and also represents one fundamental ambiguity that arises in English.

Rama ran fastest. —3.3

The above sentence could mean

Rama ran fastest of all the people who participated in the event. -3.4

Rama ran fastest of all events he participated in. -3.5

If there are any doubts as to the above interpretation the following extension of the above sentence

Rama ran fastest yesterday

would set aside all the doubts. The ambiguity of this type arises because, in English, object2 is optional in superlative usages. In Telugu such an ambiguity cannot arise because in Telugu (3.3) can not be expressed without mentioning Object2. One has to either say

రాముడు అందరికంటే వేగముగా పరిగెత్తాడు

rAmuDu oMdarikaMTE vEgamugA parigettADu **--3.6**

Rama all people than speedily ran.

Rarau ran faster than all people.

or else say

రాముడు అన్నిసార్ల కంటే వేగముగా పరిగెత్తాడు

rAmuDu annisArla kaMTE vEgamugA parigettADu. --3.7

Ramu all times than speedily ran.

Ramu ran faster than all times.

(3.6) and (3.7) being equivalent to (3.4) and (3.5) respectively.

In TELANGANA, the comparative operator scheme (3.1) is adopted. Hence the LF for the following comparative sentence is (forgetting past tense for the time being),

రాముడు రవి కంటే వేగముగా పరిగెత్తాడు

rAmuDu ravi kaMTE vEgamugA parigettADu

Rama Ravi than speedily ran.

kaMTE(ekkuva',

XX. XN. run (q(ex, samdarbham(E)& speed(E,N)), X), ravi', rama') —3.8

where kaMTE (more than) is used as special directive for subsequent interpretation by QA module. The interpretation of the lambda variables and the comparative operator, ekkuva', will become clear in the sixth chapter. For the following superlative sentence, LF is

రాముడు అందరికంటే వేగముగా పరిగెత్తాడు

rAmuDu aMdarikaMTE vEgamugA parigettADu

Rama all than speedily ran.

kaMTE(ekkuva',

XX. XN. run (q (ex, E, samdarbham (E)& speed(E, N)), X),

q (all, Y, people (Y)), rama[1]) --3.9

In (3.8), and (3.9) X-abstraction is used to indicate the LF-variables which will later be made explicit Prolog variables while interpreting the LF. The lambda variable X gets bound to the object1/object2 variables. The lambda-variables N and E are arbitrary Prolog variables. In a way, by using X-abstraction, it is possible to rephrase a sentence containing a comparative as two sentences with two actors conjoined with the comparative operator kaMTE. The following example highlights this point,

Rama ran faster than Sita,

can be rephrased as

The fastness of running of Rama was more than the fastness of the running of Sita.

The formula (3.9) after proper quantifier scope resolution would be equal to the following LF,

```
q (ex, N, int(N)&q (ex, E, samdarbham (E)&speed(E,N), run(E,rama)),
    q (all, X, people (X),
        q(ex, N1, int(Nl)&q (ex, El, samdarbham (N1)& speed(El,Nl), run(El,X)),
            more_than(N,Nl))))
```

The notion of lambda-variables is not present in Prolog. However, lambda-variables can be encoded very simply in Prolog as, X^p (X), to mean λX.p (X). This interpretation of "^" as *"X"* is in the mind of the programmer, and hence care must be exercised in preserving this meaning while using this notation.

3.5.6 Miscellaneous

Adjectives and Adverbs can be treated alike in a simple way. Adjectives modify nouns and adverbs modify verbs, essentially in the same way. The intent of both nouns and verbs is captured by the corresponding simple/event/state variable. Adjectives act like predicates over the nouns they qualify and adverbs act like operators over the verbs they modify. For example, an adjective as below

ఎర్ర చీర

yerra ceera
Red Saree
red (X) & saree (X)

and an adverb as below modify their arguments.

రాముడు త్వరగా వచ్చాడు

rAmuDu tvaragA vaccADu
Rama quickly came
[past, q (ex, E, samdarbham (E), come (E, rama) & quickly (E))]

Postpositional Phrases (PP) on the other hand take two arguments. The following sentences illustrate the typical use of PPs.

రాముడు సీతని ఉద్యానములో చూశాడు
rAmuDu seetani udyAnamulO cooSADu
Rama-nom Sita-acc park-in saw
[past, q (ex, E, samdarbham (E), q (ex, P, park (P), see (E, rama, sita) & in (E, P)))]

In the above sentence the PP has been used as an adjunct to the verb *cooSADu* (saw). PPs can also be modifiers to nouns, as shown below.

రాముడు బాక్సులో పుల్లని చూశాడు
rAmuDu boksulO pullani cooSADu
Rama-nom box-in stick-acc saw
[past,
q (ex, E, samdarbham (E),
q (ex, B, box (B),
q (ex, S, stick (S)& in (S, B), see (E, rama, S))))]

The approaches taken by [Bhar90a, Bhar90b, Sinh88] for machine translation from one Indian language into another, did not take into consideration quantifiers. That may be justified on the grounds that the use of quantifiers in Indian languages are similar in nature. Hence, quantifier scoping related issues could be glossed over. But the limits of such similarities among Indian languages was not stated and hence one does not know after what breadth and depth of translation do their systems breakdown. Their approaches can benefit from the LF given here when they expand the coverage of their translators.

3.6 Sortal Hierarchy

The notion of a sortal hierarchy is central to the syntactic and semantic processing capabilities of TELANGANA. Sentences, otherwise, syntactically correct may be rejected due to mismatch in the sortal type of the entities which the constituent words of the sentence denote. The sentence

రాముడు కలలు తిన్నాడు
rAmuDu kalalu UnnADu
Rama ate dreams

is syntactically perfect, but semantically and pragmatically incorrect. Instead of *kalalu* (dreams), if *paLlu* (fruits) was used, the resulting sentence would be perfectly valid. Both *kalalu* and *paLlu* are nouns but a sentence formed using one of the words is semantically correct and the sentence formed with the other is semantically incorrect. Hence part of speech information alone does not help in studying the acceptability of sentences. The ability to reject sentences on semantic grounds, or more precisely sortal grounds, can arise only from the ability to speak about the "sort" of entities words denote. Common sense tells us

that the act of "eating" is associated with an edible object and not with an abstract mental activity. Thus by associating the sortal type "mental_phenomenon" with the word *kalalu* (dreams) one can readily judge the above sentence as ill-formed.

The sortal types associated with the entities denoted by nouns, the actions denoted by verbs, the qualities/attributes of entities denoted by adjectives and the attributes of actions denoted by adverbs, should be so represented as to capture the differences and similarities between them. Efforts to define a distinct, purely linguistic level of semantic knowledge for words have failed [Dahl86]. There is no principled way of distinguishing knowledge of word meaning from knowledge in general [Mill78, Fill85J. Linguistic semantics has dealt with the magnitude and complexity of word knowledge by ignoring it [Chom81] and computational semantics has dealt with in ways which introduce many problems [Wilk83]. In these approaches, either semantic knowledge has been defined as knowledge of primitives [Wilk75] or the problem has been narrowed by domain-specificity. The view taken in TELANGANA is that word knowledge is open-ended and is based on cognitive categories. The content of word representation is derived from empirical data about the cognitive structure of the categories named by the words. The meanings of words are represented in terms of commonsense knowledge associated with the concepts indicated by words. This allows conceptual knowledge associated with words to be viewed from multiple perspectives.

TELANGANA uses a sortal hierarchy for representing the sortal categories of words. The sortal hierarchy in TELANGANA is based on a psycholinguistically motivated classification of entities [Dahl86] which was found useful for common sense reasoning. Since pragmatic analysis frequently requires common sense reasoning for proper disambiguation of word meanings and inter-word relationships [Dahl86j, the sortal hierarchy used in TELANGANA follows Dahlgrens' [Dahl86] ontology of verbs and nouns.

Unlike many NLP systems [Gros87, Peri83], the sortal hierarchy in TELANGANA is based on a cross-classification of entities. The word "party" may be classified as a social_event type of entity in the context of the sentence,

Rama threw a party,

whereas in the context of the following sentence it could mean a place

Rama can be found in Sita's party.

Hence the word "party" is both a **social_event** and a place. The ability to cross-classify concepts is seen as a fundamental ability of human beings [Hutt79, Trab85]. Following psycholinguistic research TELANGANA uses a type hierarchy which permits each concept (node) to be classified in many different ways. Figure 3.3 depicts the top level part of the sortal hierarchy used in TELANGANA. The top most node in the hierarchy is entity. An entity is classified in two ways, as either collective or individual and as either abstract or real. The vertical lines in Figure 3.3 are used to signify cross-classification and angular lines depict mutually exclusive alternate sorts.

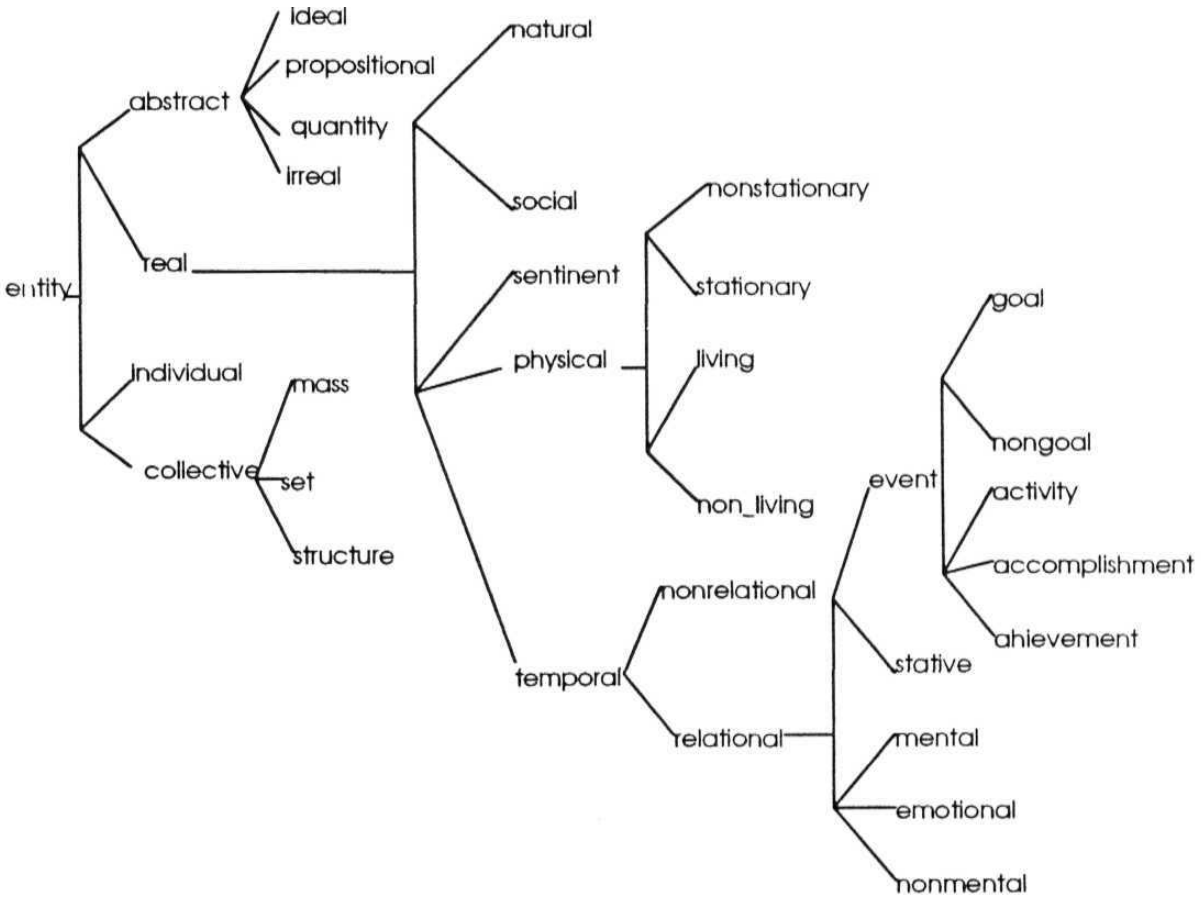

FIGURE 3.3: A Portion of the Sortal Hierarchy used in TELANGANA

For example, using the sortal hierarchy given in Figure 3.3 one can classify many entities as given below.

Rama	real & individual & physical & natural & living & nonstationary & **selfmoving**

Stone	**real & individual & physical & natural & non_living & nonstationary**
village	**real & social & collective**
secretary	**real & individual & sentient** & **social**
book	**abstract & collective & propositional**

Words are attached to the hierarchy as terminal nodes. New node names can be used to signify certain common sortal types/categories. For example a new sortal type , "role", may be defined as **real & individual** & **sentient** & social. TELANGANA uses this sortal hierarchy extensively to judge the semantic well-formedness of sentences (see Chapter 5).

Sortal hierarchy is encoded in TELANGANA as Prolog terms as proposed by Mellish [Mell88]. Mutually exclusive classification is encoded as different arguments of the same term, and entities on the same path of the hierarchy are encoded as embedded terms. For example the sort 'entity" in Figure 3.3 is encoded as entity (_,_). The sort **ideal** is encoded as the Prolog term entity (abstract (ideal,_,_,_),_). The sortal type of "secretary" is **real** & **individual & sentient** & **social.** This sortal information is encoded as the Prolog term entity (real (social, sentient),individual). This way of encoding cross-classificatory hierarchical information ensures that the unification of two sorts succeeds only in case the sorts do not violate an exclusive classification restrictions imposed by the hierarchy.

3.7 conclusion

TELANGANA program consists of four stages of sentence processing. These have been developed in an overall architecture that could readily include inter-sentential reference resolution and contextual disambiguation in future. The input sentence is transformed at each stage into a form closer in semantics to the database query language. The major contribution in this chapter is to show the possibility of representing *aakaamksha* and *yogyata* through structure sharing data structures. This is a major departure from the other approaches taken so far [Bhar90a, Sinh88] for parsing Indian languages. The details of representing *aakaamksha* and *yogyata* using the Meaning Representation Language were presented. The lexicon which contains the feature bundles associated with each word forms the backbone of the parser to be described in the next two chapters.

Chapter 4

Morphological Analyser

4.1 Motivation

Telugu language is highly inflection and suffixation oriented. About 67%[1] of the words in written text occur as compound words. Compounding often results in elision and also declination of some characters. Due to this and the high occurrence of compound words it is impossible to mechanically analyse, even semi-toy size, Telugu texts without a suitable morphological analyser (MA) and an associated lexicon or dictionary. The adoption of foreign language words from Hindi, Sanskrit, English and other languages also creates special problems in the analysis of Telugu words. In this chapter, the techniques adopted in this thesis to develop a MA for Telugu are detailed. To start with, the problems in developing a MA for Telugu are described. Subsequently, the inadequacies of the traditional English oriented MAs for the analysis of Telugu words are high lighted.

There are two hurdles in the development of a MA for Telugu. Firstly, at present machine readable dictionaries or lexicons are not available for Telugu. Without a suitable lexicon compound word analysis becomes impossible. Lack of machine readable lexicons also affects the lower end of NLP applications like spell checkers and sentence style checkers. Current word processors and desktop publishing software mostly deal with the orthography of Telugu only and can not process Telugu at any greater depth. Spell checking facilities are not available in any of them due to the lack of machine readable Telugu dictionaries.

Secondly, the morphological rules described in traditional Telugu grammar books [Brow81, Cinn51, Kris85] have been language generation oriented and hence extend little help in morphological analysis. Generation oriented rules which describe the ways and means to combine a root word and with its suffixes (postpositions, morphemes indicating tense, number, gender, and person) to generate a complete word. Given a well-formed word, these rules do not give all the possible ways of breaking the word into its constituents. Typically in Telugu, up to 6 suffixes may be added to a root to form a simple word. These simple words

[1] Based on data collected (around 2,82,000 words) from a news paper over a period of 6 months.

are combined, subject to certain constraints called ***saMdhi*** (compound word formation) rules, to form compound words.

Spoken language and the language presently used in Telugu news papers does not strictly adhere to traditional word generation rules **[Brow81,Cinn51].** This alienates these standard grammar books from giving a true picture of the language in use. Because of the "generation orientation" of the rules, and because of the language in vogue which does not follow the grammar rules very strictly, given a compound word, one cannot with 100% certainty predict the constituent roots and the suffixes. This happens more so, because there is a many-to-one relationship between root+suffix form of a word and its final word form; a single word can be analysed in different ways as roots and suffixes. For example, analysing the following simple words also leads to ambiguities as follows

అంది = అన్ + ఇంది
aMdi=an(to say)+*iMdi*(past tense, feminine gender)

అంది = అంద్ + ఇ
aMdi=aMd (to reach)+*i*(past)

పళ్ళు= పన్ను + లు
paLlu=pannu(tooth)+*lu*(plural)

పళ్ళు= పండు + లు
paLlu=paMDu(fruit)+*lu*(plural)

Lack of reliable "analysis oriented" morphological rules and a machine readable lexicon have motivated the author to study the possibility of simultaneously developing both the lexicon and the morphological rules. This lead to the development of a novel method, described later on, for building lexicons of root words from textual corpora.

4.2 Suffixes in TELANGANA

The word suffix is used in this thesis to refers to inflections, postpositions and markers indicating tense, number, person and gender, comparatives, negatives, imperatives, hortative and similar other phenomenon. The notion of suffixes used in TELANGANA is a minor

variant of the notion used by linguists. For example the words *tinnAnu, ammanu,* and *coosiMdi* are analysed in modern grammar books [Kris85] as

తిన్నాను= తిన్+ ఇన+ను
tinnAnu= tin+ ina+nu
అమ్మను = అమ్మ+అ+ను
ammanu= amm+a+nu
చూసింది= చూచ్+ఇన+ది
coosiMdi= cooc+ina+di

whereas in TELANGANA they are analysed as

తిన్నాను= తిన్+ఆను
tinnAnu= tin+Anu
అమ్మను = అమ్మ+అను
ammanu= amm+anu
చూసింది= చూడ్+ఇంది
coosiMdi= cooD+iMdi

This minor variation can be easily removed by noting that *Anu, anu* and *iMdi* can be rewritten as *ina+nu, a+nu, ina+di.* But such an approach was not taken in TELANGANA because the purpose of *ina,* to indicate past tense, can be easily subsumed by *Anu* and *iMdi* and the purpose of *a,* to indicate negation, can be easily subsumed by *anu* If this deviation in definition of suffixes was not taken, the affect of the suffix *ina* on the root needs to be calculated at the time of syntactic/semantic analysis. The suffix *ina,* which can either indicate past tense or a relative clause marker leads to ambiguity in syntactic analysis. Whereas in the present notion of suffixes, this ambiguity is resolved at the morphology stage itself. Another reason for the deviation is that words *aDiginAru, ceppinAru* which contain an embedded *ina* are not used in newspaper Telugu. Instead the present day news paper Telugu uses words like *aDigAru, ceppAru* which do not contain an embedded *ina.* Similar computational and usage considerations motivated the selection of all the suffixes in TELANGANA. Every attempt was made to be close in spirit to linguistically accepted suffixes. The deviations in TELANGANA only affect the performance of the system, and could have been easily hidden within the syntactic/semantic analyser's functionality. This was not done, in order to give a true picture of the morphological processing. For similar reasons a few deviations in the base form of verbs was also assumed.

Before understanding the details of the morphological analysis algorithm used in TELANGANA, it is necessary to understand why the traditional models used for morphological analysis of English are not suitable for analysing Telugu morphology.

4.3 Inadequacy of Older Models

Koskenniemi's "two-level model" (TLM) of morphology which modelled Finnish morphology as a finite state automata has been the subject of great attention. It has been used for a variety of languages like English [Dalr87, Kosk83], Japanese [Sasa83], and Romanian [Khan831. The two level approach defines a correspondence between the morphemic level[2] (ML) of the word plus its suffixes and the phonemic level[3] (PL) as it appears when written. Typical examples of ML and PL in English are

ML	PL
go+s	goes
die+ing	dying
index+s	indices

A single two-level rule defines the correspondences between the ML and PL of a complex word under certain local contextual conditions. These contextual conditions essentially describe the preceding and succeeding characters that should be present in the lexical form and the inflection, for the rule to apply. For example the following "I-to-Y rule" deals with English "ing" suffixation and can be used to analyse the word "dying" as "die+ing" .

i:y e:ε +:e i:i

In the above rule, the symbols occurring on the left hand side of the colon symbol, ":", are symbols that occur in the morphemic level of a word, and the symbols occurring on the right hand side of colon symbol are the symbols that occur at phonemic level. The symbol ε refers to phonological null.

The above rule is implemented by a finite-state machine with two scanning heads that move together along the morphemic and phonemic level strings. The machine starts out in state 1 and moves forward by changing state, based on its current state and the pair of characters it is scanning. If the finite-state machine ends in a final state after accepting all the characters of the input string, then the correspondence between the ML and PL of the word is accepted,

[2] Also known as lexical level
[3] Also known as surface level

otherwise an error is indicated. All the rules in the morphological analyser are compiled into finite state machines (FSMs), and when a word is analysed, all the FSMs are used in parallel.

The dictionary component of TLM contains all the root words and the allowed continuation suffixes for each root word. Each root word of the dictionary corresponds to some final state in a transition network. All the root words in the dictionary are compiled into this transition network. This finite state network and the FSMs are used in tandem to analyse a word.

Popular as it may be, the TLM does not help in modelling the Telugu morphology succinctly. The TLM based rules when applied to Telugu lead to problems of perspicuity. The number of rules tend to be too large. The rules thus lack cognitive economy. The fundamental and unsaid assumption in the two level model is that the correspondence between the morphemic and phonemic levels of representation of words is highly local. This is not true with Telugu. For example, consider the verb *tagul* (to hit or touch) when conjoined with the morpheme *am* (3rd person, plural) gives *tagalaru.*

t a g a l a r u (PL)

t a g u l+a r u (ML)

This means that in two-level rules [Dalr87] , the above phenomenon can be expressed as

u:a l:l +:ε a:a --4.1

Even though the above rule can correctly analyse many words like *pagaladu - pagul + adu, migalaru =migul + aru, ragalavu = ragul+avu,* it fails to analyse words like,

t a g a l a g a l a r u (PL)

For the above word, rule (1) would output

t a g a l a g+a r u (ML)

whereas the correct analysis should be

ta g a l a g a l a+a r u (ML)

(note: *tagalagala = tagul+gala*). This has happened because rule 4.1 should not be applied in the presence of a preceding *gal* in the word. In principle this problem of rule 1 can be alleviate by appending *g:g a:a l:l :a:a* to its left and making it.

g:g a:a l:l :a:a u:a l:l +:ε a:a **--4.2**

Many more examples of the above kind can be given to prove the non-local phenomenon in Telugu morphology. Even though these non local kind of problems can be circumvented by positing extra context into the two-level rules as above, the number of such extra-contexts tends to be so large that it raises serious questions about the validity of using two-level morphology model for Telugu. In English, both the number of suffixes and number of times they can be applied over a single root are very small. Whereas in Telugu, many more suffixes can be attached to morphemes to generate long non-compound words from simple morphemes. Due to all the above considerations, the author believes that TLM is not suitable for Telugu. In addition, there are some constraints on the successive application of the suffixes on the root morpheme. These constraints are typically like the Linear Precedence rules in GPSG [Gazd85a] but applicable at morphological levels. These can not be modelled in the TLM formalism.

From a practical point of view also there is a problem. The TLM assumes the availability of a dictionary of root words along with possible continuation suffixes and does not work with open-ended domains. While developing TELANGANA the author did not have such a lexicon. This made it difficult to use TLM. The standard TLM algorithms need modifications to suit dictionary-less operation.

From a psychological point of view also, TLM has a problem. Native Telugu speakers when confronted with a new word can guess reasonably well the likely root of the word, whereas TLM cannot do so as the root of the word would not be found in the lexicon.

In view of all the above, the MA was developed from scratch and was designed so as to work both with and without a lexicon. The author started without a lexicon and built one automatically using the morphological rules developed during the course of this study. In the rest of the Chapter the rule formalism and the methods devised by the author to build the lexicon from the declarative rules are described. These rules can be compiled into multiple finite state automata or interpreted by a program depending on ones requirements.

4.4 Rule Formalism

A good rule formalism for morphological analysis of Telugu should have at least three features;

(1) the ability to group suffixes and their applicability preconditions,

(2) a facility to express multiple analysis, and

(3) the ability to express the valid sequences of suffixes.

MA tries to achieve all these.

Some common suffixes and the constraints on their order of appearance is depicted graphically as a transition network in Figure 4.1. This figure portrays a small portion of the transition network for recognizing verbal suffixes. The network for the full set of suffixes analysed by MA would be extremely complex and difficult to depict.

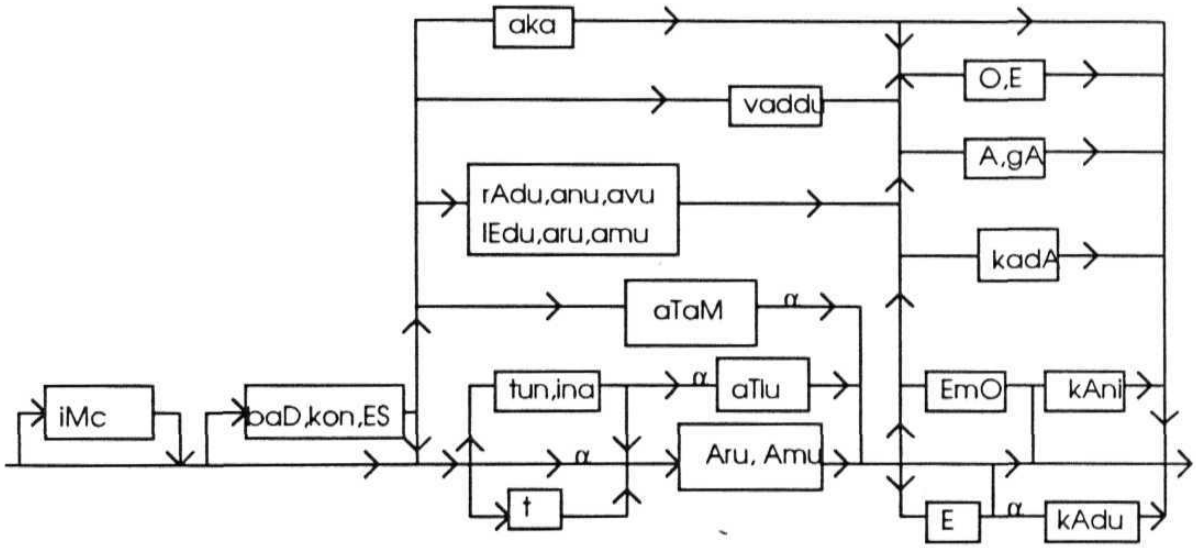

FIGURE 4.1 Precedence relations amongst some verbal suffixes

Given a root word, one can add to it only those suffixes which are encountered while travelling the network left to right following the arrow heads. If an arrow is present on some path, one is not allowed to travel on its opposite direction. The symbol "a" is used to indicate that to travel via the box containing the suffix "kAdu" the paths marked by "a" may only be taken. To generate new words from root words one should travel the network left to right. To analyse a given word, one should travel the network right to left after reversing the arrows. To see the capability of this network, one can take the example of the verb *koTT* (beat). A few of the large number of words which can be generated using this network are,

koTTADU	beat (past tense)	*koTT+ADU*
kOTTiMcADu	caused to beat	***koTT+iMc+ADu***

koTTiMcabaDDADu	was caused to be beaten	*koTT+iMC+baD+ADu*
koTTistAru	will cause to beat	*koTT+ iMc+t+Aru*
koTTistunnAru	are causing to beat	*koTT+iMc+tun+Aru*
koTTiMpabaDutunnAru	are being caused to be beaten	*koTT+iMc+baD+ tun+Aru*
koTTiMCabaDutunnArA	are being caused cused to be beaten ?	*koTT+IMc+baD+tun+ aru+A*
koTTinaTlE	equivalent to beating	*koTT+ina+aTlu+E*
koTTiMcinaTlukAdu	not equivalent to causing to beat	*koTT+iMc+ina+aTlu+ kAdu*
koTTiMcabaDinaTluEkAdA	is it not equivalent to causing to be beaten?	*kott+iMc+baD+ina+ aTlu+kAdu+A*
koTTabaDakanE	even without being beaten	*koTT+baD+aka+E*
koTTArE	oh! (they) have beaten	*koTT+aru+E*
koTTarA	do (they) not beat?	*koTT+aru+A*

MA can currently deal with about 200 suffixes. The precedence relations amongst them form a very complicated transition network. For the sake of simplicity such a complex network was simplified without losing much discriminating power by grouping the suffixes into six levels. The level numbers reflect the linear precedence relations amongst the suffixes. The grouping into six levels is somewhat crude in the sense that it can analyse all correct word formations but cannot reject some specific and highly improbable illegal combinations of suffixes. If the purpose of morphological analysis was spell-checking then one can easily modify MA to incorporate all the additional levels in rules. For database access purposes such an exhaustive approach need not be taken.

The assignment of the these levels to suffixes was done by first implementing a version of the MA without using these levels. Using this version of the MA, the author analysed a large textual corpus to identify all the legal combinations of suffixes. The textual corpus consisted of about 2600 news paper articles collected by the author from a popular Telugu news paper for this purpose. Using MA over the words in the textual corpus a large list of all suffix combinations was generated. From this list, the actual precedence relations amongst the suffixes was determined. This trick saved the author the trouble of hand tuning the levels. As some words were wrongly analysed due to lack of level information in the morphological rules, the precedence relations got as above were prone to errors. All such errors were removed manually. These levels are very useful to reject false analyses, especially when the

lexicon grows large and unexpected analyses crop up due to the abundance of short root words in Telugu.

A rule consists of a Left Hand Side (LHS), a middle part (MP) and a Right Hand Side (RHS). The syntax of the rule is ,

{xl>x2,Lvll,Lvl2;} -cat1/cat2-

{y1>[y2(cat3),Lvl3];}C.

where

- x1, x2, y1, y2 are arbitrary strings of alphabets, like *Aru, iMdi* and *tO.* They may also contain special symbols like @, + , - which are described later on.

- Lvll,Lvl2,Lvl3 are numbers ranging from 0 to 5. Lvl1 is input level. After an application of the rule, either Lvl2 (by default) or Lvl3 would be the output level.

- cat1, cat2 and cat3 are syntactic category symbols like n, v, postp, and adj. cat1 is the input category. Either cat2 (by default) or cat3 would be the output category. cat2 is optional. If absent, is taken to be same as cat1.

- C indicates the type of continuation after a rule is used. It could be "?" meaning, search lexicon immediately, ">" meaning proceed to next rule, "-" meaning fail the rule and do not invoke any other rule, and finally "." meaning accept suffix in the rule without looking into lexicon,

- "[" and "]" symbols indicates alternate morphemes, corresponding to the same phonemic substring,

- {} are meta symbols to indicate more than one occurrence of the LHS and RHS patterns, and are used only to explain the rule syntax.

The symbols x1, x2, yl and y2 are strings of characters. The string xl is typically a suffix like *Aru, iMdi, iMc, ki, ni* or *valana.* It could also be a partial suffix like *ru,* and *mu,* which are really parts of larger suffixes, *aru* and *amu* respectively. Alternately it could be an extended suffix like "uni" instead of the accusative suffix "ni". The string x2 is a replacement string and is emitted as output suffix in lieu of x1. For example when the LHS of a rule contains "uni>ni" , it means that the real suffix is *ni* but due to the word formation it appears

to be *uni.* Hence while outputting the analysis of the word *tanayuni* and *bAluni*, the suffix *ni* is output instead of "uni" as *tanayuDu+ni* and *bAluDu+ni* respectively.

The string "y1" is called the context of the suffix x1. What it means is, if the word to be analyzed contains yl as its right most characters after removing the string xl from it then x1 is really a suffix. The string y2 gives the replacement string for yl, just as x2 is a replacement string for x1. The strings y2 and x2 together give an idea of the elided/modified characters due to word formation. The following rule, is an example of this elision. The rule

mu>amu,5,3 -v- kO>kon,3;.

can be used to analyse words like *rAsukOmu* and *teesukOmu,* wherein two characters *(n* and *a)* are elided. By just using the above rule, the two words can be analyzed by MA as *rAsukon+amu* and *teesukon+amu* respectively.

The special symbol "@" when is used in x1, x2, yl and y2 is treated as a place holder for a wild character. Two occurrences of @ should represent the same wild character. This symbol can be used for checking for the presence of non-null strings, or double occurrence of some consonant. For example , the following rule can be used to analyse *Aru* ending words. In the analysis of the words *navvAru* (laughed) and *navAru* (a piece of cloth), the following kind of rule helps in blocking the *Aru* string in *navAru* from being analysed as a suffix.

Aru,5,4 -v- a@@>a@@,0;-.

The other special symbol used is "+", which helps in outputting more than one suffix in one step. A "-" symbol is used to indicate to MA not to output xl or x2 as output suffix and use y2 as suffix instead. The symbol "#" is used to indicate the beginning of a root word. When analysing the word like *cEsiddAM, rAsiddAM, ASiddAM, aDigiddAm* the following rule can give its analysis in one step.

iddAM>#ivv+Amu,5,5 -v- AS>ASiMc+t+Amu-,s>si,5;@>@i,5;.

The above rule analyses *cEsiddAM* as *cEsi+#ivv+Amu.* Further rules would analyse *cEsi* to *cEy+i.* Similarly, the words *rAsiddAM, aDigidaaM, ASiddAM* will be analysed into *rAy+#ivv+Amu, aDigi+#ivv+Amu* and *ASiMc+t+Amu.* Subsequent analysis would reduce *aDigi* to *aDug+i.*

4.4.1 Developing the Morphological Rules

Developing the morphological rules for Telugu is a significant task, as there are over 200 sufixes in Telugu. The manner in which each one of these rules behaves is different. Many rules tend to have a common behaviour over a range of combinations. To identify such combinations and delimit the applicabilty of the rules to such combinations only is the main task in developing the rules. The morphological rules used in this thesis were developed with aid of the of the textual corpus. For example the string "Aru" occurred 9502 times in the textual corpus in the suffix position of words. The morphological rules of TELANGANA were generated using these word lists for each suffix. The most frequent occurrences of the suffix, Aru, are given along with their frequency in the Figure 4.2.

Figure 4.2 brings out many interesting cases of *Aru* ending words which do not really contain *Aru* as suffix. For example the words, occurring in decreasing order of frequency, *vAru* (they), *sumAru* (approximately), *Aru* (six), *baMgAru* (made of gold), *tayaru* (ready), *mOTAru* (motor), *kharAru* (definite) have an *Aru* ending, but none of them arc verbs. These words are exceptions and must be handled in the rule set carefully. Without the help of the corpus and these word lists, it would have been very difficult to find exceptions to proposed rules. Standard text books do not give such exceptions while discussing word generation using the Aru suffix, because these are not words formed by conjoing Aru to any verb. Just as exceptions are easily brought out, so are regularities also brought out using the word lists. The current rules in TELANGANA are highly tuned to the textual corpus and perform well on this corpus. The corpus being at present relatively small, only four million bytes in size and also being gathered from a single source may not be adequate for a thorough analysis of unconstrained Telugu. However, the rules developed so far in TELANGANA far out scope the requirements for database access.

1662 telipAru
1225 annAru
887 ceppAru
732 cESAru
673 vAru
534 kOrAru
328 pErkonnAru
308 vivariMcAru
202 ArOpiMcAru
169 vimarSiMcAru
168 sumAru
162 pAlgonnAru
156 cEstunnAru
118 kOrutunnAru
102 soociMcAru
101 prasaMgiMcAru
100 icc Aru
79 nirvahiMcAru
79 Aru
78 vyaktaMcESAru
69 hAjarayyAru
67 unnAru
61 vApOyAru
50 vahiMcAru
49 jaripAru
49 aMTunnAru
44 praSniMcAru
43 mATlADA ru
42 vApOtunnAru
40 teliyajESAru
39 aMdajESAru
38 baMgAru
38 abhiprAyapaDDAru
37 vellaDiMcAru
35 ennikayyAru
34 bhAvistunnAru
33 AdESiMcAru
32 uMdannAru
30 samarpiMcAru
28 vyAkhyAniMcAru
28 vaccAru
28 khaMDiMcAru
28 guravutunnAru
27 cheppAru
26 paDutunnAru
25 lEdannAru
24 nirvahistunnAru
23 edurkoMTunnAru
21 vESAru
21 poMdAru
21 ErpATucESAru
20 vimarSistunnAru
20 ArOpistunnAru
19 viJapticESAru
18 jarupukunnAru
18 cEstAru
18 abhinaMdiMcAru
16 veLlAru
14 velibuccAru
14 tayAru
14 niyamitulayyAru
14 mOTAru
14 jaripiMcAru
13 teesukunnAru
13 taraliMcAru
13 kharAru
13 cEyiMcAru
13 ceMdinavAru
13 ayyAru
13aMdiMcAru
12 vyaktaparicAru
12 uMcAru
12 saMmAniMcAru
12 chESAru
11 teliyacESAru
11 SivAru
10 uMTAru
10 peTTAru
10 pAripOyAru
10 nirvahistAru
10 koniyADAru
9 udbhOdiMcAru
9 tAru
9 spashTaMcESAru
9 niyamiMcAru
9 istAru
9 cEsAru
9 cErukunnAru
9 baMDAru
9 aresTucESAru
8 vyaktaMcEstunnAru
8 vidhiMcAru
8 viccESAru
8 peMcAru
8 kETAyiMcAru
8 jaruputunnAru
8 digAru
8 cinnakAru
8 cESAmannAru
8 ceMdAru
7 samAdhAnamiccAru
7 sAgistunnAru
7 prAraMbhistAru
7 pATiMcAru
7 panicESAru
7 oorEgiMcAru
6 rAjAdaraNapoMdAru

Figure 4.2 List of *Aru* ending words from Text corpus (edited list)

Standard text books like [Kris85] were also used to the extent possible to decrease the amount of work needed to formulate the rules for the rare suffixes. Many suffixes, like *Amu, amu* were not found in the corpus. Rules for such suffixes were extrapolated from books and from already formulated rules corresponding to similar suffixes.

Some excerpts from typical morphological rules used in TELANGANA are given below.

ru>aru,5,3;du>adu,5,3; -v- kO>kon,3;uMDa>un,3;

iMca>iMc,1;a>NULL,0;>. --3.3

Aru,5,4; Ayi,5,3; -v- As>Ay,0;unn>un,4;nn>n,4;. -3.4

kon,3,2;kun,3,2 -v- lusu>liy,0;su>y,0;u>NULL,0;. --3.5

Using the above rules on can analyse words which are derived from the root word *rAy* (write).

rAsukonnAru = *rAy+kon+Aru*(write past, lural, beneficiary)

rAsukonaru = *rAy+kon+aru* (not write past, plural, beneficiary)

rAsukOru = *rAy+kon+aru* (not write past, plural)

rAyaru = *rAy+aru* (not write past, plural)

rAsAru = *rAy+Aru*(write past, plural)

To illustrate the use of the rules, the first word *rAsukonnAru* be taken. The word *rAsukonnAru* is first analysed by using rule 3. None of the context strings of rule 3 match the word remaining after stripping out *ru* from *rAsukonnAru.* The continuation of the rule being ">", the next rule is tried. One of the context strings in this rule matches the word. So, the suffix *Aru* is removed reducing the input word to *rAsukon.* This word is further reduced by rule 5 to *rAy.* At this stage the word is not further reducible using the above three rules. The above rules are meant for analysing verbs only. Similar rules exist for analysing nouns also.

The continuation part of the RHS of a rule indicates the action to be taken, depending on whether the rule succeeds or fails. By placing more specific rules first and less specific later, one can gain good control over the order the rules can be applied. Currently there are 91 morphological rules in MA. These rules capture the important generalizations in Telugu morphology, due to the grouping nature of the rules. If these rules were to be re-written as two-level morphological rules, the equivalent two-level rules may number close to 600 rules.

Even though TLM is not well suited for describing Telugu morphology, the computational model it assumes, that of cascaded finite-state-automata, is useful for efficient analysis of words. The morphological rules of MA can be readily compiled [Bart87] into such finite-state-machinea

(FSMs). However, the existence of any such FSM is not a pre-requisite for the use of the morphological rules in MA.

4.5 The Morphological Analysis Algorithm

The morphological analyser starts at level 5 when begins to examine a word. It takes the syntactic tag of the word to he undefined. Using the above rules, a word is first analysed right-to-left. Once a proper suffix is found, it is removed with due modifications to word boundaries (as captured in the RHS part of the rule). The tag information in the rule is unified with the tag information passed on to the rule at the time of invoking the rule. A mismatch in tag information leads to failure of the rule. If tags match, the new level is taken from the rule. With this new level number and new tag, the analysis continues as before. When level 0 is reached, the remaining character string, supposed to be a root word, is looked up in the lexicon, if lexicon mode of analysis is activated. Otherwise the word is returned as the guessed root word. The finer details of this suffix analysis are given below in a procedural form. The symbols, x1, x2, y1, y3, Lvl1, Lvl2, and Lvl3 used in describing the suffix analysis procedures below, Check_Suffix and Check_Context, mean the same as the ones described in the rule template earlier.

Procedure Check_Suffix(WORD, LVL,TAG);

STEP1: If LVL is equal to 0, then return NO; else Create a choice point for backtracking. Set the variable DONE=NO-SUCCESS.

STEP2: Checkup if any suffix at level LVL is present in the input word WORD by scanning (or by state transition in some FSM) WORD right-to-left. If a suffix is present as desired by some morphological rule, MR, and if TAG matches cat1 of that rule, then reassign WORD = WORD - suffix.

STEP3: Checkup by calling Check_Context(WORD,LVL,TAG) if WORD contains the context string, if any, required by MR. Take further action as per STEP 4 or STEP5.

STEP4: If a context string is present as required by MR, and the length of WORD is more than 3, then replace the context string yl by y2 in WORD and add x2 of the MR to WORD. Assign LVL = lvl3 of MR and TAG= cat3 of MR, and DONE=SUCCESS. Goto STEP2.

STEP5: If no context string is present as per MR rule, that is the return code of the procedure Check_Context is NO-SUCCESS, then

(1) if the continuation is "?" then if LVL is not 0, look up the dictionary by calling Lookup_Dict(WORD,TAG) else call Check_suffix(WORD,LVL,TAG). Return the code Lookup_Dict or Check_Suffix procedures to the caller of this procedure.

(2) if the continuation is ">" undo all assignments up to last backtrack point and go to STEP2.

(3) if the continuation is ".", then return to the caller of Check_Suffix the value of DONE as return-value of the proc edure Check_Suffix and LVL=lvl2, TAG=cat2.

Procedure Check_Context(WORD,MR,LVL,TAG).

STEP1: Create a choice point for backtracking.

STEP2: Checkup if a suitable context string, y 1, as required MR is present in the input word WORD by scanning (or by state transition in some FSM) WORD right-to-left. While checking up for y1, do special processing for "@" characters. If a context string is not present return to the caller NO-SUCCESS.

STEP3: If a context string, y1, is present, then replace the y1 substring of the word WORD by the string y2. That is, reassign WORD = WORD - context string y1+ y2.

STEP4: If a cat3 string is present in the MR rule, then

(1) if cat3 starts with "?" and is some string like "?xyz", then if LVL is 0, look up the dictionary by calling Lookup_Dict(WORD,xyz) else call Check_Suffix(WORD, LVL, TAG) with LVL=lvl3 and TAG=xyz. Return the code returned by the above called Lookup_Dict or Check_Suffix procedures to the caller of this procedure.

(2) if cat3 does not start with "?" or if it is not present then assign LVL=lvl3, and TAG= cat3 if cat3 is present else assign TAG=cat2. Return SUCCESS.

The procedure Lookup_Dict implements two important functions of the analysis. One is it simply looks up a lexicon to see if the given word is present in it. If not present directly, Lookup_Dict tries to break the input word according to *saMdhi* (compound word formation) rules and see if the right hand side constituent is a root word. For example the word *rAjAdaraNapoMdAru,* taken from Figure 4.2, when analyzed by the **Check_Suffix** procedure yields ***rAjAdaraNapoMd+Aru.*** The word ***rAjAdaraNapoMd*** is a compound word and needs further analysis.

Compound words cannot be analysed without the aid of the lexicon. In the case of compound words, the initial suffixes are removed as above. The remaining word and all its right-most substrings (length of the string - 1 substrings) are looked up in the lexicon moving the left boundary of the remaining word left-to-right. This ensures that words which are longer get preference over shorter words when word splitting is done. Once a word is found in the lexicon, the compound word is broken up at this point following compound forming rules of Telugu. The rules used in TELANGANA, at present, are capable of analysing the following kinds of word formations.

గత్యంతరం= గతి + అంతరం
gatyaMtaraM= gati+aMtaraM where *i+a* has become *ya.*

కథాసరిత=కథా + సరిత
kadhAsarita = kadha + sarita, where *a+Vowel* has become *AVowel.*

రాజాదరణ= రాజ + ఆదరణ
rAjAdaraNa= rAja + AdaraNa, where *a+A* has become *A.*

వాడదృష్టం= వాడి + అదృష్టం
vADadrushTaM= vADi + adrushTaM where *i+a* has become *a.*

ప్రజలన్నారు= ప్రజలు + అన్నారు
prajalannAru= prajalu + annAru where *u+a* has become *a.*

అష్టావధానం= అష్ట + అవధానం
asTAvadhAnaM= ashTa + avadhAnaM where *a+a* has become *A.*

ముఖ్యోపన్యాసం= ముఖ్య + ఉపన్యాసం
mukhOpanyAsaM= mukhya + upanyAsaM where *a+u* has become *O.*

These rules again can be represented as regular morphological rules. However this has not been done. These rules are treated as **meta-rules** and are invoked only for compound word analysis. If these rules are used for suffix removal, they can lead to gross over generation of invalid patterns. Hence these rules are used as meta-rules, which can be applied only when compound word breaking is being attempted. These rules are few in number but highly context-

free. They are tried one after the other in a backtracking fashion. Compound word analysis, thus, requires enormous amount of backtracking, as no compound formation rule is deterministic. The following procedure further explains the compound word analysis and lexicon look up procedure Lookup_Dict.

Procedure Lookup_Dict(WORD,TAG).

STEP1: If length of the word is less than three characters, do a quick look in a memory resident table containing words which are only one or two characters long. If the lexicon mode is not set, goto STEP8.

STEP2: If length of WORD is less than 5 characters, then look up in lexicon directly. If WORD is present in lexicon, return to caller SUCCESS, else goto to STEP3.

STEP3: Set a pointer, P, to the beginning of the word - 1.

STEP4: Increment P by one. If P points to beyond WORD'S last character, return to caller NO-SUCCESS. At any time, P conceptually breaks WORD into a left part (LP) and a right part (RP).

STEP5: Check if RP, is present in the lexicon. If so, really break WORD at this point, into a left part(LP) and right part.

STEP6: Then look for further analysis of LP by calling Check_Suffix(LP,5,UNKNOWN) procedure. If Check_Suffix returns SUCCESS, then return SUCCESS to the caller of Lookup_Dict, else goto STEP7.

STEP7: Change the first character of RP and the last character of LP suitably in accordance with *saMdhi* (compound word formation rule of Telugu) rules. Try all *saMdhi* possibilities. If all possibilities are exhausted goto STEP 4. If some *saMdhi* rule is applicable, break WORD according to that *saMdhi* rules into LP and RP.

Goto STEP6.

STEP8: If the lexicon mode is not set, return SUCCESS to the caller.

To achieve compound word analysis, one requires a lexicon/dictionary. However when this work was started, no such dictionary was available. A "boot strapping" like approach was used to generate the lexicon, as explained in the next section.

4.6 Generating the Lexicon

The author ran morphological rules over the textual corpus of 282,000 words with dictionary look up option switched off. The rule application system extracted from each word the constituent root word to the extent possible. Lack of the lexicon, meant compound words got their right most words analysed and the other embedded simple words in the compound word were left intact. About 9,000 words out of the 282,000 remained un-analysed any further at this stage as they did not have any suffixes. The semi-root words extracted were about 34,500 in number. These semi-root words were used as a lexicon to analyse the same corpus of words, this time setting the lexicon enable switch on. This ensures that if a compound word contained some simple words that were present in the textual corpus as simple words only, then the compound word got totally analysed. At this stage the compound analysis portion of the rule execution system was modified a little bit so that while it scanned a word left-to-right to find the sub-strings of the word in the dictionary, it skips the first character from the word. This ensured that the entire word did not show up as the root word again!. Words were thus analysed as far as possible before being considered not found in the dictionary. After going through this step, about 4,500 words were still left unanalysed. These words either contained words that never occurred any where else in the corpus, or had some spelling mistakes. In addition, the rules, when used in dictionary-less mode, tend to be more conservative, and hence also some words would have been left unanalysed. From the rest of the words that were analysed the roots were extracted again. Surprisingly this strategy worked out very well, and the root words guessed from the corpus now reduced to about 24,100 words. This meant that the process was converging towards the root words. Further using the same strategy once again, the root words decreased to about 19,200. Further reduction was not possible thereafter. These were the best possible guesses that could be made without using a standard lexicon.

On a careful examination of the words, it was found that many words were minor variants of the others. This happened because the newly generated lexicon was unreliable to start with and the rules conservative. Hence some filter programs were developed, to judge the similarity between words, based on their endings, tag information, and the suffixes immediately following them in the text. Once two words were judged to be sufficiently similar, they were queried for final confirmation. Development of the filter was based on a trial and error process, but did not

warrant significant effort. After using the filter, the number of words reduced to about 11,000. Using these root words the entire corpus was analysed again along with the words that were left unanalysed previously. Not many of them could be analysed as most of them contained spelling mistakes or were a single occurrence a word. These single occurrences were also included into the lexicon, making it a little more than 11,600 words in total.

The lexicon contains the root words, the suffixes they go with, and the most probable syntactic category the word belongs to. The category information is mostly correct but often tends to be more general than necessary. For example, some element is tagged as a noun when it could be an adjective, as it is very difficult to differentiate an adjective from a noun through our rules. Adjectives and adverbs do not take suffixes but participate heavily in compound words. Hence adjectives are mistaken as nouns. A test to see whether a word always occurs with no suffixes in the corpus helped disambiguate many adjectives and adverbs. However, for some words the frequency of occurrence was so low that, it was not wise to use these heuristics. Larger corpuses would probably circumvent this problem. Finally, some amount of manual intervention was required to fine tune the category information. It may be noted that the numbers given above are indicative of the convergence rate and obviously would change, once one starts redoing the exercise with a larger textual corpus. However the order of magnitude of effort involved in generating the lexicon would remain the same.

The lexicon developed in this fashion gives the part of speech information at most. But that is not adequate for further syntactic/semantic analysis. For the further analysis one needs *aakaamksha* (expectancy) and *yogyata* (ability) information as described in Chapter 3. This kind of information was gathered for a few verbs from the textual corpus by constructing concordances for selected words. Software was developed to generate such concordance information. Figure 4.4 gives the concordance for the various variants of the root word *kOr* (desire/want) from a small portion of the corpus.

LINE NUMBER *KEY-WORD*

kOraaru (29) <-- *number of occurances*

10	/aa adhikaarini	**kOraaru./maddoor** manDalaMlO
25	DAkTar venkaTayya	kOraaru./daaniki **yam-el-e-**
44	cEyaalani adhyaxuDu	kOraaruikareeMnagar, phibravari
151	aMdaroo kRUshi cEyaalani	**kOraaru./urdoo bhaashagoppatanaanni**
321	**amalu** cEyaalsiMdani	**kOraaru./graamaallO bhoomiSistu**

kOrutu (1)

516	SaaMti vardillaalani	kOrutu sOmavaaraMnaaDu

kOrutunnaaru (9)

64	tolagiMcaalani vaaru	kOrutunnaaru./saMgaareDDi,
413	teesukOvaalani prajalu	kOrutunnaaru./nijaayiteegala
739	maarcaalani /sthaanikulu	kOrutunnaaruVprastutaM
991	paluvuru vidyaarthulu	kOrutunnaaru./iMDOr kreeDalaku
1231	cEyaalani paluvuru	kOrutunnaaru./kalvakurti,
1264	aMdiMcaalani, paluvuru	kOrutunnaaru./ee paMjaabu
1687	jillaa kalekTarnu	kOrutunnaaru./idilaauMDagaa
1719	badilcc cEyaalani	kOrutunnaaru./jaipoor maMDalaMlOni
1831	sarpaMculu	kOrutunnaaruAhaanaapuraM,

FIGURE 4.3 Concordance for the word *kOr*.

From the concordance information *aakaamksha* (expectancy) and *yogyata* (ability) information can be manually constructed. This being a very laborious process, it was attempted for a very few verbs which were thought to be important for database access. In fact for these few verbs, *aakaamksha* (expectancy) and *yogyata* (ability) can easily generated even without using concordance data. But it was done to provide a platform for future enhancements to TELANGANA. The concordance software also lends some generality to the information provided in the Feature Bundles (Chapter 3) of the verbs currently understood by TELANGANA. In any case to generate FBs of all the verbs in the lexicon, some automatic methods need to be developed. Currently such methods are not available in literature.

4.7 Conclusion

In this Chapter a method for analysing compound words in Telugu texts was described. The method relies on novel declarative style morphological rules to analyse the words. These rules capture the linear precedence conditions of the suffixes and are hence more powerful than simple morphological rules employed for analysis of English. These rules can work with or without the help of a lexicon. However, the accuracy of analysis improves with the presence of a lexicon. Initially the declarative rules of morphology in TELANGANA were used to analyse a large Telugu textual corpus without the help of a lexicon. The words analysed in this way, were used

for further morphological analysis as an initial approximation to the root words. Using this kind of an iterative methodology a lexicon for Telugu was developed. This lexicon, at present, contains only the syntactic category information. To the best knowledge of the author, this is the first Morphological Analyser for Telugu with such an extensive coverage of the language. The lexicon developed by the author is probably the first machine readable lexicon for Telugu, generated by automatic methods.

Chapter 5

Syntactic and Semantic Analysis

5.1 Introduction

To understand natural language queries two hurdles have to be overcome. The first one is the morphological hurdle and the second one is the syntactic and semantic hurdle. In chapter 4, the methods used in this thesis to overcome morphological hurdles were described. This chapter details how the syntactic and semantic hurdles are overcome. The morphological analyser (MA) of TELANGANA breaks the words of the input sentence into root words and suffixes, and passes them, in the form a list, to the next stage of processing. The syntactic/semantic analyser (SSA) component of TELANGANA takes this list as input, parses it and generates a feature bundle(FB) representing the meaning of the input sentence as output. *Samarthah* theory was used as the basis for developing the SSA component of TELANGANA. SSA forms the heart of TELANGANA.

In chapter one, *samarthah* theory was presented as described in the traditional philosophical schools of India. The way *samarthah* is described in those schools leads to some problems in interpretation [Maha84] and also leaves some assumptions implicit. The first purpose of this chapter is to present the notion of *samarthah* more formally and explicitly. This formalization of the notion of *samarthah* suggests a novel way for parsing Telugu sentences, called functional application parsing (FAP) in this thesis. The link between *samarthah* theory and FAP is explored subsequently. After that, the details of parsing simple Telugu sentences using FAP are presented in the rest of the chapter. As a prelude to the description of SSA, the notion of *samarthah* and how it leads to the parsing technique FAP, is explained in the next section.

5.2 Making computational sense of *samarthah*

In chapter three, the notions of *aakaamksha* (expectancy/desire) and *yogyata* (ability) were formalised. From that formalization, it was seen that they could be represented as FBs with structure sharing capabilities. A FB, encapsulating the *aakaamksha* (expectancy), *yogyata* (ability) and some other related information, is associated with every word of the language. Certain feature-value pairs of the FB represent the

aakaamksha (expectancy) aspect of the word and certain other feature-value pairs of the FB represent the *yogyata* (ability) aspect of the word. Once *aakaamksha* and *yogyata* are formalised, then *samarthah* can also be formalised. Samarthah is said to be present between two words X and W, under the following conditions

(i) If the *aakaamksha* of one of the two words is satisfied by the *yogyata* of the other word, X and W occur in the right word order and if the interaction between X and W, computed by the functional application (to be explained soon) of one over the other, leads to some simple word, compound word or a word conglomerate Z, that is in *yogyata* (ability) or *aakaamksha* (expectancy) relation to the rest of the words in the sentence, then X and W are said to be 'equi-meaning' *(samarthah).*

(ii) If the two words X and W do not have the necessary mutual *aakaamksha-yogyata* relationship as above, but have *yogyata* with respect to some other *aakaamksha* word Y in the sentence, then also *samarthah* is said to be present between X and W.

If neither of the above defining conditions hold, then the two words will not be *samarthah* and hence can neither participate in the formation of a compound-word nor be part of a single sentence.

The first of the *samarthah* conditions takes care of PP and compound word formations. For example, the words *rAjAnah* and *purushah* in Sanskrit can normally give rise to the compound word *rAjAnahpurushah* as given below.

రామస్య భార్యా రాజానః పురుషం అపశ్యత్
rAmasya bhAryA rAjAnahpurshaM apasyat
Rama's wife saw the kings man

On the other hand, the same two words, *rAjAnah* and *purushah,* appearing in the same word order, cannot participate in the formation of a compound word in the following much quoted Sanskrit example sentence [Maha84, pp 16]

భార్యా రాజానః పురుషో దేవదత్తస్య
bhAryA rAjAnah purushO devadattasya
The wife of the king and the man of Devadatta

In the above sentence, due to lack of *aakaamksha/yogyata* relationship between the compound word *rAjAnahpurushO* with any other word in the sentence, the words

***rAjAnah* and *purushah* are not *samarthah*. Hence the compound word, *rAjAnahpurushah*,** should not be used in the above sentence. Whenever any two words (or morphemes) have *aakaamksha/yogyata* relationship, their *sannidhi* (proximity) can potentially lead to a new single word or a compound word. The derived new word and its FB, should be either a *yogyata* or an **aakaamksha** type of a word with respect to the rest of the words in the sentence, otherwise the words in question will not be *samarthah* and hence cannot form a combined word. Thus *samarthah* dictates the formation of words and sentences.

The expression "word conglomerate" was used in the definition of *samarthah*, above, to signify that even if words are not compoundable, *samarthah* can exist between them owing to their *yogyata/aakaamksha* relationship with respect to the other words in the sentence. Words in a noun phrase conjunction are typical examples of word conglomerates. For example, in the following Sanskrit sentence

మామకాః పాండవాశ్చైవ కిమకుర్వత్ సంజయ
mAmakAh pAMDavAscaiva kimakuravat saMjaya
What did my sons and Pandavas do Sanjaya?

the words *mAmakAh* and *pAMDavAscaiva* form a word conglomerate. These two words are *samarthah* as they form a co-ordinate postpostion phrase that is in *yogyata* relation to the verb *kri* (do).

The second constraint in the definition of *samarthah* entails that complement and adjunct words have *samarthah*. This takes care of the *samarthah* amongst the postpostion phrases in the following Sanskrit sentence [Maha84, pp33-34].

రామహ కూపాత్ హస్తేన శ్యామాయ పాత్రే జలం ఆనయతి
rAmah koopAt hastEna shyAmAya pAtre jalaM Anayati
Rama fetches for Shyama water in a pot by hand from a well

Thus definition of *samarthah* given above through the conditions (i) and (ii) captures the full purport of *samarthah* propounded in Panini's Ashtaadhyaayi (AST) [Josh68, Maha84].

In the Telugu sentence given below

రాముడు సీతని ఒక చిన్న పార్కులో చూశాడు
rAmuDu seetani oka cinna pArkulO cooSADu

Rama saw sita in a small park

by taking recourse to the constraint (i) in the definition of *samarthah,* one can observe that *samarthah* exists between

rAmuDu and *cooDaTaM,*
seeta and *cooDaTaM,*
(oka cinna pArk) and *cooDaTaM,*

due to the match between *aakaamksha* of *cooDaTaM* and the *yogyata* of *rAmuDu, sita, pArk* . Also there is *samarthah* between

oka and *cinna,*
oka and *pArk*

owing to the match between the *aakaamksha* of *pArk* and the *yogyata* of *oka,* and *cinna.* In addition to the above sets of *samarthah,* by the second part of the definition, *samarthah* exists between

raamuDu and *seeta*
seeta and *pArk*

On the other hand, even though the words *seetani* and *oka,* and *oka* and *cinna* occur adjacent to one another there is no *samarthah* between them.

Just as there is *samarthah* relationship between different words of a sentence, there is also *samarthah* at a micro-level between the various root words like *seeta, pArk, cooD* and their suffixes *ni, lO* and *ADu.* The *samarthah* between root words and their suffixes constrains the kind of suffixes that can go with root words.

At this stage some observations need to be made. If instead of the word *cooSADu,* the word *cooSAru,* were used in the above sentence, there would have been no *samarthah* between *rAmuDu* and *cooDaTaM,* and hence the sentence would have been ungrammatical. This observation entails that *samarthah* encompasses agreement constraints also. The second crucial observation is that, the difference between the given words, *cooSAru* and *cooSADu,* is only due to the difference in their suffixes. This means that suffixes do contribute significantly to the formation of *samarthah* relation between words. They do so by changing the *aakaamksha* and *yogyata* of words. That is, suffixes act as functions over the FBs corresponding to the root words. They take a FB as an argument and generate another FB as output. So suffixes can be viewed as functions whose domain and range is FBs.

The other important observation is that the condition (i) of *samarthah* is very close to the Subcategorization Principle (SCP) of HPSG (section 2.3.2.5) as far as complements are concerned. HPSG at present deals with complements only and has not yet developed enough theory to handle adjuncts [Poll87, pp 158]. Hence condition (i) which also encompasses adjuncts cannot be fully modelled using the SCP of HPSG (see chapter 2). The problem is that the contribution made to the semantics by adjuncts is not easily expressed through structure sharing alone. This makes it very difficult to express declaratively the semantic contribution of adjuncts. Hence in the rest of the chapter a rather procedural approach is taken in explicating the *samarthah* between words and sentences in various situations. The second condition of *samarthah* does not have any, even, near equivalents in HPSG. The second condition entails compound word formation from complement and adjunct words.

Now the question is, how does one utilise the notion of *samarthah,* defined as above, to develop a parser for a natural language? The key idea is to extend the observation made earlier regarding viewing suffixes as functions. If the different words of the sentence which have *aakaamksha* be viewed as functions over the other words of the sentence having matching *yogyata* and *sannidhi* (proximity), then it would be possible to generate a parsing mechanism centred around these functional applications. Just as the functional application of a suffix change the FB of its argument root word, the functional application of the word with *aakaamksha* to the words with matching *yogyata,* would bring about the necessary changes in *aakaamksha* and *yogyata* of the interacting words. These changed *aakaamksha* would allow sentence, and also compound sentence formation. The parser would have to simulate these functional applications to unravel the semantic content the sentence. Since the parser simulates the functional applications rather than analysing the phrase structure of the sentences, FAP is a novel approach to parsing.

This enterprise of "functional viewing" is tenable only if, in the first place, one is able to conceive of sentences as generated from functional applications. The example in the next section explains how sentences can be generated from functional applications. Once that is done, then parsing can be viewed as applying the FB of one word to the FBs of other words, ultimately culminating in the FB of the full sentence. These functional applications should necessarily be constrained by *samarthah.*

5.3 Viewing sentence generation as functional application

The idea of functional applications can be best explained using an example. Let us look at how complex sentences are generated from the functional application of connective words like "and", "but", and "or" over simpler sentences. For example, conjunctive sentences, are formed by the functional application of *mariyu* (and) on simple sentences (with one verb). The function word *mariyu* (and) takes as arguments the two sentences (x1 y w) and (x2 y w) and generates a new conjunctive sentence as below,

x1 *mariyu* x2 y w' —5.1

where w' is a word derived from w possessing the suitable agreement features to match xl and x2. A more concrete example is given below. To convey the meaning of the following two sentences simultaneously,

రాముడు పార్కులో తిరుగుతున్నాడు
rAmuDu paarkulO tirugutunnADu, --5.2
Rama is roaming in the park

and

సీత పార్కులో తిరుగుతున్నది
seeta parkulO tirugutunnadi. --5.3
Seeta is roaming in the park

one would use the compound sentence

రాముడు మరియు సీత పార్కులో తిరుగుతున్నారు
rAmuDu mariyu seeta paarkulO tirugutunnaaru --5.4
Rama and Seeta are roaming in the park.

The above compound sentence can be thought of a generated from the following functional application of the *mariyu* over the two sentences 5.2 and 5.3

mariyu (rAmuDu paarkulO tirugutunnADu, --5.5
seeta paarkulO tirugutunnadi)

The functional application, due to the nature of the function *mariyu,* generates the sentence 5.4 according to the rule template 5.1.

After viewing complex sentences as above, one can then extend the functional application to the formation of simple sentences and to the constituent words in turn. A word is formed by the functional application of the suffixes over the root word. Postpositions, affixes, verbs, adjectives, adverbs and function words like,

మరియు కంటే కానీ

mariyu (and), *kaMTE (*than), *kAnee* for)

act as functions over their arguments manifested in the form of other sentences, nouns, adjectives, and verbs. Consider further the same sentence 5.4 and its functional form 5.5, but now, with the constituent words morphologically analysed as below.

రాముడూ మరియూ సీతపార్కులో తిరుగుతున్నారు

rAmuDu mariyu seeta paarkulO tirugutunnaaru

rAmuDu+ε mariyu seeta+ε paarku+lO tirug+tun+Aru —5.6

rama+nom+conj and seeta+nom+conj park+loc roam+tense+png

Rama and Seeta are roaming in the park

mariyu (rAmuDu+E paarku+10 tirug+tun+ADu,
seeta+E paarku+10 tirug+tun+adi)

In the above sentence 5.6, e stands for phonemic null character and "png" stands for person-number-gender. Considering only the first sentence,

rAmuDu+ε paarku+10 tirug+tun+ADu,

one can view it as being generated from the following functional applications,

(ADu (tun (tirug))) (e *(rAmuDu), lO (paarku))*

The above functional form implies that the tense affix *tun* fa function) is first applied to the verb *tirug,* to give some intermediate function word *(tirugutun).* Here it may be noted that the results of the functional application include the morphological changes that accompany the word formation, like the inclusion of the letter *u* in between *tirug* and *tun* in the word *tirugutun.* The intermediate function word *tirugutun* can be considered to be the mnemonic for the function *tun (tirug).* The FB corresponding to *tirugutun* is also generated in a similar vein by applying the function *tun* over the FB of *tirug.* To

continue, the png (person-number-gender) affix *ADu* (a function again) is applied to the above intermediate function word *(tirugutun),* leading to yet another function word *(tirugutunnADu).* The functional representation of which is *ADu (tun (tirug)).* This function is applied to the arguments resulting from the application of the case functions e and *lO* to *rAmuDu* and *paarku* respectively. Thus the complete functional representation of the sample sentence is

mariyu ((ADu (tun (tirug))) (e *(rAmuDu), lO (paarku))*
(adi (tun (tirug))) (e (*seeta*), *lO (paarku)))*

The above functional form is responsible for generating the sentence S1. Therefore it may be seen that every sentence is generated from and has an internal functional form unique to it. Alternately, every sentence is generated due to certain functional applications between its constituent words.

From a mathematical point of view, the notion of functional application given here, in effect, is similar to the functional application in languages like the λ–calculus, and the Lambek calculus [Vanb87]. The notion of unification is not used in those systems. Whereas here, the notion of functional application rests totally on unification and structure sharing. Functional application here affects both the syntactic and semantic structures associated with words. Whereas in Lambek Calculus functional application is used only to model semantics. The concept of *sannidhi* (proximity) is similar to the adjacency requirements needed to be met for functional composition to be possible in Categorial Grammars [Vanb87, Stee88].

5.4 Functional Application Parsing

The idea of 'functional application parsing' (FAP) used in TELANGANA originated from the close relationship that exists between *samarthah* theory and sentence formation through functional applications. To analyse the syntax/semantics of sentences, FAP tries to mimic the functional applications that occur during sentence formation.

FAP deals with FBs. The details of how the FB structure of a simple sentence is analysed by FAP, as derived from the FBs of its constituent words through functional applications, is given in the ensuing example. The way a word, X, acts as a function over the other words in proximity *(sannidhi)* largely depends on the syntactic and semantic classification of the word, X, and its neighbours in the sentence. As already noted, the various inflections and suffixes on words are treated as functions. For example, the word

rAmuDu is formed by applying the function corresponding to the null suffix ε to the FB corresponding to *rAmuDu*. Similarly the *'iMc'* suffix in the word *'ceppiMcADu'* (*cepp*=tell, *iMc*= caused to, *ADu*= 3rd person singular), is a function that takes the feature bundle (FB) related to *cepp* as argument and produces a FB related to *'ceppiMc'*. This functional application is responsible for modifying the *aakaamksha* of *cepp* so as to subcategorize for a PP with the suffix *cE*, instead of a PP with the suffix *ni* which *cepp* does. The *'ADu'* ending in turn takes up the resulting FB and produces a new FB corresponding to *'ceppiMcADu'*. When a partial sentence like

సీతచే చెప్పించాడు
seetacE ceppiMcADu
Sita-by was made tell
(Some man) made Sita tell

is considered, the FB of the word *'ceppiMcADu'* is applied to the FB of *seeta+cE* (where A+B means, apply B to A) giving rise to a new FB whose semantic content would be "Some man made Sita tell something". In this way, the FB of an entire sentence can also be formed. Let us take the following full sentence, as an example.

రాముడు సీతచే పూలని కోయించాడు
raamuDu seetacE poolani kOiMcADu
raamuDu+E seeta+cE puvvu+lu+ni kOy+iMc+ADu
raamuDu-nom seeta-dat flower+png+acc pluck+made+png
Rama made Sita pluck some flowers

The verb *kOy* (pluck) has *aakaamksha* (expectancy) for an animate independent agent in nominative case, and an inanimate[1] object in accusative case. More formally the *aakaamksha* of *kOy* is (substantially simplified for pedagogical reasons)

aak: [nom:X:animate, ni:Y:inanimate] --5.7

The expression in 5.7 is part of a FB that is associated with the word *kOy*. Note how the information regarding nominative/accusative cases, their corresponding surface realisation, and the information regarding animate/inanimate objects are encoded in 5.7.

[1]The sorts, animate and inanimate, should be understood as short forms of the actual complete sortal formulae given in Chapter 3.

In the above formula, 5.7, the symbols in capital letters, X and Y, are variables or place holders to capture the semantics of *yogyata* elements that are going to satisfy this aakaamksha.

The suffix *iMc* acts as a function on the *aakaamksha* of the verb *kOy* and generates a new *aakaamksha,*

aak: [nom:X:animate, cE:Y:animate,ni:Z:inanimate] --5.8

It may be noted that in this FB fragment, surface case markers have changed significantly. The suffix png, in turn acts on 5.8 to give

[aak: [nom:X:animate, cE:Y:animate,ni:Z:inanimate]

agr: [m,s,3]] -5.9

In the above FB fragment, the additional information regarding agreement features is present. The yogyata of *raama, seeta* and *puvvu* are respectively,

yog: [isa:R:animate], yog: [isa:S:animate], and yog: [isa:P:inanimate]

The postfix *lu* applied to *puvvu* gives,

yog: [isa:P:inanimate,agr: [f,pl,3]]

The postfixes, e, *cE* and *ni,* when applied to *raama, seeta* and *puvvu+lu,* give

yog: [isa:R:animate,agr: [m,s,3],post:nom], yog: fisa:S:animate,post:cE],

and

yog: [isa:P:inanimate, agr: [f,pl,3],post:ni] --5.10

Finally when 5.9 is applied to 5.10, the resulting *aakaamksha* of all the words put together becomes aak: []. Since the resulting *aakaamksha* is null, the sentence is accepted.

In brief, this is how TELANGANA parses sentences and builds the associated FB structures. The FB structures hold both the functional and the semantic structures of a sentence. Functional application parsing extracts the functional form and the corresponding FB structure of a given sentence, from its surface representation as a sequence of words.

It should be noted that *samarthah, aakaamksha* and *yogyata* have been used in TELANGANA as constraints on sentence formation. They do not drive the analysis process as was done in the expectancy based parsers of Schank's conceptual dependency theory [Scha75]. The notion of head driven parsing as used in the HPSG parsers is also not used in TELANGANA. In HPSG parsers the head word is located first and then the complements are linked to the head words [Prou85]. The parsing mechanisms in TELANGANA were developed to be close in spirit to AST. In AST the notion of heads does not appear and consequently all grammatical entities are of equal status. Hence parsing has to be done using a system that has no concept of hierarchy either structural as in GB-theory or non-structural as in HPSG. FAP provides such a parsing capability. The parsing in FAP is very much like that of CG [Stee88] grammars, but additionally uses notations like *aakaamksha* (expectancy), *yogyata* (ability) and unification not used in standard CG grammars.

Using this parsing scheme as its parsing engine, TELANGANA was developed. Presently TELANGANA can parse and build the semantic representation for a variety of Telugu sentence structures consisting of, simple sentences, copula sentences (verb-less sentences), relative clauses, comparatives sentences (including clausal comparatives, adjectival and adverbial comparatives, superlatives), some amount of intra-sentential anaphora and limited conjunctions. TELANGANA parses all these varied types of sentences in a uniform and a simple way. In this chapter, analysis of only simple sentences is presented while the analysis of more complex sentence structures is presented in the next chapter. Material in the next chapter presupposes a good understanding of the parsing technique detailed below.

5.5 Parsing Simple Sentences

A sentence with at most one verb and not containing any conjunctions and comparative constructs, is considered to be a simple sentence in this thesis. Thus assertive sentences, interrogative sentences, and exclamatory sentences, with one verb are covered. Stative sentences with no explicit verb like the following are also simple sentences.

రాముడు అందగాడు
rAmuDu aMdagADu
Rama is handsome

సీత పొడుగు
seeta poDugu
Sita is tall

Simple sentences can also be characterized as passive sentences, causative sentences, hortative sentences or copula sentences depending on the verb aspects. By simple sentences, all such verb aspectuals are also implied. Simple sentences could contain a verb (explicitly or by implication), noun complements and optionally adjuncts. If a sentence contains only complements (and no adjuncts), then it would contain only those words that are related to one another by ***aakaamksha/yogyata*** relations. Thus such sentences consists of only verb complements and adjectives qualifying some nouns. From the parser point of view such sentences are the simplest to analyse. Hence parsing such sentences will be considered first.

The following are some typical simple sentences of Telugu consisting of only complements

రాముడు సీతని పార్కులో చూశాడు
rAmuDu *seetani pArkulO cooSADu*
Rama saw Sita in a park.

రాముడికి రసాయనశాస్త్రంలో ఎన్ని మార్కులు వచ్చాయి?
raMuDiki *rasAyanasAstraMlO* ***enni mArkulu*** *vaccAyi?*
How many marks did Rama get in Chemistry?

రాముడు సీతకి రాధని చూపించాడు
rAmuDu seetaki rAdhani ***coopiMcAdu.***
Rama showed Radha to Sita.

రాముడు సీతచే అన్నం తినిపించాడు
rAmuDu seetacE ***annaM*** *tinipiMcADu*
Rama made Sita eat food.

రాముడు ఇంటికి వెళ్లాడే!
rAmuDu ***iMTiki*** *veLlADE!*
oh! Rama has gone home!

In all the above sentences the verbs have *aakaamksha* for the rest of the words. From the above examples, it may be observed that the variety[2] of sentences, that can be constructed using complements alone is very large.

Frequently, what goes into the *aakaamksha* is decided by domain dependent factors. In the domain of relational database access, the entities in the fields of a tuple are the likely candidates to enter the *aakaamksha* of the verb describing the database relation. For example in the case of a part-supplier kind of a relation, it is easy to visualise a description of the relation using a sentence like

ఈసీఐఎల్ 1992లో గుంటూరుకి హైదరాబాదునుంచి 2000టీవీలు సరఫరా చేసింది
ECIL 1992lO guMTooriki haidarAbAdunuMci 2000 TVlu saraphara cEsiMdi
ECIL supplied 2000 TVs to Guntur form Hyderabad in 1992.

Hence the *aakaamksha* of the word *saraphara* (supply) would have entries containing suffixes such as *nuMci, lO,* and *ki.* Sentences of this restricted nature also capture a wide variety of database relations. Hence *aakaamksha* is to some extent domain dependent. But in this chapter only linguistically motivated *aakaamksha* is presented. Linguistically motivated *aakaamksha* tends to have more generality in its extent of coverage of a language and thus tends to be more application neutral. Finding the linguistically general *aakaamksha* of a word requires consulting large corpora of sentences to see the typical usage of the word. When it comes to using such aakaamksha in some domain of application, one needs translation rules to get the specific connotation of the word in the given domain of application (see Chapter 7).

5.5.1 Main Procedures of the Parser

The parser of TELANGANA processes a sentence left to right. It reads in each word, gets its FB from the lexicon and collects the FBs into a list, L, if it finds these words not related through some *aakaamksha/yogyata* relationship. This process continues until a verb is read. Verbs typically have an "aak" feature-value such as "1: [...] indicating that **they have expectancy** for some words that occur to their left, that is earlier in the sentence. The parser matches the elements of the list, L, with this "aak" feature value of

[2]Semantic variety in terms of the number of relations implied by complements is potentially infinite. Syntactically, complements do not show much variety, in the sense that all complements are dominated by the *aakaamksha* word and hence have a simple structure.

the verb. If any word appears in the list, L, whose *yogyata* does not match with any value in the "aak" feature of the verb, then all the words that appear to the left of the non-matching word are treated as not related to this verb. The rationale for this decision being that the words may belong to the matrix sentence[3] if one exists, or they may be adjunct words, or else they may be extraneous words. If they were to be extraneous words, the parser has to ultimately reject the sentence as ungrammatical. TELANGANA has been designed to accept even erroneous input sentences as much as possible and construct their semantic structure to the best possible detail. However all sentences that have extraneous words are tagged as ungrammatical.

The following fragment of Prolog code exemplifies the above kind of processing done by the parser. The clause parse (SemIn-SemOut, SenIn-SenOut), takes as input a difference list, Senln-SenOut, encoding the sentence and generates as a difference list, SemIn-SemOut, encoding the feature bundle (FB) representing the meaning of the sentence in the meaning representation language (chapter 3). The clauses describing the body of parse procedure are

```
parse (Z-Z, []- []).

parse (SemIn-SemOut,SenIn-SenOut) :-    lex_sem (SemIn, SenIn-X, FB),
                                        apply (SemIn-Out1,FB),
                                        parse (Out1-SemOut,X-SenOut).
```

The procedure lex_sem takes a difference list Senln-X as input, removes the first word in it, and outputs its, FB, stored in the lexicon. The first argument, Semln, in case of ambiguity, provides the neccessary semantic context which can be used by lex_sem to disambiguate the word and retreive the correct FB. The input word to lex_sem is in the form of root+suffixes. The definition of the lex_sem procedure contains the necessary clauses to compute the functional applications due to the various suffixes. The definition of the lex_sem procedure consists of clauses such as the following.

```
lex_sem(L,[XIY]-Y,FB):- lex-sem1(L,X,FB).
```

[3] A sentence that embeds another sentence is said to be a matrix sentence. The verb of the embedding sentence is called the matrix verb.

```
lex-seml(L,X,FB)        :- atom(X),!, lookup_lex(X,FB).

lex_seml(L,X+lu,FB)     :- lex_sem1(L,X,FB1), change_agr(FB1,FB).

lex_sem 1 (L,X+Z,FB) :- noun_suffix(Z), lex_sem 1 (L,X,FB 1),
                        insert(post:Z,FB,FB1).

lex-sem1(L,X+Z,FB)      :- verb_suffixes(Z), lex_sem1(L,X,FB1), gnp(Z,GNP),

                        insert([agr:[GNP,J,post:Z],FBl,FB).
```

As shown above, lex_sem computes the FB of a whole word from its constituent root word's FB.

The procedure apply (SemIn-Out1,FB), applies the FB to the already built up semantic structure of the sentence seen so far. When the FB is not related to the rest of the already seen words through an *aakaamksha/yogyata* kind of relationship, then this FB is simply appended to the SemIn giving Out1 as [FB|SemIn]. However if the new FB has an "aak" feature value as "1: [...]" then, the elements of Semln could be examined to see if their *yogyata* matches with the *aakaamksha* of the new word. This is done using a procedure called 'fillall". The Prolog clauses that achieve the behaviour of the procedure apply, just described, are given below.

```
apply ([]- [FB],FB).

apply (SemIn-SemOut, [aak:l:AIT]):- fillall (SemIn-SemOut, [aak:l:AIT]).

apply (Semln- [FBISemln]).
```

The procedure apply has many more clauses in its definition, but the above three are relevant for parsing sentences of the complexity being considered in this section. The procedure fillall takes care of checking the first condition in the definition of *samarthah,* and the procedure apply takes care of checking the second condition.

The procedure fillall, takes each element of the input difference list, Semln-SemOut, and tries to match it with the elements of "aak" feature. The procedure fillall has many clauses defining it. To simplify matters, only those clauses related to complements are considered first. As the sections progress, further additions/modifications will be made to the defining clauses of these two procedures to cater to more complicated cases. The

procedure fillall/2, which has two arguments, is defined in terms of another procedure by the same name, fillall/4, that takes four arguments[4].

fillall (X-XO, [aak:l:AinlT]):- fillall (X-XO, Ain-Aout, [aak:l:AinlT]- [],AI).

fillall (X-X,A-A, []- [],_).

fillall ([]- [[aak:l:A|VF]],A-A, [aak:l:JVF]- [],_).

fillall (X-XO,AI-AO,VF-VFO,AAK) :- fill (X-Y,AI-A1,VF-V1,AAK),

fillall (Y-XO,A1-AO,V1-VFO,AAK).

The first argument of fillall/4 is the difference list consisting of the list of FBs accumulated till the point where fillall was called. The second difference list captures the *aakaamksha* of the verb, and is the list against which the elements from the first argument list are matched. The third difference list indicates the input FB and the output FB of the verb. The fourth argument is useful for processing comparative sentences.

5.5.2 Handling Sentence Type Information

The FB of the verb undergoes some modifications pertaining to the type of the sentence as complement PP are matched against its *aakaamksha.* The third argument of fillall is meant to keep track of these changes. In Telugu the sentence type, in terms of assertive, interrogative, hortative and so forth, encoded by the feature "sentype", depends on the type of complements and adjuncts that are present in the sentence. This is unlike English where the syntax decides the type of the sentence to a great extent. For example, in English, subject-auxiliary inversion in a sentence makes the sentence interrogative. In English minor suffix variations in the prepositional phrases do not have any effect on the type of the sentence. On the other hand, in Telugu, the sentence type changes drastically even by minutely modifying some of the PPs or the quantifiers. For example, in the following sentences, the first one is the assertive sentence. The others are interrogative sentences formed by just changing some suffixes or quantifiers in the first sentence.

[4]This technique of using the same procedure with different number of arguments is common in Prolog programming [Ster86] and helps in information hiding. To disambiguate the procedure name the arity (number of arguments) of the procedure is appended to the procedure name. For example fillall/2 and fillall/4 refer to the procedure fillall with two arguments and four arguments respectively.

రాముడు సీతకి పార్కులో చాలా పూలు చూపించాడు
rAmuDu seetaki pArkulO cAlA poolu coopiMcADu
Rama showed many flowers to Sita in a park

రాముడు సీతకి ఏపార్కులో చాలా పూలు చూపించాడు?
rAmuDu seetaki EpArkulO cAlA poolu coopiMcADu?
In which park did Rama show Sita many flowers?

రాముడు ఎవరికి పార్కులో చాలా పూలు చూపించాడు?
rAmuDu evariki pArkulO cAlA poolu coopiMcADu?
To whom did rama give many flowers in the park?

రాముడు సీతకి పార్కులో ఎన్నిపూలు చూపించాడు?
rAmuDu seetaki pArkulO ennipoolu coopiMcADu?
How many flowers did Rama show Sita in the park?

రాముడు సీతకి పార్కులో చాలా పూలు చూపించాడా?
rAmuDu seetaki pArkulO cAlA poolu coopiMcADA?
Did Rama show many flowers to Sita in the park?

*రాముడు ఎవరికి పార్కులో చాలా పూలు చూపించాడా?
* *rAmuDu evariki pArkulO cAlA poolu coopiMcADA?*
*Did Rama show many flowers in the park to who?

*రాముడు సీతకి పార్కులో ఎన్ని పూలు చూపించాడా?
* *rAmuDu seetaki pArkulO enni poolu coopiMcADA ?*
*did Rama show Sita how many flowers in the park.

As indicated above, not all modifications are grammatical. The last two sentences are ungrammatical owing to the wrong interaction of the sentype feature bearing entities, and the tense/aspect and the modality of the verb. In the parser, initially, the type of the sentence, sentype, is guessed to be "none". Then as parsing progresses the guessed value of sentype is changed as each new complement is processed. Any complement can potentially change the sentype value. The last obtained value of the feature sentype is

matched against the sentype feature of the verb. If they match the sentence is grammatical as far as sentence type information is concerned.

An attempt could be made to model the feature value, sentype, as a foot feature in the GPSG [Gazd85] style as explained in Chapter 2. However it is not possible. This can be seen from the following two sentences.

*రాముడు ఎవరినీ ఎక్కడ చంపలేదు
**rAmuDu evarinee ekkaDa caMpalEdu*
Where did Rama kill nobody ?[5]

రాముడు ఎక్కడ ఎవరినీ చంపలేదు
rAmuDu ekkaDa evarinee caMpalEdu
Where did Rama not kill anybody?

The two sentences are variants in terms of the word order only. The first one is ungrammatical, where as the second one is grammatical. If unification in the GPSG sense is used for the propagation of foot feature information from the daughter categories to the mother category, then both cases will be grammatical because the foot feature principle (FFP) of GPSG envisages a unification of daughter features as being propagated to the mother. Unification is order independent. Hence the feature sentype is not a foot feature. Neither can it be a head feature, because then information flows to the mother category only from the head daughter. But in the above sentences, sentype information seems to originate from every complement daughter and percolate to the head category. Hence sentype is neither a head feature nor a foot feature, nor a combination of them. Hence the feature sentype requires a special treatment as done here. The behaviour of the sentype is modelled using a finite state machine (FSM). As every complement is processed, it changes the value of sentype guided by a state transition like table. This takes care of the temporal order of the complements in deciding the grammatical well formedness of sentences.

5.5.3 Matching *aakaamksha* and *yogyata*

The procedure fill, which was used in the description of fillall, is defined below.

[5]This sentence is not properly translatable

fill (X- [aak:l: []|T]X], []- [], [aak:l:AIT]- D,_).

fill ([X|Y]-Y,AI-AO,VF-VFO,_) :- fill1 (X,AI-AO,VF-V1,_),

agree_sentype (X,V1,VFO).

fill1 (X,AI-AO,VF-VF,_):- (mem (pos:n,X);mem (pos:pro,X)),

mem (post:PP,X),mem (isa:ISA,X),

compl (PP:X:ISA,AI-AO).

The procedure agree_sentype in the fill procedure achieves the sentence type agreement as explained. The procedure fill calls, ultimately, the procedure compl which really does the matching between the *aakaamksha* of the verb and the *yogyata* of the complement PPs. The procedure fill 1 in the definition of fill, extracts the postpostion information in each complement PP, appends that information to the sortal type, ISA, of the entity represented by the complement and passes it on to the procedure compl. There is one more clause in the definition of filll, which handles adjuncts. It will be discussed later on.

The procedure compl, unlike the procedure agree_sentype, does not modify the FB of the verb, but uses it to deposit the FBs representing the complements into the "sem" feature of the verb by using unification of Prolog. It also handles the agreement between the subject and the verb of the sentence. There are some interesting aspects to the definition of the compl procedure. It has to deal with cases where postpositions are not present even though they should be. Normally words in nominative case do not have any postpositions. However, accusative and possessive cases are also many times expressed with out a suitable case marking suffix. In the following three sentences

శివ రాధకి చాలా పూలు ఇచ్చాడు
*Siva rAdhaki **cAlA poolu** iccADu*
Siva gave many flowers to Radha

శివ అన్నం తినలేదు
*siva **annaM tinalEdu***
Siva did not eat food

శివ తన తండ్రి ఫొటో చూశాడు

siva tana taMDri phoTO cooSADu

Siva saw his father's photo

the words *siva, poolu, annaM,* and *taMDri* have occurred with out any postpositions. The word *siva* is in nominative case, the words *poolu* and *annaM* are in accusative case, and the word *taMDri* is in possessive case. So one needs some heuristics to predict their missing postposition/case marking suffixes for "subj", "i" and "ni". It may be noted that the problem of missing postpositions occurs in common spoken Telugu and to a very little extent in traditional *granthika* (scholarly) Telugu. Anybody who attempts to build a parser for the "Telugu in vogue" or "Newspaper Telugu" will meet with this problem. The simple heuristics used in TELANGANA are to treat a missing postposition on a word representing

- an in-animate object as a "ni",
- an animate object as "subj",
- a relational words like *taMDri* (father), *talli* (mother) as "dflt_i" and,
- an "a" ending word like *siva, seeta* etc as "dflt_i"

The above, being only heuristics, can fail. But on a majority of cases these four heuristics seem to work very well. The postposition "dflt_i" can lead to formation of noun-noun relations. For example in the sentence fragment *"seeta taMDri phoTO" seeta* and *taMDri* are needed to have "i" as the postposition for the noun-non relations *"seeta taMDri"* and *"taMDri phoTO".* But they do not have the "i" suffix due to morphological reasons. The post position "dflt_i" can work in lieu of "i". "dflt_i" can also be treated as "subj" if the word in question cannot take part in a noun_noun relation. Noun-noun relations are analysed by the parser before matching the *aakaamksha* of verbs and *yogyata* of complements. The clauses defining the procedure apply which handle noun-noun relationships look for a possible relationship between a word missing the requisite "i" postposition and the adjacent noun to see if the words in question can be related. If so the problem of determining if a word is missing an "i" suffix or is truly a subject ends there itself. Otherwise the problem percolates to the compl procedure. In the procedure compl sortal entires in the *aakaamksha* help resolve the dilemma. If the dilemma is not handled even at this stage then the sentence is tagged as having an extra complement and hence ungrammatical.

An example of a different nature, where a missing "subj" case is wrongly assumed to be noun-noun relation, is very easy to construct The sentence

సీత కుర్చీకి రంగు వేసింది
seeta kurceeki raMgu vEsiMdi
Sita painted the chair

రాముడిచే సీత కుర్చీకి రంగు వేయించాము
rAmuDicEsita kurceeki raMgu vEYiMcAmu
(We) made Rama paint Sita's chair.

illustrates the point where the missing "subj" case of sita could potentially be a missing "i". This is exactly opposite to the dilemma discussed above where in a missing "i" is wrongly assume to be "subj". Backtracking in Prolog handles this problem easily. If the missing case marker, is assumed to be "i", then a noun-noun relation is established between the words. The FB of this relation will not have suitable *yogyata* to match the *aakaamksha* of the governing verb. Hence the compl procedure would cause a backtracking, and the assumption of missing "i" is reverted to missing "subj" and the parsing proceeds this time correctly.

In case of sentences with multiple verbs, the procedure compl may come across some complements which do not match the *aakaamksha* of the verb. Then there are two possibilities. The first one is to treat the new word as an adjunct. So compl tries to see if the new word can indeed be an adjunct to the verb. If so, the matching proceeds further on. In the second possibility, the new word is not an adjunct, the matching process is stopped and compl returns all the words of the input list as words not belonging to the current verb. Such an action is justified as can be seen from the following examples.

రాముడు సీతకి ఇంట్లో ఉంటానని చెప్పాడు
rAmuDu seetaki iMTlO uMTAnani ceppADu
Rama told Sita that he will be at home

రాముడు రాధకి బొమ్మలు కడిగి ఇచ్చాడు
rAmuDu rAdhaki bommalu kaDigi iccADu
Rama washed the toys and gave to Radha

రాముడు ఆ లెక్కగురించి బస్సులో వెళ్ళ్లా ఆలోచించాడు
rAmuDu aa lekkaguriMci bassulO veLltoo alOciMcADu
Rama thought about that problem while going in the bus.

The verbs *uMTAnu, tuDici,* and *veLltoo* do not have *aakaamksha* for the words *seetaki, rAdhaki,* and *lekkalaguriMci* respectively. The procedure compl, quite rightly, stops the process of matching the moment the words *seetaki, rAdhaki,* and *lekkalaguriMci* are encountered while processing the above sentences indicating that these words are not complements to the verbs *uMTAnu, tuDici,* and *veLltoo* irrespectively. However this frequently leads to some of the mandatory feature values in the *aakaamksha* of the verb remaining unfilled. If the number of unsatisfied entries in the *aakaamksha* is more than two, then the sentence is ill-formed. Under some circumstances, more than one unsatisfied entries, as in the above sentences, can also lead to ungrammatical sentences. These cases are further discussed in the sections on relative clauses and compound sentences.

5.5.4 Handling Adjuncts

The complements account for a significant fragment of Telugu sentences; through structures in which every constituent is either an *aakaamksha* word or a complement. But not all constituents of a sentence belong to these two categories. Relative clauses, adjectival modifiers of nouns, adverbial modifiers of adjectives and verbs, and various sentential adjuncts clearly can not be reduced to the above two categories. To cater to the presence of such entities, the notion of *uthita-aakaamksha* (aroused-expectancy) was explained in Chapter 1. In English, the concept of adjuncts is comparable. An adjunct is some entity, a word or a group of words, X, which is loosely dependant on the *aakaamksha* of another word or group of words, W, in the sense that (i) the syntactic/semantic categories of X and W are mutually constraining, and (ii) the contribution of X to the semantic content of W is other than filling in some *aakaamksha* entry of W. This should be seen in the contrast to the tighter relation between *aakaamksha* and *yogyata.*

It is important to be aware that optional complements are not adjuncts. Examples of adjuncts in Telugu are given below. The underlined words of the sentences form the adjuncts.

(a) subordinate clauses:

రాముడు లెక్కలు చేసినప్పుడు/చేయటం వలన సీత సంతోషించింది
rAmuDu lekkalu cEsinappuDu/cEyaTaMvalana seeta saMtOshiMciMdi
Sita was happy **when/because** Rama solved math problems.

(b) predicative clauses

రాముడు కలెక్టర్ అవ్వటంతో సీత కష్టాలు తీరాయి
rAmuDu kalekTar avvaTaMtO Seeta kasTAlu teerAyi
With Rama becoming a collector, **Sita's** problems came to an end

(c) adjuncts of purpose

రాముడు క్రికెట్ ఆడటానికి బాట్ కొన్నాడు
rAmuDu krikeT ADaTAniki bAT konnADu
Rama bought a bat to play cricket

(d) manner adverbials

రాముడు హడావిడిగా/మెల్లిగా/త్వరగా ఇంటికి వెళ్ళాడు
rAmuDu haDAviDigA/melligA/tvaragA iMTiki veLLADu
Rama went home hurriedly/slowly/quickly

(e) frequentatives

రాముడు నాలుగు సార్లు పార్కుకి వెళ్ళాడు
rAmuDu nAlugu sArlu pArkuki veLlADu
Rama went to park four times.

(0 duratives

రాముడు 10 గంటలు అన్నం తిన్నాడు
rAmuDu 10 gaMTalu annam tinnADu
Rama ate food for 10 hours

(g) relative clauses

రాముడుకి సీత కొన్న బిళ్ళలు నచ్చలేదు
rAmuDuki seeta konna biLlalu naccalEdu
Rama did not like the toffees which Sita bought

(h) adjectives

రాముడు వృద్ధ / ముఖ్య/ మాజీ మంత్రిని చూశాడు
rAmuDu *vrudha/mukhya/mAjeemaMtrini cooSADu*
Rama saw the old/chief/former minister

Although there is no general agreement as to how the difference between adjuncts and complements should be characterized in theoretical terms, there are a few rough semantic and syntactic diagnostics to make the distinction [Poll87].

Order-dependence of meaning: The order of complements does not alter the meaning of the sentence. Whereas the order of adjuncts does frequently alter the meaning of the sentence. The following three sentences mean the same, inspite of word order variation. The underlined words are the complements of *iccADU.*

రాముడు సీతకి బిళ్లలు ఇచ్చాడు
rAmuDu seetaki biLlau iccADu
seetaki rAmuDu biLlalu iccADu
biLl alu rAmuDu seetaki iccADu
Rama gave toffees to Sita

The following sentences containing adjuncts are not so immune to order variation of constituents.

రాముడు రోజుకి రెండు సార్లు బిగ్గరగా మాట్లాడతాడు
rAmuDu rOjuki reMDu sArlu biggaragA mATlADatADu
Rama talks twice a day loudly

రాముడు బిగ్గరగా రోజుకి రెండు సార్లు మాట్లాడతాడు
rAmuDu biggaragA rOjuki reMDu sArlu mATlADatADu
Rama talks loudly twice a day.

The first sentence entails that Rama possibly talks many times a day but only twice does he talk aloud. The second sentence could possibly entail that the manner of Rama's talking is loud and that he talks only two times a day. Another example sentence can bring out this difference further.

రాముడు ఆరోజు తెలివి తక్కువగా మాట్లాడాడు
rAmuDu ArOju *telivi takkuvagA* ***mATlADADu.***

Rama talked that day foolishly.

తెలివి తక్కువగా రాముడు ఆరోజు మాట్లాడాడు

telivi takkuvagA rAmuDu ArOju mATlADADu

Foolishly, Rama talked that day

The second sentence given above entails that it was foolish of Rama to talk on that day.

Multiple usage: Adjuncts with the same suffix can be used multiple times in a sentence but this is impossible for complements.

రాముడు జులైలో మద్రాసులో తాజ్ హోటలులో సీతని కలిశాడు

rAmuDu julailO madrAsulO tAj hOTElO seetani kaliSADu

Rama met Sita in Madras in Taj hotel in July

రాముడు ఇంట్లో చెప్పకుండా భయపడకుండా సినిమాకి చెక్కేశాడు

rAmuDu iMTlO ceppakuMDA bhapaDakuMDA sinimAki cekkESADu

Rama went to a cinema without telling at home without feeling afraid

రాముడు మెల్లిగా దొంగచాటుగా వంట గదిలోకి దూరాడు

rAmuDu melligA doMgacATugA vaMTa gadilOki dooRDu

Rama slowly, stealthily entered into the kitchen.

Difficulty in **relativisation:** Many adjuncts do not allow relativisation. Some adjuncts allow relativisation but require that a quantifier be additionally present. many complements are much more flexible and allow relative clause formation around them. For example,

రాముడు కలంతో పరీక్ష రాశాడు

rAmuDu kalaMtO pareeksha rASADu

Rama wrote the exam with a pen

రాముడు పరీక్ష రాసిన కలం ఇది కాదు

rAmuDu pareeksha rAsina kalaM idi kAdu

The pen with which Rama wrote the exam is not this

In the above sentence the PP *kalaMtO* (Pen) is a complement of *rAy* (to write). Hence it can lead to the gap and consequently to relativisation as in the second sentence above. In the following sentence *jvaraMtO* is an adjunct and it cannot be relativised.

రాముడు జ్వరంతో పరీక్ష రాశాడు
rAmuDu jvaraMtO pareeksha rASADu
Rama wrote the exam with fever

*రాముడు పరీక్ష రాసిన జ్వరం ఇది కాదు
**rAmuDu pareeksha rASina jvaraM idi kAdu*
This is not the fever with which Rama wrote the exam

This gap diagnostic for distinguishing adjuncts/complements is not fool proof.

Constancy of meaning: In general, a given adjunct can co-occur with a relatively broad range of *aakaamksha* words while seeming to make a more-or-less uniform contribution to the semantic content across the range. A given complement is typically limited in its distribution to co-occur with a small class of *aakaamksha* words only. In addition the semantic contribution of complements is idiosyncratically dependant on the *aakaamksha* of the head word. For example the PP with suffix *meeda* is a locative adjunct and combines with many verbs to describe the location of the event/state of affair. In the first sentence below, the PP *pakkameeda* is an adjunct giving the location of the activity.

రాముడు పక్కమీద పాడుతున్నాడు/ఎగిరాడు/ఉన్నాడు/ఆలోచిస్తుంటాడు
rAmuDu pakkameeda pADutunnADU/egirADU/unnADU/AlOcistuMTADu
Rama is singing/ jumped/ is/would be thinking on the bed.

On the other hand, in the following sentences it is a complement and the meaning contribution is idiosyncratic to the verb.

రాముడు సీతమీద నేరం మోపాడు
rAmuDu seetameeda nEram mOpADu
Rama put the blame on Sita

రాముడు (ఈ వ్యవహారం తీర్చి దిద్దటానికి) సీతమీద ఆధారపడ్డాడు
rAmuDu (ee vyavahAraM teerci diddaTAniki) seetameeda adharapaDDADu
Rama depended on Sita (to set right this affair)

రాముడు సీతమీద గౌరవంతో ఈ పనికి ఒప్పుకున్నాడు
rAmuDu seetameeda gouravaMtO ee paniki oppukunnADu
Rama agreed to do this work due to his respect for Sita

The above four diagnostics were used in TELANGANA to decide if a particular PP is an adjunct or a complement. Similar criteria for selection of adjuncts are not specified in traditional grammar books of Telugu [Kris'85, Cinn51]. Of all the above criteria, the last one was exploited most for computing the semantic contribution of adjuncts to the corresponding *aakaamksha* words. If a PP is a mandatory complement it goes into the *aakaamksha* of the verb or noun with the tag "+man". If the PP is an optional complement then it is indicated with the tag "+opt" as already explained in Chapter 3. If a PP is judged to be an adjunct, it is represented as a procedures (to be precise, Prolog facts) using either the functor verb_mod or noun_mod depending on its type.

In the parser, when a PP is being analysed one needs to decide if (i) the current PP is a complement of the next word (PP), or (ii) an adjunct of the next word, or (iii) a complement of the verb or (iv) an adjunct of the verb of the present sentence (or clause). This is the so called PP attachment ambiguity. In Chapter 2, some cases of PP attachment ambiguities were highlighted and it was indicated that PP ambiguity resolution is one of the difficult areas of NLP. The procedures verb_mod and noun_mod help the parser in deciding which word the PP modifies. Let PP_2 and PP3 be two adjacent PP in a sentence. Let S_i be the sortal category of main word in PP_i and P_i the suffix present in PP_i Then the structure of noun_mod is

noun_mod (P_2, X^S_2, Y^S_3, SEM).

SEM, in the above Prolog fact, indicates the semantic contribution of PP_2 to PP3. X and Y are the variables of the FBs representing PP_2 and PP3. Typical examples of noun_mod are as follows. The following predicate in the definition of noun_mod

noun_mod (meeda,X^place,Y^physical,location (Y,X))

helps in analysing the relationship between an object and a physical place manifested by the use of the postposition *meeda* in the following sentence.

రాముడు బల్లమీద ఉత్తరం చదివాడు
rAmuDu ballameeda uttaraM cadivaDu

Rama read the letter on the table

Similarly, the following predicates of noun_mod

noun_mod (meeda,X^entity,Y^propositional,content (Y,X))

can be used to analyse the relationship between an entity and a propositional object as manifested by the use of the postposition *meeda* in the following two sentences.

రాముడు తెలుగు వ్యాకరణంమీద పుస్తకం రాశాడు
rAmuDu Telugu vyakaraNaMmeeda pustakaM rASADu
Rama wrote a book on Telugu grammar

రాముడు సీతమీద చాడీలు చెప్పాడు
rAmuDu seetameeda caaDeelu ceppADu
Rama complained about Sita

The following predicate definition caters to the case where the postposition *lOni* is used to relate an enity of sortal type "animal" and another entity of sortal type "mental_state"

noun_mod (lOni, X^animal, Y^mental_state, mental_state_of (X,Y))

It can be used to analyse the following sentence.

రాముడు మనుషులలోని భయాలను హిప్నాటిజంద్వారా వెలికి తెప్పిస్తాడు
rAmuDu manushulalOni bhayAlanu hipnoTisamdvArA velki testADu
Rama brings out the fears in persons through hypnotism

Similarly in the case of verb and PP attachment, the verb_mod procedure is defined as

verb_mod (P_2,X^S_2,Y^SV_1,SEM).

wherein X, P_2, S_2 and SEM convey the same meaning as before. The third argument, SV_1, represents the sortal hierarchy of the verb. The variable Y represents the *samdarbham/sthiti* (event/state) variable of the verb. Typical examples of the predicates comprising of the definition of the verb_mod procedure and the example sentences they can be used to analyse are given below.

verb_mod (ki, X^time,Y^_, time_point (Y,X))

రాముడు సీతకి 10 ఇంటికి పూలు ఇచ్చాడు

rAmuDu seetaki 10 iMTiki poolu iccADu

Rama gave flowers to Sita at 10 O'Clock.

verb_mod (tO,X^mental_state,Y^social_Act, manner (Y,X))

రాముడు సీతని ప్రేమతో ఆదరించాడు

rAmuDu *seetani prEmatO AdariMcADu*

Rama honoured Sita with love

verb_mod (valana, X^animate, Y^activity, caused (Y,X))

రాముడు సీతవలన పరీక్ష పాస్ అయినాడు.

rAmuDu seetavalanapareeksha pass ayinADU

Rama passed the exam because of Sita

In PP attachment disambiguation a key role is played by the sortal hierarchy of TELANGANA. In verb_mod, the second argument can also be the event/state variable representing some activity/state of affairs. So verb_mod can handle the semantics of subordinate sentences also by using inter-sentential suffixes such as *tE, ani, valana,* and *i,* as the first argument and the sortal category of the main verb of the sub-ordinate sentence as the second argument. It may be noted that the procedures noun_mod and verb_mod take care of the *samarthah* condition (i) and help in assessing the semantic contribution of adjuncts.

The procedures apply and fill working in close co-operation and using the above two procedures noun_mod and verb_mod parse all adjuncts and complements. The parsing process is best understood by examining a few examples. Consider the sentence

PP_1 PP_2 PP3 V_1

where the PP_2 is the postposition phrase or a subordinate sentence TELANGANA is attempting to analyse. If PP3 has *aakaamksha* for the entity represented by PP_2 then PP_2 is treated as a complement of PP3. If PP3 has no *aakaamksha* for PP_2 then, PP_2 could either be an adjunct of PP3 or a complement/adjunct of V_1. During the call to the procedure fill, the fill procedure sees if PP_2 is a complement of V_1. If PP_2 is a

In the sentence 5.11 given above, the PP *kArulO* is first tested by one of the clauses in the apply procedure to see if *kArulO* can be a complement to *iMTiki.* The word *iMTiki* has no *aakaamksha* for any other word. Hence the related clause in apply fails to relate the PPs *kArulO* and *iMTiki.* The processing proceeds till the procedure fill is called to fill in the *aakaamksha* of the verb *veLlADu.* The procedure fill realizes, by matching sortal types, that the PP *kArulO* which indicates the mode of transport is indeed a complement of the word *veLlADu* (went) . Since all the words of the sentence are *samarthah* the sentence is accepted.

Typically only relational words like *taMdri, AdayaM, vayassu* have *aakaamksha* for another word to their left. Such words are handled by the procedure apply as already explained. In certain domains it may be necessary for nouns to have *aakaamksha* for words which impart some sort of uniqueness to them. A typical domain of this nature is VLSI design wherein parts are often mentioned to which bigger part/module they belong to. For example usages like *sarkyooT2lO TrAnsisTaru* (a transistor in circuit 2), *sarkuT31O DayODu* (a diode in circuit3) can be best accommodated by positing suitable *aakaamksha* in the words *DayODu* and *TrAnsisTaru.* In such situations it is economical to test for the possibility of complement formation before adjunct formation.

Getting back to the examples, while processing the sentence 5.12, the PP *veedhilO* (in the street) is not a complement of the word *iMTilO.* The procedure fill is called in due course and it realizes that the sortal type of *veedhi* (street) does not match the sortal type required for the mode of transport in the *aakaamksha* of the verb *veLlADu* (went). Hence the noun_mod is called to see if the PP *veedhilO* can be an adjunct to the word *iMTiki.* The procedure noun_mod indicates that *veedhilo* (in street) can indeed be an adjunct to *iMTiki* (to house) and gives the semantic content of the adjunction as "location of *illu* (house) is *veedhi* (street)".

In sentence 5.13, the PP *pareekshalO* (in the exam) is clearly not a complement nor an adjunct of *seetaki* (to Sita). It is an adjunct to the verb *rAni* (not know). There is also the possibility that it is an adjunct to the word *ayidu SlOkAlu* (five verses). But due to its proximity to the verb, and due to the possibility of adjunction with the verb, it is taken as an adjunct to the verb *rAni.*

In sentence 5.14 the word *pArkulO* is not an adjunct or complement of the word *seeta* as before. In addition, it is not an adjunct or complement of *rAni* either. Hence it is tried with the matrix sentence's PP *ayidu SlOkAlu* and verb *valliMcADu* (memorised). It is not

complement then the problem is solved. If PP_2 is not a complement of the verb V_1 then it could possibly be an adjunct of PP3. To test this, the procedure noun_mod is invoked. If PP_2 could possibly be an adjunct of PP3 then the FB of PP3 is modified accordingly. If PP_2 is not an adjunct of PP3 then it is possibly an adjunct of *V*. This is tested with the aid of the **verb_mod** procedure. If PP_2 is not an adjunct of V_1 also, then PP_2 is passed to the matrix sentence (if one exists) for further analysis, else the sentence is tagged as ungrammatical.

The operation of the above PP attachment disambiguation procedure can be illustrated using the following few examples. The underlined portions of the text are the related once either as adjuncts and head words or as complements and head words.

రాముడు ఈ కారులో ఇంటికి వెళ్ళాడు
rAmuDu ee kArulO iMTiki veLlADu --5.11
Rama went home in this car

రాముడు ఈ వీధిలో ఇంటికి వెళ్ళాడు
rAmuDu ee veedhilO iMTiki veLLADu —5.12
Rama went to a house in this street

రాముడు ఈ పరీక్షలో సీతకి రాని ఐదు శ్లోకాలు వల్లించాడు
rAmuDu ee pareekshalO seetaki rAni aidu slOkAlu valliMcADu -5.13
Rama memorized the five verses which Sita did not know[6] in this exam

రాముడు ఈ పార్కులో సీతకి రాని ఐదు శ్లోకాలు వల్లించాడు
rAmuDu ee pArkulO seetaki rAni aidu SlOkAlu valliMcADu -5.14
Rama memorized the five verses which Sita does not know in this park

రాముడు మొదటి అధ్యాయంలో సీతకి రాని ఐదు శ్లోకాలు వల్లించాడు
rAmuDu modaTi adhAyaMlO seetaki rAni aidu SlOkAlu valiMcADu —5.15
Rama memorized the five verses which Sita does not know in the first chapter

[6]To be more precise, the word *rAni* in this context of the sentences 5.13 to 5.15 would mean "does not know by heart".

an adjunct of the PP so as a last guess the possibility of adjunct formation with the verb is tried. That succeeds and location of memorisation is taken to be the park. As a contrast to the last step in the processing of 5.14, in sentence 5.15 the word *adhyAyaMlO* (in chapter) is an adjunct of the word *ayidu SlOkAlu*. Hence both 5.14 and 5.15 arc accepted.

The PP attachment regime given here may be contrasted with those found in literature for English [Fraz79, Shie83, Hirs87]. Many of these approaches are not directly applicable to analyse the PP attachment problems in Telugu. In the rest of this paragraph, an attempt is made to see what would happen if their methods were used for the analysis of Telugu[7]. The Frazier and Fodor's [Fraz79] approach, which uses the notion of "right association[8]" and the principle of "minimal attachment", cannot be used for Telugu as it fails to cover the examples 5.11, 5.13 and 5.15 correctly. The approach of Shieber [Shie83] which uses syntactic notions of resolving shift-reduce conflicts and reduce-reduce conflicts fails to analyse correctly sentences 5.12, 5.13 and 5.14. The approach advocated by Hirst [Hirs87], the principle of parsimony, requires a model of beliefs and presuppositions for PP attachment disambiguation. Even though this model is very powerful and can analyse all the above cases well, the notions of belief and presuppositions are not in a state to provide the computational frame work for PP attachment disambiguation.

5.5.5 Handling noun-noun relations

It was earlier mentioned that many fields in database relations are naturally expressed as verb complements and the relations themselves expressed as sentences. However this one to one mapping between database relations and natural language sentences is not always directly possible. Many relations are not naturally expressible using verbs only. Typically, noun-noun relations, and possessive noun-PP relations fall into this category. The interaction between non-complement PPs has been seen in the previous section 5.4.4. In section 5.4.3 handling the ambiguity concerning suffix information of possessive nouns was given. In this section handling the semantics of the possessive noun-PP relation and other noun-noun relations is highlighted. For the sake of uniformity, possessive noun-PP combination will also be called as noun-noun combinations in the rest of the section. For example, all the sentences below

[7]Their methods can not be directly used and requie some adaptation to suit the analysis of Telugu. The subsequent comparative study assumes that such an adaptation is feasible.
[8]Its equivalent for Telugu may be "left associative".

రాముడు సీత ఇంట్లో అల్లరి చేశాడు
rAmuDu seeta iMTlO allari cESADu
Rama made mischief at Sita's house

రాముడు సీత తండ్రిని చూశాడు
rAmuDu seeta taMdrini cooSADu
Rama saw Sita's father

రాముడు సీత తండ్రి జనకుడిని చూశాడు
rAmuDu seeta tamDrijanakuDini cooSADu
Rama saw Sita's father Janaka

రాముడు బెంజికారు కొన్నాడు
rAmuDu beMji kAru konnADU
Rama bought a Benz car

రాముడు బేతంచెర్ల రాళ్లతో ఇల్లు కట్టాడు
rAmuDu bEtaMcerla rALlatO illu kaTTADu
Rama built a house with stones from Betamcharla

రాముడు లండన్ విమానం ఎక్కాడు
rAmuDU laMDan vimAnAM ekkADu
Rama boarded the flight to London

involve noun-noun combinations. Even though these sentences can be expressed without noun-noun combinations using some extra verbs the resulting sentences are not at all natural. Hence they must be handled as they are. The case of handling possessive nouns is comparatively easier than the general case of noun-noun combinations. The variety of relationships that can be possibly expressed using possessive case are not many. There are two ways in which possessive nouns are used. When relational words like *taMDri* (father), *AdAyaM* (income), or *vayassu* (age) are used they subcategorize for a word in possessive case. In such a situation, a possessive case PP is just like any other complement PP. Hence its semantic contribution is straight forwardly indicated in the *aakaamksha* of the relational word. The FB for the word *taMDri* is

[aak:l: [var:X, isa:humanl_],

var:Y,

yog:[isa:human, pos:n, ref: [n:___],agr: [[m,s,3J, [_,cnt]]],
sem:[reln:taMDRi , id:Y, father_of:X]]

The FB clearly indicates the role of the possessive PP. The *aakaamksha* given above is in a slightly different format than the one for verbs. This was done so that the semantic processing of possessive could be optimised by not requiring to use the procedure fill which is used for filling verb complements. In addition, in the case of matching a complement to the *aakaamksha* of a verb, the FB of the complement is embedded into the semantics of the verb through a case relationship, such as experiencer, agent, giver, taker, taken etc. But in the case of nouns it is not customary to use such case relations, and hence the semantics of the possessive PP should be explicitly represented. In addition, from point of view of assigning scopes to quantifiers representing possessive PP's, semantics in an unembedded fashion has certain advantages (Chapter 7) of uniformity. Another compelling reason is that certain nouns which idicate units of measures like *mArkulu* (marks), *samvatsarAlu* (years) when used as complements to some verbs, do not lend their semantic content to the verb but for their quantifiers. Based on such considerations, the complement filling in case of nouns has been implemented some what differently from the complement filling of verbs and hence requires a different kind of treatment. The FB of the word sequence *rAmuDi taMDri* would be the FB of the PP *taMDri* with the additional feature called "and" with the value equal to FB of *rAmuDu.* When the logical form for this word sequence is generated subsequently, the feature "and" is treated as a logical "and" operator and it also participates in quantifier scoping considerations. Thus thc FB of *rAmuDi taMDri* is simply

[aak:l: [*...same as the contents of the* "and" *feature below...*],
var:Y, post:none,
yog:[isa:human, pos:n, ref: [n:_,_], agr: [[m,s,3], [_,cnt]]],
and: [aak:n: [],var:X, isa:human, pos:n,
agr: [[m,s,3], [none,_]], sem: [reln:name, id:X, string:rAma]l,
sem: [reln:taMDri, id:Y, father_of:X]]

The second kind of possessive occurs in word sequences like *seeta illu* (Sita's house), and *rAmuDi yoonivarsiTi* (Rama's university) where the exact relationship between *seeta* and *illu,* and *rAmuDu* and *yoonivarsiTi* are not clear. In such possessive cases, words like *illu* and *yoonivarsiTi* have no *aakaamksha* for the words *seeta* and *rAmuDu.* A proper treatment of such possessive nouns and other noun-combinations requires world knowledge [Gros87], domain knowledge and at times discourse knowledge also. For

example in the first sentence above, *seeta illu* (Sita's house) could mean (i) the house owned by Sita, or (ii) the house sita presently lives in, or possibly (iii) the house Sita is going to get built. Similarly the noun-noun relationship in *beMj kAru* (Benz car) could mean (i) a car manufactured by the Benz company or (ii) a car hired from the Benz hiring company or (iii) a car whose model name is Benz. Thus the relationships expressed by noun-noun combinations are too idiosyncratic to be captured by a few rules. In TELANGANA a few well known relationships are stored under the procedure name noun-noun. The procedure noun_noun has three arguments. The first and second are the sortal types of the two nouns involved in a noun-noun combination. The third gives the semantic relationship. For example,

noun_noun (X^manufacturer, Y^entity, manufactured _by (Y,X)).

noun_noun (X^place,Y^entity,supplied_from (Y,X)).

When the procedure, apply, comes across two adjacent PPs and the first PP has the suffix "a", or "dflt_i", or "i", or no suffix at all, then a noun-noun combination is suspected. The possible relationship accruing from the noun-noun combination is computed using the procedure noun_noun. The computation is done so as to be backtrackable in case the suffixes were guessed wrongly (see Section 5.5.3).

5.5.6 Handling adjectives and quantifiers

The lexical entries for adjectives and quantifiers were described elaborately in Chapter 3. The mechanisms for achieving the agreement between adjectives and nouns, quantifiers and nouns, and between multiple quantifiers and nouns are described in this section.

The procedure apply has three clauses dedicated to analyse the adjective, determiner and quantifier interactions and to build the FB of the result of the interaction. The procedure code for adjective analysis is as follows.

```
apply ([[aak:r: [],var:XITl]IT2]-RES, [aak:n:A,var:XIT3]) :-
        mem (pos:adj,Tl), (mem (pos:n,T3);mem (pos:pro,T3)),
        (adjagree (T1,T3),!; error (' adj and noun do not agree'),nl,fail),
        apply (T2-RES, [aak:n:A,var:X,vish:THT3]).
```

The procedure adjagree, in the above clause, verifies the syntactic conformance of case, gender and person between the adjective and the noun. It also verifies semantic conformance in terms of checking up if the adjective is appropriate to the sortal type of

the noun. This is done by seeing if the quality the adjective qualifies is an attribute of the noun or not. For example in the word sequence *tella poolu* (white flowers) the adjective *tella* (white) indicates the colour attribute of the noun *poolu.* If *poolu* (flowers) do not have an attribute, colour, then this is an ungrammatical word sequence. The procedure adjagree does this kind of conformance checking. For example the FB[9] for the word *tella* is

[aak:r: [var:YI_]_, var:X,pos:adj,
isa:ramgu, agr: [_, [none,deg]],
sem:raMgu(X,Y)]

The FB of *poolu* is

[aak:n: [], var:X,pos:n,post:none,
isa:flora, attr: [ramgu, odor, size, weight ...*etc*],
agr:[[n,pl,3], [none,J],
sem:puvvu(X)]

The feature "attr" of *poolu* has a list value that contains the element "ramgu", and the sortal type of *tella* is "ramgu", hence *tella* can be a suitable adjective for *poolu.* The sortal hierarchy is used to get the attributes for different nouns. The lex_sem procedure explained in section 5.4.1 takes care of inserting the attributes of objects into their FBs during run time.

Determiners are handled in a fashion similar to adjectives, but using the procedure detagree. Determiner-noun agreement is purely syntactic in nature and can be affected by unifying the "agr" features of the determiner and the noun.

The procedure quantagree is invoked to test the conformity amongst quantifiers and nouns. Handling quantifiers requires some tricky case analysis as indicated in Chapter 3 because, the semantic content of quantifiers is sensitive to the entity they quantifiy. For example *cAlA poolu* means many flowers and *cAla baMgAraM* means much gold, and *cAlA maMci* means very good. To handle this mutiple meanings of quantifiers like *cAlA,* the "agr" and "sem" features of quantifiers are made list valued. The list values of "agr" are closely related to the list values of "sem". The procedure quantagree traverses these lists to select the right sense of the quantifier by using the values in the "agr" feature list.

[9]The 'sem' feature is shown in predicate form for pedagogical purposes.

For example, the noun *lekkalu* (arithematic problems) takes a count quantifier, like *padi* (ten) as in *padi lekkalu* (10 mathematics problems), and hence *cAlA lekkalu* means many problems rather than much problems. As given in Chapter 3, the FB for *cAlA* is

[var:X,
yog: [pos:quant,
agr: ([[_,pl,_], [indef,cnt]]/ [[_,s,_], [indef,mas]]/ [_, [indef,deg]])],
sem: (number (X,caalaa)/quantity (X,caalaa)/degree (X,caalaa))]

The procedure for quantagree is called by the apply procedure in three different contexts. The first one is when a quantifier and a noun/ adverb as in *cAlA lekkalu, kAsta melligA, cAlA tvaragA,* are to be related. In this case, quantagree is called with its first argument as "qn", indicating a quant-noun or a quant-adverb combination is being analysed. The second one is when a quantifier and another quantifier as in *cAlA maMci, emtO konni* are being related. In this case, quantagree is called with its first argument set to "qq". The third one is in comparative sentences like

రాముడు 10 కంటే ఎక్కువ లెక్కలు చేయలేడు
rAmuDu 10 kaMTE ekkuva lekkalu cEyalEDu
Rama can not solve more than 10 problems

wherein *10 kaMTE* (than ten) is a modifier to the quantifier *ekkuva* (more). The third case will be treated in the next Chapter. The procedure quantagree has four arguments. The first argument as mentioned above indicates the type of the construct being analysed. The second argument is the FB of the quantifier with list valued "agr" and "sem". The third argument is the "agr" feature value of the entity (noun/ adverb/ another quantifier). being modified by this quantifier. The fourth argument gives the disambiguated FB of the quantifier that is appropriate in the given context. Strictly speaking, quantagree procedure is not necessary, and can be subsumed by the adjective-noun/adverb case. But that would need backtracking to get the appropriate sense of the quantifier. To avoid such costly backtracking and for the sake of effeciency, this kind of list structure and list traversal have been used. The quantagree procedure is defined as follows,

```
quantagree (Type,QFB,AGR,FBOut) :- rem ([agr:AGRList,sem:SEMList],QFB,FBl),
                                   setsem (Type,AGRList,SEMList,AGR,FB 1 ,FBOut).
```

setsem (qn, [G, [D,Y]],S, [G, [none,Y]],T, [agr: [G, [D,Y]],sem:S|T]).

setsem (qq, [G, [indef,deg],S, [G, [indef,Z]],T, [agr: [G, [indef,deg]],sem:S|T]).

setsem (kaMTE, [G, [D,Z]],S, [G, [indef,Z]],T, [agr: [G, [D,Z]],sem:S|T]).

setsem (Type,G1/G2,S1/S2,AGR,FBin,FBOut) :-

setsem (Type,G1,S1,AGR,FBin,FBOut).

setsem (Type,G1/G2,S1/S2,AGR,FBin,FBOut) :-

setsem (Type,G2,S2,AGR,FBin,FBOut).

5.5.7 Handling stative sentences

Stative sentences in Telugu usually contain the verb *un* (exists) or no verb at all. The case where the verb *un* is present is straight forward and can be analysed using the procedures mentioned hiterto. When no verb is present, as in,

రాముడు మంచిబాలుడు
rAmuDu maMcibAluDu
Rama is a good boy
* *maMcibAluDu rAmuDu*

సీత పొట్టి
seeta poTTi
Sita is short
* *poTTiseeta* (can not be a senetence, but can be a PP)

కుక్కలు జంతువులు
kukkalu jaMtuvulu
Dogs are animals
**jaMtuvulu kukkalu*

సీత తండ్రి జనకుడు
seeta taMDrijanakuDu
Sita's father is **Janaka**

జనకుడు సీత తండ్రి
janakuDu seeta taMDri
Janaka is Sita's father

సీత వయస్సు 10 సంవత్సరాలు
seeta vayassu 10 samvatsarAlu
Sita's age is 10 years
10 samvatsarAlu seeta vayassu

వాడు రాముడు
vADu rAmuDu
He is Rama

రాముడు వాడు
rAmuDu vADu
Rama is that person

the verb *ayiun* (happens to be) with suitable agreement features like number, gender, person is assumed to be present. The *aakaamksha* of the verb should allow for all the above sentences and disallow the starred variants of some of them which happen to be syntactically invalid. The problem is in correctly positing the *aakaamksha* of *ayiun* to accommodate only the valid variants. This is not a difficult problem if multiple senses of *ayiun* are accepted and their suitable *aakaamkshas* are appropriately set.

5.6 Conclusion

The author wasinspired by the rich tradition available in the Sanskrit grammar for explaining word and sentence formation. The treatment of Telugu syntax presented in this chapter draws its support from the ideas found in Panini's Sanskrit grammar; particularly *aakaamksha, yogyata, sannidhi, and samarthah.* These ideas were couched in non-formal terms as seen from the present day grammatical formalisms. Hence these ideas were formalized, the first three in the third chapter and the last one in this chapter. From this formalisation a novel and a computationally viable method for parsing Telugu sentences has been developed. This method of parsing is called 'Functional Application

Parsing'. This parsing technique has been used as the basis of the syntactic/semantic analysis module of TELANGANA. This chapter justifies and describes Functional Application Parsing and then details parsing and building the semantic representations for a variety of simple Telugu sentence structures such as, sentences with no verbs (copula) sentences, sentences consisting of one verb and its complements, sentences with noun modifiers like adjectives, determiners and quantifiers, and sentences with verb modifiers like adverbs, and adjuncts. In doing so, the operation of the syntactic/semantic analysis module of TELANGANA is also detailed.

Handling more complex setences consisting of relative clauses, comparatives sentences (including clausal comparatives, adjectival and adverbial comparatives, superlatives), some amount of intra-sentential anaphora and limited conjunctions is dealt in the next chapter. TELANGANA parses all these varied types of sentences in the uniform and a simple way layed out in this chapter.

Chapter 6

Syntactic and Semantic Analysis of Complex Sentences

6.1 Introduction

The foundations of functional application parsing were laid in the previous Chapter. In addition, syntactic and semantic analysis of simple sentences was also presented. To deal with more complex sentence types, the principles underlying their analysis need to be incorporated into the parser. This chapter presents those principles and the manner in which they are incorporated into TELANGANA.

Complex sentences are sentences with multiple verbs. Sentences containing relative clauses, subordinate clauses, coordinate clauses, and comparative clauses are examples of complex sentences. Analysing complex sentences leads to several interesting questions like, What kind of constraints exist in Telugu on the formation of relative clauses?, How are gaps in relative clauses matched to their fillers?, Can the SLASH feature of GPSG be used for this purpose?, How can subordinate clauses be related to their matrix verbs?, and Can comparative sentences be handled using *aakaamksha* information only? Questions of this nature will also be answered in this chapter. Details of parsing sentences with relative clauses, multiple verbs, comparative clauses and pronouns will be presented in the subsequent sections.

6.2 Handling Relative Clauses

The purpose of a relative clause is to restrict the set of referents of the noun[1] which the relative clause modifies. The process of relativization[2], in Telugu, involves removing a noun phrase together with its suffixes from the sentence, and making the rest of the sentence (the

[1] Called antecedent noun
[2] The process of converting a sentence into a relative clause

relative clause), a modifier to that noun by suitably changing the suffixes of the main verb of the sentence. The suffixes indicating the number, gender, and person of the main verb of the relativized sentence the verb are deleted and *ina, E, ani* or *O* is added to the root form of the verb. The noun that is taken out of the sentence loses all its old suffixes, that indicated its deep case relationship to the verb of the relative clause, and gets a new set of suffixes as constrained by the *aakaamksha* of the matrix verb. In Telugu, a relative clause is placed just before the antecedent noun phrase it modifies. For example, the sentence

రాముడు నిన్న పార్కుకి వెళ్లాడు
rAmuDu ninna pArkuki veLlADu
Rama went to a park yesterday

can be relativised into the following clause,

రాముడు నిన్న వెళ్లిన పార్కు
rAmuDu ninna veLlina pArku
The park to which Rama went yesterday

The suffix *ki,* which indicates that *pArku* (park) is the destination of the act *veLl* (go), is removed at the time of relativization. For example, the above relative clause can be used as shown below,

రాముడు నిన్న వెళ్లిన పార్కులో చాలా గులాబీ చెట్లు ఉన్నాయి
rAmuDu ninna veLlina pArkulO cAlA gulAbi ceTlu unnAyi
There are many rose plants in the park which Rama went to yesterday

Telugu relative clauses can be classified into three classes[3], based on the relationship between the clause, its antecedent and the way the antecedent is syntactically related to the gap. Specifically, these three classes are characterized by,

1. Antecedent fills a gap in the relative clause,
2. Antecedent is semantically null, and
3. Antecedent is expressed idiosyncratically

[3]This classification is an invention of the author for the purpose of computational anlaysis. This classification is not found in Telugu grammar books or liguistic literature.

The following sections present analyses of the above types of relative clause constructs, and detail how they are dealt with in TELANGANA.

6.2.1 Antecedent Fills a Gap in the Relative Clause

In the above type of relative clauses, a complement PP or a certain class of adjuncts in the relative clause would be missing. The antecedent of the relative clause fills in the role of the missing PP. Typical examples of this type of relative clauses are

రాముడు సీతకి ఇచ్చిన పూలు చాలా బావున్నాయి
rAmuDu seetaki iccina poolu cAlA bAvunnAyi --6.1
The Flowers which Rama gave to Sita are very beautiful

రాముడు అడిగిన ప్రశ్నకి ఎవ్వరూ సమాధానం చెప్పలేదు
rAmuDu aDigina praSnaki evvaroosamAdhAnaM ceppalEdu --6.2
Nobody replied to the question which Rama asked

ఇంటికి వెళ్లిన సీత రాముడిని కలిసింది
iMTiki velina seeta rAmuDini kalisiMdi —6.3
Sita who went home met Rama

రాముడు దాగుకొన్న చోటు అందరికీ తెలుసు
rAmuDu dAkkonna cOTu amdarikee telusu --6.4
The place where Rama was hidden is known to everybody

గాంధి పుట్టిన ఊరిపేరు ఏమిటి?
gAMdhi puTTina ooripEru EmiTi? -6.5
What is the name of the city where Gandhi was bom?

రాముడు వెళ్ళిన ఊరిపేరేమిటి?
rAmuDu veLlina ooripErEmiTi? -6.6
What is the name of the city to which Rama went?

రాముడు వెళ్ళని ఊరు లేదు
rAmuDu veLlani ooru lEdu --6.7
There is no city to which Rama did not go (visit)

టీవీలో రామాయణం చూపించిన ఆరోజు సీత ఇంట్లో లేదు

TVlO rAmAyaNaM coopiMcinarOju seeta iMTlO lEdu -6.8

Sita was not at home when Ramayana was shown on TV

రాముడు క్రికెట్టాడే స్టేడియం నాకు తెలుసు

rAmuDu krikeTTADE sTEDiyaM nAku telusu --6.9

I know the stadium where Rama plays cricket

In the first sentence above, the antecedent noun, *poolu* (flowers), fills the role of the missing object noun in the relative clause. The object must be expressed in accusative case using the suffix *ni.* The antecedent, on the other hand, is in nominative case, and is used as a subject in the matrix sentence. In the second sentence the word *praSna* fills the role of an object noun in the relative clause, but unlike *poolu* of the first sentence it is used in dative case in the matrix sentence. In the third sentence, the antecedent noun, Sita, is used in nominative case as a subject in both the relative clause and the matrix sentence. In the third sentence *cOtu* (place) fills the locative case of the relative clause which must be expressed with the suffix *lO.* In the sixth sentence, the noun *ooru* (city) fills the role of destination in the relative clause. Normally destination is expressed using the suffix *ki.* The word *ooru* is used in possessive case in the matrix sentence. Thus, in Telugu relative clauses, the case of the missing element is in no way related to the case of the antecedent. This means that the entire complement PP is lost. Its suffix/case is also not available for subsequent reference. Even in the case of adjuncts, a similar thing happens. For example, in the previous two example sentences, where the antecedent contributes an adjunct to the relative clause, there is no clue as to the nature of the adjunct; adjunct of manner, time or location.

The deletion of all the suffixes in this way, imparts to Telugu a kind of uniqueness not present in English. In English some clues as to the possible relation of the antecedent to the main verb of the relative clause are left behind when sentences are relativized. For example, in the following sentences

This is **the** topic on which I gave a lecture _ yesterday[4].

The person whom you gave this book to __ is not here.

[4] As customary in linguitis literature, the symbol "__" indicates a gap.

the antecedent preposition phrase "on which" in the first sentence, and the word sequence consisting of a gap, "to _", in the second sentence have the necessary preposition markers to indicate the case role of the antecedent When antecedents are adjuncts, suitable relative pronouns like where, when, which are provided as clues to the nature of the adjuncts. Thus, in English, a few clues always exist to enable one to relate an antecedent to the gap in the relative clause. However in Telugu no such syntactic clues are left behind. The gap-filler relations in Telugu, on the other hand, can only be analysed using semantic clues. Syntactic clues tend to be of little or no help. This makes analysing Telugu relative clauses more difficult and interesting.

Telugu linguists do not know the grammatical restrictions governing the formation of relative clauses in Telugu [Kris85, p 248]. Certain case relations between the antecedent and the relative clause can not be implied by the simple syntax of relativization given before. From the sentences

రాముడు సీతతో వెళ్ళాడు
rAmuDu seetatO veLlADu
Rama went with Sita

రాముడు సీతచే పూలు కోయించాడు
rAmuDu seetacE poolu kOyiMcADu
Rama made Sita pluck flowers

one cannot derive the relative clause

*రాముడు వెళ్ళిన సీత
**rAmuDu veLlina seeta*
*రాముడు పూలు కోయించిన సీత
***rAmuDu poolu kOyiMcina seeta*

The lack of well articulated constraints on relative clause formation makes it difficult to analyse good sentences and recognise illformed sentences. In TELANGANA the working hypothesis taken is that complements indicating an independent agent, an object, an instrument, or a predicative argument and adjuncts indicating a place or a generic time are the only possible candidates for right extra-position during relative clause formation.

The above hypothesis, in conjunction with the observation that the sortal type of the antecedent helps in choosing the correct semantic relationship, leads to the following heuristic rules for analysing relative clauses.

(A) If the sortal type of the antecedent is human or propositional[5], then it can satisfy only an agent or an object related gap in the *aakaamksha* of the main verb of the relative clause.

(B) If the sortal type of the antecedent is human or propositional, then the antecedent can not be an adjunct.

(C) If the sortal type of the antecedent is physical_object, then it can satisfy any gap in the *aakaamksha* of the main verb of the relative clause, subject to sortal constraints. If there are no gaps in the *aakaamksha,* then the antecedent can be an adjunct.

To test the possibility of an antecedent being an adjunct one needs postposition information. However, such information would be missing due to the relative clause formation rules of Telugu. So suffixes like *tO, lO, meeda, paina* are tried out one after the other in TELANGANA. If the order in which the suffixes are tried is according to the statistical co-occurrence of the word and the suffixes, better performance can be obtained. This co-occurrence information can be gathered from textual corpora. The frequency information can be stored as part of lexical information associated with every noun. In fact this kind of information also helps in morphological analysis, even though TELANGANA's morphological analyser, MA, does not use this information at present. Such a strategy, no doubt, would improve the speed of TELANGANA. However, performance being not the key issue in the design of TELANGANA, such enhancements have been left to future implementation.

The above heuristics are implemented as part of the fill procedure and the apply procedure described in chapter 5. A set of additional clauses in the definition of the apply and fill procedure, handle the relative clause structures of this type where the antecedent fills a gap in the relative clause. While performing morphological analysis, if a verbal root, V, is found to be present in the input along with one of the suffixes *ina, E,* or *ani*, then the morphological analyser outputs the string "V,suffix" instead of outputting the normal string sequence "V+suffix". This helps the parser in applying the verb, V, to its *yogyata* words (complements) before applying the suffix to the verb. If the "V+suffix" form was used, suffix would have been applied to the verb, V, right in the lex_sem procedure, leading to problems that will become evident shortly. The order of application of suffix and verb is

[5]Objects like books, news papers, and stories belong to this category.

important to the proper processing of relative clauses. When the verb is applied to the words seen so far, the **fill** procedure, described in chapter 5, forms a FB wherein all the *aakaamksha* of the verb, V, is satisfied except for the missing complements. The *ina, E* and *ani* suffixes takes such an FB as their argument, and checks if the *aakaamksha* of the FB is well formed for relativization purposes. The unsatisfied elements in the residual *aakaamksha* essentially have a bearing on the acceptability of the relative clause formation as follows.

(i) If the number of unsatisfied complements in the *aakaamksha* is greater than two, the relative clause is tagged as **ungrammatical**.

(ii) If that number is two, then the list elements of the "for" feature (cf. Section 6.3) are checked to see if any one of them is unfilled. If more than one is unfilled then again an error is indicated.

(iii) If the number is zero, then reject the relative clause formation if the antecedent is of sortal type human or propositional, and indicate a failure, otherwise accept it.

(iv) If the number is one, then accept the FB argument as valid.

If the FB is found to be acceptable, then it is fit to be an argument of the relative clause formation suffixes *ina, E* and *ani*. The suffix *ina* acts as a function over the FB of the main verb of the relative clause, and converts the verb's residual *aakaamksha* from aak:l:A to aak:r:A'. This causes a new FB to be associated with the verb. It may be observed that owing to the presence of the string "r" in the new *aakaamksha*, the relative clause acquires the ability to seek a complement to its right in the **sentence**. The elements of A' determined got by changing the suffixes of the complement in A to "relc" from whatever they were earlier. The feature value "relc" is used to signal to the fill procedure to ignore suffix incompatibility, when the *yogyata* of complement is matched to *aakaamksha* of the embedded verb. This enables the **fill** procedure to accept a complement or an adjunct based on its sortal category alone. Hence, the effect of applying the suffix *ina* or *E* or *ani* to a verb, is to modify the *aakaamksha* of the verb as above. In effect, the FB of the words processed upto the relative clause suffix, has *aakaamksha* for a noun that occurs to its right. That is, the relative clause behaves as if it were an adjective. However, as the antecedent could be a missing complement or an adjunct, an additional clause is defined in the apply procedure. This calls the **fill** procedure to handle both these cases. This way of processing relative clauses, as if they had *aakaamksha* for an element to their right, obviates the necessity of using the SLASH features in the analysis of Telugu relative clauses. This also

makes relative clauses similar to adjectives, the way linguists have been treating them [Kris85, p 237]. This is one of the major deviations of TELANGANA from GPSG [Gazd85], HPSG [Poll87], and JPSG [Gunj87]. This is achieved due to viewing suffixes as functions over FBs of the root words.

The third case, (iii), in the analysis of the well formedness of the relative clause requires some explanation. The failure to accept the case where there is no unsatisfied element in the *aakaamksha*, and the antecedent is of sortal type human or propositional, leads to backtracking. This leads to rejection of the last complement that was accepted while processing applying the verb to the words on its left in the sentence. This rejection leads to the creation of a gap in the *aakaamksha* of the main verb of the relative clause. The antecedent fills this gap subsequently. The initially accepted, but later rejected complement, in effect, is passed on to the matrix verb. The following example clarifies the situation. In the following sentence,

రాముడు పరీక్షలో కాపీ కొట్టిన విద్యార్థిని పట్టుకున్నాడు
rAmuDu pareekshalO kaapee koTTina vidyArthini paTTukunnADu.
Rama caught the student who copied in the examination.

while analysing the antecedent, *vidyArthini*, the *aakaamksha* of the main verb, *koTT,* of the relative clause would be totally satisfied. The subject role is the one that gets filled last. It gets filled initially because the noun, *rAmuDu,* has the appropriate *yogyata* with respect to the *aakaamksha* the verb, *koTT.* However, the acceptance of *rAmuDu* as the subject of *koTT,* renders further analysis of the relative clause impossible. This is because the word *vidyArthini* does not have a place to **fill** in the *aakaamksha* of the verb *koTT,* even though it has the right *yogyata to* satisfy the *aakaamksha* of the verb *koTT.* The above rule handles this situation. It flags the relative clause as not being well formed. Hence, backtracking takes place and the subject role in the *aakaamksha* is left empty. This is subsequently filled by the antecedent. It may be noted that this problem can not be overcome by just changing the eager matching of complements in the **fill** procedure to lazy matching. The following example shows where eager matching is profitable.

రాముడు పరీక్షలో కాపీ కొట్టిన రోజు పట్టుబడ్డాడు
rAmuDu pareekshalO kaapee koTTina rOju paTTubaDDADu
Rama was caught the day he copied in the examination

The case of the antecedent filling a gap in the relative clause is the most frequently occurring one in real life texts. However, there are many situations, which warrant the use of the second type of relative clauses. In these clauses, the antecedent does not contribute anything to the semantics of relative clause.

6.2.2 Semantically Null Antecedent

In some relative clause formations, the clause determines the semantic content of the antecedent and adds substance to it. Without the relative clause, the semantic content of the antecedent would be null. For example, in the following English sentence,

Nobody noticed the fact that Rama went home

the relative clause specifies the content of the noun "fact". Such relative clauses in Telugu are formed using the same syntax as the previous type of relative clauses. For example, in the sentences given below

రాముడు పరీక్షలో తప్పిన సంగతి ఎవరికీ తెలియదు
rAmuDu pareekshalO tappina saMgati evvarikee teliyadu
The fact that Rama failed in the exam is not known to anybody

రాముడు ప్రధానమంత్రి అయ్యాడనివార్త క్షణంలో దేశమంతా పాకింది
rAmuDu pradhAnamaMtri ayyADaninavaArta kshanaMlO dEsaMaMtA pAkiMdi
The news that Rama has become the prime minister spread in a minute all over the country

the words *saMgati* (fact) and *vArta* (news) are neither complements or adjuncts to the embedded relative clauses. On the contrary, the relative clause fills in the content of the antecedents. Nouns which can be antecedents of this type are few in number. However their use in the sentence must be tested before accepting them as semantically null, because these nouns can often be genuine complements or adjuncts also. For example, in the following sentences the words *saMgati* (fact) and *vArta* (news) can be complements or adjuncts as follows.

రాముడు సీతకి చెప్పినసంగతులు రాధకి కూడా చెప్పాడు
rAmuDu seetaki ceppinasaMgatulu rAdhaki kooDA ceppADu
Rama told the facts to Radha which he told to Sita

రాముడు చదివిన వార్త సీత చదవ లేదు
rAmuDu cadivina vArta se eta cadava lEdu
Sita did not read the news that Rama read

To the best knowledge of the author, Telugu linguists have not published any constraints on formation of relative clauses of this type. Hence TELANGANA makes use of a simple heuristic to handle such relative clauses. The key observation to be made in the case of these types of relative clauses is that there are a very few number of nouns which can act as antecedents to them. In addition, the main verb of the relative clause in such cases is one of words like *an* (say), *cepp* (tell), and *caduv* (read/study) which have a sortal type of mental. When the parser of TELANGANA encounter a word like *saMgati, mATa,* or *vishyaM* as an antecedent to a relative clause, it tests the sortal type ("isa" feature value) of the main verb of the relative clause. If the sortal type is mental, then the sthiti (state)/samdarbham (event) variable of the relative clause is unified with the "var" value of the antecedent. This indicates that the antecedent and the relative clause refer to the same state/event. In the domain of database access relative clauses of this category seem to occur very infrequently.

6.2.3 Antecedent Expressed Idiosyncratically

It was stated in section 6.2.1 that certain kinds of relative clauses can not be constructed by using the simple relativization mechanism given earlier in that section. The only complements which surely admit relativization using the direct simple suffixes *ina* and *E* are the subject and the object. Other complements and adjuncts occasionally allow direct relativization. The situations under which the simple relative clause formation works is not understood well [Kris85, p 248]. It appears that, when the antecedent has no specific reference or refers to an unknown entity or if it is meant to fill certain kind of complement PP gaps, an idiosyncratic syntax is used in Telugu to from relative clauses. For example for each of the following starred sentences with relative clauses, the ordinary relative clause formation mechanism leads to incorrect sentence formation. By using the idiosyncratic syntax as shown in the corresponding valid sentences, relative clauses can be formed.

*రాముడు పరీక్ష రాసిన జ్వరం చాలా బాధాకరమైనది
* *rAmuDu pareeksha rASina jvaraM cAlA bAdhAkaramainadi*

రాముడు ఏజ్వరంతో అయితే పరీక్ష రాశాడో అజ్వరం చాలా బాధాకరమైనది
rAmuDu EjvaraMtO aitE pareeksha rASADO ajvaraM cAlA bAdhAkaramainadi

The fever with which rama wrote the exam is very painful

?రాముడు చదువు చెప్పిన వాళ్ళ వృద్ధిలోకి వచ్చారు

• *rAmuDu caduvu ceppina vALlu vruddhilOkivaccAru*

రాముడు ఎవరికైతే చదువుచెప్పాడో వాళ్ళ వ ృద్ధిలోకి వచ్చారు

rAmuDu evarikaitE caDuvuceppADO vALlu vrudhilOki vaccAru

Those Whom Rama taught prospered well

*రాముడు ఈప్రయోగం కలిసి చేసినవాడికి కూడా బహుమతి ఇచ్చారు

rAmuDu eeprayOgaM kalisi cESinavADikikooDA bahumati iccAru

రాముడు ఈప్రయోగం ఎవర్తోనైతే కలిసి చేశాడో వాడికి కూడ బహుమతి ఇచ్చారు

rAmuDu eeprayOgaM evartonaitE kalisicESADO vADiki kooDdbahumati iccAru

The person together with whom Rama did this experiment also got an award

All of the above sentences have one commonality, which has been highlighted with underlining. These sentences have the interrogative pronoun *evaru, evaDu* or *Edi* or a noun with the interrogative prefix *E,* in conjunction with the suffix *aitE,* an antecedent that is a matching pronoun or a noun, and the suffix *O* on the main verb of the relative clause. All these give strong clues to the parser in establishing a correspondence between the complements and the antecedents. At present, the analysis of these types of relative clauses in TELANGANA is simplistic. When the lex_sem procedure, described in chapter 5, encounters a word, W, with the prefix, £, and the suffix *aitE,* it does a look ahead for a word in the input sentence that matches the word W. If it finds such a word, M, it adds to that word the indicator, ind(V). The variable, V in ind(V), is the variable representing the word W. The indicator indicates to the parser that the "var" feature value of the matching word, M, must be identical to the "var" feature value of the word, W. When the parser subsequently encounters the indicator ind(V) with the word M, it unifies V with the value of the "var" feature of the FB representing M. This ensures that both W and M refer to the same entity when the FB of the whole sentence is computed.

6.3 Handling Subordinate Constructions

Subordinate clauses in Telugu are formed by adding suffixes like *ι, tE, 0, oo, akuMDA aka, inaa,* **and** *EmO* to the main verb of a sentence, and then using such a clause in conjunction with **a** main sentence. Typical examples of Telugu subordinate constructs using these suffixes are

రాముడు ఇంటికి వెళ్ళి బట్టలు మార్చుకొన్నాడు

rAmuDu iMTiki veLli baTTalu maarcukonnADu --6.10

Rama went home and changed (his) clothes

రాముడు ట్రైనులో వెళ్ళూ పేపరు చదువుతాడు

rAmuDu TrainulO veLltoo pEparu caduvutADu --6.11

Rama reads paper while going in the train

రాముడు అన్నం తినకుండా స్కూలికి పరిగెత్తాడు

rAmuDu annaM tinakuMDA skooliki parigettADu —6.12

Rama ran to school without eating food

రాముడు చదవకపోతే పరీక్షలలో తప్పుతాడు

rAmuDu cadavakapOtE pareekshalalO tapputADu —6.13

If Rama does not study he will fail in the examinations

రాముడికి మంచి మార్కులు వస్తే మనందరికి పార్టీ ఇస్తాడు

rAmuDiki maMci mArkulu vastE manaMdariki pArTeeistADu —6.14

If Rama gets good marks, he will give us all a party

రాముడు పరిగెత్తినా నేను పరిగెత్తను

rAmuDu parigettinA nEnu parigettanu —6.15

Even if Rama runs, I will not run

మనం పరిగెత్తకపోబట్టి వానలో తడిశాము

manaM parigettakapObaTTivAnalO tadiSAmu —6.16

We got wet because we did not run in the rain

In the above sentences, the underlined words do not appear in the corresponding Telugu sentences. Those words represent gaps in the corresponding Telugu matrix sentences. The case of the gap is not always same as the case of the filler. Subordinate sentence constructions in Telugu have one striking difference with respect to English. The subject and to a lesser extent certain other complements of the subordinate sentence can fill the gaps in the matrix sentence. That is an element in the embedded sentence fills a gap in the matrix sentence! This situation never occurs in English. This entails that the simple unidirectional SLASH mechanism and the Foot Feature Principles of HPSG and GPSG proposed for

English are inappropriate for handling Telugu gap-filler structures which are bi directional. In addition in Telugu multiple gaps can be present in the embedded or the matrix sentences. This makes it necessary to sortally match gaps and fillers before they are bound together.

The method used in TELANGANA is to make use of a new list valued feature called "for" (forward), which indicates the complements that can become gaps. This information is present in the lexical entry of every verb. The elements on the "for" list are the complements which can be missing in the clause or the subordinate/coordinate sentence. If they are missing in the embedded clause then they can be bound to some eligible complements in the matrix sentence. During this binding process certain suffix changes are allowed to take care of a possible suffix mismatch between complements of the embedded clause and matrix sentence. If complements are not missing in the embedded clause, but are missing in the matrix sentence then the complements in the matrix sentence are matched with the complements in embedded clause with appropriate suffix changes. For this purpose the complements in the embedded clause should be forwarded to the matrix sentence and should be available for reference while processing the matrix sentence. If the corresponding complements of the matrix clause are indeed present and are not gaps, then they need not be matched with the forwarded complements of the embedded clause. In this situation, the forwarded complements are ignored. In other words complements in the embedded clause are treated as default complements available for use only if they are needed at the matrix sentence level. This default value like behaviour is what makes the HPSG and GPSG style SLASH feature based approach unsuitable for gaps and fillers in Telugu. In HPSG and GPSG, information about the fillers in the matrix sentence percolate into the embedded sentence through the SLASH category. Fillers can be in the matrix sentence only, but not in the embedded sentence due to the fact that English allows only this. Hence, information flows is in one direction, from matrix sentence to the embedded sentence. Whereas in Telugu, as shown in the above analysis, information needs to flow in both directions. Thus the use of "default values" and the two way percolation of information pertaining to fillers and gaps, makes the nature of information flow across FBs in Telugu fundamentally different from HPSG, GPSG, JPSG [Gunj87] and CG [Vanb87]. Functional application parsing enables this information flow, hence it is more generic and unconstrained than the simple unification based information flow.

Some examples can clarify this kind of information flow and the appropriateness of treating gaps using the "for" feature described above. Consider the following sentence.

రాముడికి ఈపరీక్షలో మంచి మార్కులు వస్తే మనాందరికీ పార్టీ ఇస్తాడు

rAmuDiki eepareekshalO maMci mArkulu vastE manaMdarik.ee pArTee istADu --6.17

If Rama gets good marks in this exam, he will give us a party

In the above sentence, *rAmuDiki* is a complement of the main verb, *vastE,* of the embedded sentence. It has the correct suffix also, as required by the *aakaamksha* of *vastE.* Even so, it can satisfy the *aakaamksha* of the main verb, *istADu,* belonging to the matrix sentence. Hence, a filler in the embedded clause is able to fill a gap in the matrix sentence. It could be argued that *rAmuDiki* really belongs to the matrix sentence and only fills the gap in the embedded clause. If this is true, then as per Telugu syntax, any surface order of complements must form an acceptable sentence. Hence, every sentence that is a surface word order variant (at the complements level) of the above sentence must be grammatical. However, a minor variation of the given word order, where the word *rAmuDiki* is scrambled with other complements of the matrix verb *vastE* as below, renders the sentence ungrammatical.

*ఈపరీక్షలో మంచి మార్కులు వస్తే రాముడికి మనందరికీ పార్టీ ఇస్తాడు

eepareekshalO maMci mArkulu vastE rAmuDiki manaMdarikee pArTee istADu

*ఈపరీక్షలో మంచి మార్కులు వస్తే మనందరికీ రాముడికి పార్టీ ఇస్తాడు

**eepareekshalO maMci mArkulu vastE manaMdarikee rAmuDiki pArTee istADu*

Hence *RamuDiki* rightly belongs to the embedded sentence. Not only this, the following sentence is also valid; the matrix sentence has its own subject complement, *sOmu.*

రాముడికి ఈపరీక్షలో మంచి మార్కులు వస్తే సోము మనందరికీ పార్టీ ఇస్తాడు

rAmuDiki eepareekshalO maMci mArkulu vastE sOmu manaMdarikee pArTee istADu

If Rama gets good marks in this exam, then Somu will give us a party

On the other hand, in the following sentence, which has the same meaning as the example sentence 6.17, the word *rAmuDu* really belongs to the matrix sentence. This is because scrambling the word *rAmuDu* across the matrix sentence does neither harm the syntax nor the semantics of the sentence.

రాముడు ఈపరీక్షలో మంచి మార్కులు వస్తే మనందరికీ పార్టీ ఇస్తాడు
rAmuDu eepareekshalO maMci mArkulu vastE manaMdarikee pArTee istADu

ఈపరీక్షలో మంచి మార్కులు వస్తే రాముడు మనందరికీ పార్టీ ఇస్తాడు
eepareekshalO maMci mArkulu vastE rAmuDu manaMdarikee pArTee istADu

ఈపరీక్షలో మంచి మార్కులు వస్తే మనందరికీ రాముడు పార్టీ ఇస్తాడు
eepareekshalO maMci mArkulu vastE manaMdarikee rAmuDu pArTee istADu

The above example sentences prove that fillers in the embedded clauses can fill gaps in the matrix sentences in Telugu. In the above example sentence, 6.17, the embedded clause's complement was having the suffix *ki.* Another case where the embedded clause's forwardable complement has null suffix (ie. nominative case) can be taken. Even in these cases, it can be seen that a filler in the embedded clause fills the gap in the matrix sentences

అతను స్టేషనుకి ఆలస్యంగా వెళ్ళినప్పటికీ రైలు దొరికింది
atanu sTEshanki AlasyaMgA veLLinappaTikee railu dorikiMdi
Even though he went late to the station, he got the train

In the above sentence the word *atanu* can be scrambled within the embedded sentence without altering the meaning of the sentence. Whereas, it can not be moved into the matrix sentence as that would render the sentence syntactically unacceptable. Whereas in the following sentences which convey the same meaning as the above example sentence, the word *ataniki* can be freely moved across the matrix sentence.

అతనికి స్టేషనుకి ఆలస్యంగా వెళ్ళినప్పటికీ రైలు దొరికింది
ataniki sTEshanki AlasyagA veLLinappaTikee railu dorikiMdi

స్టేషనుకి ఆలస్యంగా వెళ్ళినప్పటికీ అతనికి రైలు దొరికింది
sTEshanki AlasyagA veLLinappaTikee ataniki railu dorikiMdi

The question now remaining to be answered is, how does one relate the complements in the embedded clause to the complements in the matrix sentence? Linguistic literature in this regard is again unhelpful, and some heuristics were developed using the textual corpus. By observing several Telugu sentences it was realized that suffix changes can be accounted for, to a fair degree of accuracy, as indicated in the following table. In the following table, the entry X--> Y indicates that a complement in the embedded clause with a suffix X can fill a gap in the matrix clause with an expected suffix Y. The entry X <— Y indicates that a gap in

the embedded clause with an expected suffix X can be filled by a complement in the matrix clause with the suffix Y. In either case the entity X must be in the "for" list of the verb.

Change	Condition
subj —> subj	If subj complement is missing in matrix clause.
ki or ni —> subj	If there is no subj in embedded clause and matrix clause.
ni —> ni	If accusative complement is missing in matrix clause.
ni or ki —> ki	If dative complement is missing in the matrix sentence.
subj --> ki	If dative or subj complement is missing in the matrix clause.
tO —> subj	If subj complement is missing in the matrix clause (less reliable rule).
ki <— subj or ni	If dative complement is missing in the embedded clause.
subj <-- subj or ki	If subj complement is missing in the embedded clause.

In the situation where fillers are in the embedded clause and gaps are in the matrix clause, the fillers which are on the "for" feature list are passed to the matrix clause as default complements (at the end of the complement list indicated by the first argument of apply procedure). This is done at the time of processing the subordination suffixes such as *i, tE, akuMDA,* and *inA.* If default complements are not consumed by the matrix sentence, no error is indicated. If the embedded clause has gaps, then they will be available for inspection in the for list of the embedded clause when the matrix clause is being processed. The filler procedure (described in Chapter 5) while matching *yogyata* to *aakaamksha* of the matrix verb, comes across the embedded clause. At that stage, the filler procedure looks at the missing complements and matches them to the entries in the *aakaamksha* of the matrix verb. Once suitable matches are found, the values of the "var" feature are unified indicating sharing of complements. In this fashion, information flow in both directions is achieved, with the possibility of default value assignment to matrix complements.

6.4 Handling Comparative Sentences

Handling comparative sentences in Telugu is challenging both from the syntactic point of view and from the semantic point of view. In, Telugu, comparative sentence must necessarily have a degree adjective like *ekkuva* (more), *takkuva* (less) , *tella* (white) , *pedda*

(old/big) or an adverb like *muMdara* **(early/before)**, *tvaragA* (fast), *kOpaMgA* (angrily) and a noun with the suffix *kAMTE* as its constituents. Typical examples of comparative sentences in Telugu are the following.

రాముడు సీతకంటే ముందర ఇంటికి వెళ్లాడు
rAmuDu seetakaMTE muMdara iMTiki veLlADu **--6.18**
Rama went home earlier than Sita

రాముడు సీతకంటే రాధకి ఎక్కువ పూలు ఇచ్చాడు
rAmuDu seetakaMTE rAdhaki ekkuva ***poolu*** *iccADu* **--6.19**
Rama gave more flowers to Radha than to Sita

రాముడు సీతకంటే పది పూలు ఎక్కువ కోశాడు
rAmuDu seetakaMTE padi poolu takkuva kOSADu —6.20
Rama plucked flowers ten less than Sita

సీతకంటే ఐదు నిమిషాలు తరువాత రాముడు ఇల్లు చేరాడు
seetakaMTE aidu nimishAlu taruvAta rAmuDu ***illu*** *cErADu* --6.21
Five minutes after Sita, Rama reached home

సీత కంటే ముందర ఇంటికి వెళ్లిన వాళ్లు అందరూ చేతులు ఎత్తండి
seeta kaMTE muMdara iMTiki veLlina ***vALlu*** *aMdaroo cEtulu ettaMDi* --6.22
All people who went home earlier than Sita raise your hands

రాముడికి వచ్చే జీతంకంటే సీత చేసే ఖర్చు ఎక్కువ
rAmuDiki vaccE jeetaMkAMTE seeta cEsE kharcu ekkuva —6.23
Sita spends more than what rama gets as salary

రాముడు సీతకి ఇచ్చిన బీళ్లలకంటే రాధకి ఎక్కువ పళ్లు ఇచ్చాడు:
rAmuDu ***seetaku*** *iccina biUalakaMTE rAdhaki* ***ekkuva paLlu*** *iccADu* —6.24
Rama gave more fruits to Radha than he gave toffees to Sita

రాముడు సీతకంటే పొడుగైన వాళ్లను జట్టులోకి చేర్చుకొన్నాడు
rAmuDu seetakaMTE poDugaina vALlanu jaTTulOki cErcukonnADu —6.25
Rama admitted people taller than Sita into the team

రాముడు అనుకున్న దానికంటే సీత త్వరగా ఇంటికి వచ్చింది

rAmuDu anukunna dAnikaMTE *seeta* ***tvaragA iMTiki vacciMdi*** -6.26

Sita came home earlier than **rama** thought

రాముడు ప్రపంచ రికార్డుకంటే వేగంగా పరిగెత్తాడు

rAmuDu ***prapaMca rikArDukaMTE*** *vEgaMgA parigettADu* --6.27

Rama ran faster than the world record

The above sentences show a large variation in the syntactic structure of the comparative sentences. Sentences 6.18 to 6.21 are examples of comparative ellipses. The sentences 6.24 to 6.26 are typical clausal comparative sentences. Sentence 6.22 and 6.23 have relative clauses with embedded comparative clauses and are mixtures of both the categories. Sentence 6.27 is valid though peculiar. It does not fall under either category. It is called in this thesis as a non standard comparative.

There is very little linguistic literature covering both the syntactic and semantic processing of comparatives in GPSG [Klei80], LFG [Bres77] and GB formalisms. HPSG [Poll87j and JPSG [Gunj87] have not dealt with comparatives at all. Practical systems like TEAM [Gros87], Linguistic String Project [Sage81], and CLE [Alsh89] have dealt with some simple cases of comparative constructions only. In linguistic literature most of the issues dealt with are syntactic in nature, and do not get carried to Telugu. In order to deal with comparatives in TELANGANA, novel and detailed theories had to be developed to analyse the syntax and semantics of comparative sentences. Consequenlty, TELANGANA can handle, both the syntactic and semantic issues of comparatives, in considerable depth.

Observing the example sentences given above it is easy to see that, nouns conjoined with the postposition *kaMTE* indicate the presence of a comparative construction in Telugu. Let such a noun be called CC (comparative complement). The CC has to occur to the left of the Comparative Operator (CO) which could either an adverb as in sentences 6.18 (*muMdara*) and *6.2l(taruvAta)* or an adjective/quantifier as in 6.19 *(ekkuva)* and 6.20 *(takkuva).* A CC can occur to the left of a CO with any number of intervening words as in 6.19, 6.23 and 6.24. If the CC occurs to the right of a CO then either it does not belong to that CO or it is an erroneous sentence. For example, observe the following sentences.

రాముడు సీతకంటే ధనవంతులని పొగిడాడు

rAmuDu ***seetakaMTE*** *dhanavaMtulani pogiDADu*

Rama praised people wealthier than Sita

*రాముడు ధనవంతులని సీతకంటే పొగిడాడు

rAmuDu dhanavaMtulani *seetakaMTE* **pogiDADu**

రాముడు ధనవంతులని సీతకంటే ముందర పొగిడాడు

rAmuDu *dhanavaMtulani seetakaMTE* **muMdara** *pogiDADu*

Rama praised wealthy people before Sita did

రాముడు సీతకంటే ధనవంతులని ముందర పొగిడాడు

rAmuDu seetakaMTE dhanavaMtulani muMdara pogiDADu

Rama first praised people wealthier than Sita

Syntactically all the above sentences can be handled by positing optional *aakaamksha,* in adverbs and adjectives, for a complement noun with suffix *kaMTE.* However the standard relative position markers like left (1) and right (r) are inadequate for the purpose, as they do not allow for any intervening complements belonging to other words. Hence, the type of position marker needed to cater to CCs is different. As already explained in Chapter 3, it is "lr", indicating "left but remote". The value "lr" in the *aakaamksha* of a word indicates to the filler procedure that the complement needs to be down the list of complements accumulated so far, ignoring the intervening ones. If more than one CCs are present or if the CCs are misplaced, then an error is indicated. Thus, syntactic errors can be taken care of easily. The problem that now needs to be attended to is, how does one construct the semantics of the comparative sentences? The next section details this.

As indicated above, the comparative sentences 6.18 to 6.27 can be categorized into three classes. Elliptical comparatives, clausal comparatives and non standard comparatives. Computation of semantics is predicated on the type of the comparative sentence. Determining the type of the comparative sentence is highly dependent on the sortal types of the complements and the *CC.* The following two comparative sentence belong to two different categories owing to the differences in sortal category of CCs and their matching complements. The matching complement is called comparand complement (COMC). In the following two sentences CC, CO and COMC are marked using subscripts.

రాముడు ఆ మనిషి కంటే ఎక్కువ పూలు కోశాడు

rAmuDu$_{COMC}$ *aa* **manishi**$_{CC}$ *kaMTE* **ekkuva**$_{CO}$ *poolu kOSADu* —6.28

Rama plucked more flowers than that person

రాముడు ఆపూల కంటే ఎక్కువ పూలు కోశాడు

rAmuDu aapoola$_{CC}$ kaMTE ekkuva$_{CO}$ poolu$_{COMC}$kOSADu **-6.29**

Rama plucked more flowers than those flowers.

In the first sentence, it is the case that both Rama and "that person" have plucked the flowers and that the number of flowers plucked by Rama is greater than the number of flowers plucked by "that person". However, in the sentence it is not explicitly stated that "that person" plucked the flowers. This is what makes this an elliptical comparative sentence. That is, the semantics of CC is not self contained, it requires the support of the matrix sentence. In the second sentence there is no implication that Rama plucked those flowers *(aapoolu).* The reference to those flowers is indeed a reference to their number. The second sentence thus is not an elliptical sentence and is purely a clausal comparative sentence. The meaning of the CC is self contained. To see which complement in a sentence is a possible COMC one needs to try for sortal match between the CC and all the complements of the main verb of the sentence. If the sortal type of the CC matches the sortal type of some complement X in the sentence then X becomes COMC.

In the elliptical comparative sentences, mostly, both COMC and CC do not have degree modifiers (eg. 6.18, 6.19, 6.20) and they occur before CO in the sentence. If COMC has a degree adjective and also occurs earlier than CC in the sentence, then also the sentence is deemed to be elliptical. If the COMC with the degree modifier occurs later than CC in the sentence and the modifier is not *ekkuva* or *takkuva,* then such sentences are not usually acceptable to native speakers. This is because they lead to ambiguities at times. For example, the following sentence was not acceptable to many native speakers as a good sentence as it leads to the following ambiguity.

రాముడు సీత కంటే తెల్ల అమ్మాయిని ముందర చూశాడు

rAmuDu$_{COMC}$ seeta$_{CC}$kaMTE tella ?$_{CO}$ ammAyini muMdara?$_{CO}$ COOSADU

Rama saw a fair girl earlier than Sita

or

Rama saw a girl fairer than Sita earlier.

However, the first meaning is generally choosen as more appropriate. In all such cases, the heuristic used in TELANGANA is to consider the sentential adverb as the appropriate CC and not the adjective modifier of the COMC as the CC. This heuristic prefers the first meaning over the second meaning.

In the case of clausal comparatives, COMC should necessarily occur later than CC in a sentence, and also the modifier of COMC should be the CO which subcategorizes for CC. It may be observed that this condition holds good in sentences 6.24 to 6.26. For example in the sentence 6.29 the CO *ekkuva,* which is a degree adjective, modifies COMC. The lexical information for the word *ekkuva* has *aakaamksha* for a comparative complement. This optional *aakaamksha* is satisfied by the CC ***aapoola kaMTE.*** Hence 6.29 is a clausal complement.

In non standard comparative sentences, there is often no sortal match between the CC and any of the verbal complements, as in the sentence 6.27.

Implementing the above methods of syntactic identification of comparative clauses raises a curious problem for the parser. Considering the above sentence 6.29 as an example, when the parser encounters the word *tella* it has to make a decision as to accept *seeta kaMTE* as a comparative complement of *tella* or not. If it accepts it, then the subsequent occurrence of the word *muMdara* will create a problem, because *seeta kaMTE* indeed must be a complement of *muMdara,* instead of being a complement of *tella* as mentioned already. If *seeta kaMTE* is not taken as a comparative complement of *tella,* it can lead to problems in cases where there is no subsequent adverbial modifier as in the following sentence.

రాముడు సీత కంటే తెల్ల అమ్మాయిని చూశాడు రాముడు

rAmuDu$_{COMC}$ *seeta*$_{CC}$ *kaMTE tella*$_{CO}$ *ammAyini cooSADu*

Rama saw a girl fairer than Sita

There are two possible solutions to this problem. The parser can accept the CC, *seeta kaMTE,* as a complement of *tella* initially. Then subsequently, when the adverb ***muMdara*** is encountered by the parser, the CC *seeta* can be removed from *tella* and attached to the word ***muMDAra.*** This means one has to undo some work already done. This is against the spirit of deterministic parsing [Marc80]. The other possibility is to leave the CC *seeta kaMTE* as it is and take care of it in the filler routine while matching the *yogyata* of the complements to the *aakaamksha* of the main verb. This means matching *yogyata* and *aakaamksha* of comparative complements and comparative operators must be done lazily. Till now all such matching has been done eagerly. Thus if a lazy matching strategy is used, then this problem of matching CC to CO is solved more elegantly. TELANGANA uses this approach. In the filler procedure, when a word with *aakaamksha* such as aak:rl:X is encountered, the search for the CC is initiated. Once the CC is found, it is hooked to the FB of the CO as a feature value of the feature "comp".

Matching a CC to a CO only solves the syntactic aspects of processing comparative sentences. The semantic aspect lies in locating the COMC which was meant to be compared against the CC. This task cannot be done eagerly in the filler procedure itself at the time of matching complements to the *aakaamksha* of the main verb, because, in the case of relative clauses, COMC may not be present till the antecedent of the relative clause is processed. For example in the following relative clause,

నిన్నటి కంటే ముందర ఇంటికి వెళ్ళిన రోజు

rAmuDu ninnaTikaMTE muMdara iMTiki veLlina rOju

The day rama went home earlier than yesterday

the CC *ninnaTi kaMTE* has no corresponding (sortally matching) COMC in the relative clause. When the filler procedure gets invoked while processing the verb *VeLl*, it encounters the adverb *muMdara.* It looks for a CC and finds *ninnaTi kAMTE.* At this stage if it tries to locate the COMC matching the CC, then it can not find one. Hence it has to postpone all such match making till the antecedent is processed. At this stage there are two options. One can process and locate the COMC for CC at the time of filling in the antecedent or alternately postpone the matching until the next stage[6] of sentence analysis, namely Logical Form generation. In TELANGANA, the later approach is taken. This is motivated by the fact that to generate logical forms from the FBs (feature bundles) of sentences, one has to go through every substructure of the FB in any case. At that time, the additional work involved in locating the COMC can be done.

The exact work to be done, wherever it is done, depends on the type of the comparative construction. All comparative sentences have the comparative operator. The semantic content of the comparative operator CO is taken to be that of a numerical operator such as lessthan, greaterthan or equal with two arguments N1 and N2. These arguments indicate the measure of the degree adjectives or adverbs. For example, for the adjective *tella* (white), N1 and N2 may be some numbers indicating the degree of whiteness. For the adverb *muMdara* (early) N1 and N2 may be some numbers that stand for the time of the occurrence of the events. Hence the semantic denotation of CO would be some formula like lessthan(Nl,N2), greaterthan(N1,N2), and equal(N1,N2), meaning N2 is less than N1, N2 is greater than N1, and N2 is equal to N1 respectively. When N1 and N2 are abstracted out of the formula using β-reduction [Bare84, p 50-53], and the denotation of CO becomes λM. λN Π(M,N) where Π

[6]The different stages of sentential analysis are described in Chapter 3.

stands for operators like lessthan, greaterthan and equal. A more general denotation of for the degree adjectives is one which has three arguments. The first and second arguments are as before, the third one corresponds to the amount by which first is less than or more than the second. The following sentence motivates the neccesity for using three arguments.

రాముడు సీతకంటే 10 పూలు ఎక్కువ కోశాడు
rAmuDu seetakaMTE 10 poolu ekkuva kOSADu
Rama plucked 10 more flowers than Sita

The words "10 more", in the above sentence, precisely indicate how many more flowers did Rama pluck than did Sita. Hence the denotation for *ekkuva* should be λM. λN morethan(M,N,10). In either case the number of variables is two only. Let the abstracted denotation, λM. λN. n(M,N), be called IT. Comparative expressions like "many more", "much less", and "very many more", can also be given a denotation fitting the context.

6.4.1 Type 1 Comparatives

Let S be a sentence containing the entities CC, COMC and CO. S can be thought of as a sentence made up of the sentences S1 and S2, such that S1 consists the COMC and CO parts of S, and S2 just contains CO. Let the denotation of S1 be Φ. From Φ, the denotation of S1, and IT, the denotation of S2, the denotation of the sentence S can be formed. Since S1 is a simple non-comparative sentence its semantics can be easily computed using the techniques described so far. The semantic contents of S2 has been described in the last paragraph. Hence computing the semantic content of S from S1 and S2 would just need a suitable rule schemata. This idea of rule schemata is explained in the next few paragraphs. For example the sentence

రాముడు సీతకి రాధకంటే ఎక్కువ పూలు ఇచ్చాడు
rAmuDu seetaki rAdhakaMTE ekkuva poolu iccADu

can be thought of as

రాముడు సీతకి ఎక్కువ పూలు ఇచ్చాడు + రాధకంటె
rAmuDu seetaki ekkuva poolu iccADu + rAdhakaMTE

The left side of "+" in the above line is S1 and the right side is S2.

The denotation of S1 has two important parts; one contributed by the COMC and the other contributed by the degree modifier CO. By giving a wide scope to the variables belonging to the comparative operators and by ignoring other quantifiers, the denotation of S1, Φ, using predicate calculus can be written as

3N. $\exists$M.{ivv(rAmuDu', seeta', q(some, X, puvvu(X) & number(X,N))) &
ekkuva(N,M,K)}

where rAmuDu' and seeta[1] are short hand notations to mean the denotation of rAmuDu and seeta respectively. For simplicity, event and state variables are not shown in this section unless their specification has a special bearing to the discussion on hand. In the above formula the contribution of the complement, COMC, is rAmuDu[1] and the contribution of the comparative operator, *ekkuva*, is the sub-formulae number(X,N) and ekkuva(N,M,K). Breaking the above formula at ekkuva(N,M,K) one gets

ivv(rAmuDu', seeta', q(some, X, puvvu(X) & number(X,N)) --6.30

ekkuva(N,M,K) --6.31

The above two formulas are instances of Φ and Π respectively. These two formulae can be further abstracted by β-reduction [Bare84, pp 50-52] on some of their arguments. Abstracting COMC, and N out of the first formula gives

AR. AN. {ivv(R,seeta, q(some,X, povvu(X) & number(X,N)))} -6.32

and abstracting the predicate "ekkuva" from the second formula gives

AP.AN. AM. P(N,M,K) --6.33

In the above formula mostly K is bound to a constant. The formula, 6.32, is an abstract version of an instance of Φ, 6.30, with respect to the COMC complement. The formula, 6.33, is an abstraction of an instance of Π, 6.31.

Let such an abstract version of Φ be called Φ'. Let Ψ_1, Ψ_2 represent the denotations of COMC and CC. Then giving wide scope to N_1 and N_2, the denotation of the sample comparative ellipse sentence can be re-written more abstractly as

$\exists N_1. \exists N_2. \Phi'(\Psi_1,N_1) \& \Phi'(\Psi_2,N_2) \& \Pi'(N_1,N_2)$ --6.34

The above formula gives a clue to **the** most generic form the denotation of a comparative sentence. In the most general case where quantifier scoping of quantifiers belonging to Ψ_1, Ψ_2, fl>\ and IT, could be arbitrary, the formula, 6.34, is best described using a rule-schemata such as

$$\text{kaMTE}(\Pi'(N_1,N_2), \lambda P.\lambda N.\Phi'(P,N), \Psi_1,\Psi_2) \quad \text{--6.35}$$

This rule schemata encodes very compactly the denotation of a comparative sentence S, in terms of the denotations of its constituent comparative operator, comparative complement and the rest of the clauses of the sentence. The above rule schemata, should be translated to the following logical form.

$$\Pi'(q(\text{exw}, N_1, \text{int}(N_1) \&\Phi'(\Psi_1,N_1)), q(\text{exw},N_2, \text{int}(N_2) \& \Phi'(\Psi_2,N_2))) \quad \text{--6.36}$$

In the above formula, the quantifier "exw", is a special case, a weaker form, of the existential quantifier "ex" and allows the embedded quantifiers such as "all" to outscope. Thus the predicate kaMTE acts as a template that can emit the correct semantics of the comparative sentence given Ψ_1, Ψ_2 Φ', and IT. It may be noted that for type 1 comparative sentences, the top level predicate denoting the semantics of the entire sentence would be a kaMTE predicate. The purpose of parsing comparative sentences in TELANGANA is to develop the above kind of an encoding of the sentence. The Meaning Representation Language (Chapter 3) encodes the above kind of information for comparative sentences using the predicate kaMTE. The Logical Form generation step (Chapter 7) which expects formulae to be in a certain format, expands this formula into the correct form before quantifier scoping related modifications are performed.

6.4.2 Type 2 Comparatives

In the type 1 comparative sentences both the third and the fourth arguments of the rule schemata F6.6 would be nominals. However, in the case of type 2 comparative sentences, the third argument, which corresponds to the CC, will be a relative clause missing a sentential complement. That means Ψ_1 has a missing sentential complement. In other words, it is of the form $\lambda P.\alpha(P)$ where P belongs to sentential category. This missing complement can be filled by the predicate of the matrix sentence. This difference in semantics is accompanied by a matching difference in syntax also. The CC, instead of being a simple nominal complement, would be a pronoun in feminine gender modified by a relative clause. The relative clause would be missing a sentential complement whose role the antecedent pronoun

can take on. The following sentence illustrates both these syntactic and semantic phenomenon.

రాముడు సీత అనుకున్న దానికంటే ముందర ఇల్లు చేరాడు

rAmuDu seeta ***anukunna dAnikaMTE muMdara iLlu*** *cErADu* -6.37

Rama reached home earlier than Sita thought.

In the above sentence the comparison is between the reality and what Sita thought about Rama's reaching home. This makes type 2 comparative sentences interesting and also leads to some amount of semantic complication. This kind of sentence being intensional [Davi90, pp 74-76], can really have two readings; *de-dicto* and *de-re* . However out of these two reading, *de-re*[1] reading is most appropriate with respect to the variable indicating the degree of comparison, and *de-dicto* reading is most appropriate with respect to the event/state variable, in most cases of comparative sentences. This approach is consistent with the naive semantics of the above sentence, 6.37, because Sita takes some time value, say 5:30PM, to be the arrival time of Rama. Whereas the event of Rama reaching in that time is a factious event, entirely in the mind of Sita. Hence, the event variable indicating the factious event gets a *de-dicto* interpretation. This topic of *de-dicto* and *de-re* meanings is extremely complicated and leads one into modal logic and possible worlds [Davi90, Chapter 2]. In the context of relational databases, the application domain of this thesis, *de-re* and *de-dicto* do not have much significance as relational databases tend to encode extensional information only [Ullm88, pp 78-79]. Hence they can be ignored without significant loss.

The rule-schemata for type 2 comparative sentences gives, as indicated above, a blend of *de-re* and *de-dicto* interpretation to intensional sentences. Consequently, wide scopes are assigned to N_1 and N_2 in the rule schemata. Subsequent processing can change the scope of the quantifiers. Hence, except for the preassigned rigidity of *de-dicto* and *de-re* readings, all other readings, including the most appropriate quantifier scoping reading, can be easily generated.

The discussion above leads to the following rule schemata for type 2 comparative sentences

$$\text{kaMTE}(\Pi'(N_1,N_2), \lambda P.\ \lambda N.\Phi'(P,N), \Psi_1,\Psi_2)$$

[1] It is quite simple to change the rule-schemata to give the *de-dicto* reading also.

The translation of the above rule-schemata is done as below.

$$\Pi'(\text{q(exw, } N_1, \text{int}(N_1) \,\&\Psi_1(\Phi(\Psi_2, N_1))), \text{q(exw, } N_2, \text{int}(N_2) \,\&\, \Phi(\Psi_2, N_2)))$$

While translating the rule-schemata, one can choose the correct translation by analysing the form of third argument. If it has lambda abstraction then it must be type 2 comparative kaMTE. Hence even though only one rule-schemata is given for both cases of comparative sentences there would be no confusion.

For example, with respect to the above sentence,

$XP.\ \lambda N.\Phi'(P,N) = \lambda P.\ \lambda N.$ cEr(q(ex, E, samdarbham(E) & time(E,N)),P, home[1])
$\Psi_1 = XP.$ anukon(sita', P)
Ψ_2 = rama'
$\Pi' = \lambda M.\ \lambda N.$ tvaraga(M,N,K)

hence the denotation of the entire sentence is

tvaraga(q(exw,N_1, int(N_1) & anukon(sita',cEr(q(ex, E_1, samdarbham(E_1) &
time(E_1,N_1)),rama', home'))),
q(exw,N_2, int(N_2) & cEr(q(ex, E_2, samdarbham(E2) &
time(E_2,N)),rama', home'))),K).

The above sentence after suitable quantifier scoping yields

q(exw,Nl, int(Nl) & anukon(sita', q(ex, E_1, samdarbham(E_1) & time(E_1,N_1),
cEr(E_1,rama',home'))),

q(exw, N_2, int(N_2) & q(ex, E_2, samdarbham(E_2) & time(E_2,N_2),
cEr(E_2, rama[1], home')), tvaraga(N_1,N_2,K)))

which means

$\exists N_1.\ \exists N_2.\exists E_2.$ anukon(sita,$\exists E_1$.cEr(E_1,rama,home)& time(E_1,N_1)) &
cEr(E_2,rama,home) & time(E_2,N_2) & tvaraga(N_1,N_2, K)

The event variables E_1 and E_2 represent the factious event and the real event. In most database applications, however, there is no necessity to represent sentential information as above. It is only in knowledge based systems that such sentential embedding become important.

6.4.3 Type 3 Comparatives

In type 3 comparative sentences, the third argument has no relation to the second argument. In the following sentence,

రాముడు ప్రపంచ రికార్డుకంటే వేగంగా పరిగెత్తాడు

rAmuDu prapaMca rikArDu kaMTE vEgaMgA parigettADu --6.38

Rama ran faster than the world record.

the world record does not run, as in type 1 comparative sentences, nor does the word "world record" indicate or imply a sentence missing a sentential complement, as in the type 2 comparative sentences. Additionally, with respect to semantic category the word has no proximity to the person *rAmuDu*. This makes recognizing this type of sentences easy. Hence, while analysing comparative sentences, when no semantic compatibility is perceived between CC and COMC, a simple trick is used to get the effect of CC not actually participating in the action indicated by the matrix verb. This is done by taking the semantic content of CC to be a A-abstracted formula without the lambda abstracted variable occurring any where in the formula! Hence type 3 sentences can be treated as type 2 sentences, with the addition of the above minor twist. The above twist deletes the re-use of the matrix verb and at the same time introduces the comparand variable signifying the CC. This twist may be viewd as semantic type raising; a semantic variant of the notion of type raising in Categorial Grammars [Vanb87, Stee91].

An example best illustrates the processing. Taking 6.31 as the example sentence, one can see that

XP. $\lambda N.\Phi'(P,N)$ = AP. AN. pariget(q(ex, E, samdarbham(E) & speed(E,N)),P)

Ψ_1 =AP. world_record(N)

Ψ_2 = rama[1]

Π' = AM. AN. vEgaMgA(M,N,K)

Hence, the denotation of 6.31 is

kaMTE($vEgaMgA(N_1,N_2,K)$,λP. AN. pariget(q(ex, E, samdarbham(E) & $speed(E,N))$,P), AP. world_record(N), rama')

which means

$vEgaMgA($ q(exw, N_1, $int(N_1)$& pariget(q(ex, E, samdarbham(E)& $speed(E,N_1)$), rama')),

$$q(exw, N_2, int(N_2) \& world\_record(N_2)), K)$$

which is the intended denotation.

The above three types of comparative constructions exhaust the most frequently occurring comparative sentence formations in Telugu. The semantics of the less frequently occurring comparative sentences consisting of phrases such as *emta tvaragA avutE amta tvaragA* (as soon as), and *enni xyz anni* (as many as xyz) can also be analysed using the above rule-schemata. However, recognizing them syntactically requires more idiosyncratic processing. Such cases have not been considered in TELANGANA.

The last type of linguistic phenomenon to be considered in this thesis is handling pronoun references and anaphora. Since there is no discourse component in TELANGANA the treatment of anaphora in TELANGANA is limited to inter-sentential references only. For such inter-sentential references both reflexives pronouns and ordinary pronouns can be used in Telugu.

6.5 Handling Pronouns

Traditionally, the analysis of what a pronoun refers to in a sentence, has been done by taking recourse to the phrase structure tree of the sentence. Transformational grammar and GB theory [Chom81, Chom86] both have advocated, the binding theory (Chapter 2.3.1), which lays down structural constraints such as c-command on phrase structure trees to identify the possible referents of pronouns and reflexives. In TELANGANA however, as stated repeatedly, there is no notion of phrase structure. The question that naturally arises is, how does one analyse pronominal references when phrase structure is not conceived of? In this section an alternate approach based on the contents of *aakaamksha* is presented.

The words *vADu, atanu, AmE, vALlu, nuvvu, meeru, nEnu,* and *mEmu* are pronouns in Telugu. Double usage of the pronouns[8] as in *nannu nEnu, nAku nEnu, vALIni vAlu, vALallO vALlu, neeku nuvvu, memmalini meeru,* and *mAtO mEmu* makes them reflexive pronouns. Reciprocal pronouns are formed, again, by the double pronouns, such as *okaDini iMkokaDu*, and *okaLlatO iMkokaLlu*. The word *tana* is used in Telugu to indicate "self. For translation purposes, it is some what like the *zibun* of Japanese[Kuno73] but different in distributional properties as indicated by Wali[Wali91]. Unlike *zibun,* it can be doubled, just like the other

[8]except for atanu

Telugu pronouns, to give additional reflexive stress as in ***tanaku*** *tAnE,* and ***tanani tAnE.*** Because of these differences, the analysis of pronouns in Telugu needs special attention. Techniques borrowed from English, Japanese and other Indian languages can not be directly applied to analyse Telugu pronominal references.

A few examples of pronoun usage in Telugu are given below. As is common in the linguistic literature, the subscripts are used to indicate the nominals the pronouns refer to. When "*j" is used, it means that the given pronoun cannot refer to the word indicated by the subscript "j". The symbol, 7", is used to indicate the alternate words that can also be used in that context.

రాముడు వాడి / అతని / తన ఇంటికి వెళ్ళాడు
rAmuDu$_i$ vADi$*_{i/i}$ / atani$*_{i/i}$/ tana$_{i/*i}$ iMTiki veLlADu --6.39
Rama went to his$*_{i/i}$ / his $*_{i/i}$ / self$_{i/*j}$ home

రాముడు వాడి ఇంటికి వాడు వెళ్ళాడు
rAmuDu} vADi iMTiki vADu$_{i/*j}$ veLlADu --6.40
Rama went to himself s (his)$_{i/*j}$ home

అందరూ వాళ్ళ వాళ్ళ ఇళ్ళకు వెళ్ళారు
aMdaroo$_i$ vALla vALla$_{i/*j}$ iLlaku veLlAru --6.41
Every$_i$ one went to their$_{i/*i}$ repective homes

రాముడు అతనిని / తనని చూశాడు
rAmuDu} atanini$*_{i/i}$ / tanani $*_{i/j}$ cooSADu --6.42
Rama$_i$ saw him$*_{i/j}$ / self$*_{i/j}$

రాముడు సీతని ఆమె కథ చెప్పమని కోరాడు
rAmuDu} seetani; Ame$_{i/*i}$ kadha ceppamani kOrADu --6.43
Rama$_i$ asked Sita; to tell her$_{j/k}$ story

రాముడు సీతని తన ఇంటికి తీసుకు వెళ్ళాడు
rAmuDu} seetani tana} iMTiki teesuku veLlADu --6.44
Rama; took Sita to self(his)i home

రాముడు సీతని తన కథ చెప్పమని కోరాడు

rAmuDu} seetanij tana$_{j/i}$ kadha ceppamani kOrADu -6.45

Rama$_i$ asked sitaj to tell **self(her)$_j$/self(his)$_i$** story

వాసు మోహన్ తనకి పుస్తకం ఇచ్చాడని కమలతో చెప్పాడు

vAsu$_i$ mOhan$_j$ tanaki$_{i/*j}$ pustakaM iccADani kamalatO ceppADu **--6.46**

Vasu$_i$ told **Kamala$_k$** that **Mohan$_j$** gave the book to **self$_{i/*i/*k}$**

రాముడూ సీతా ఒకళ్లను ఒకళ్లు చూసుకొన్నారు

[rAmuDoo sitA]$_i$ okaLlanu okaLlu$_i$ coosukonnAru --6.47

[Rama and Sita]$_i$ saw one another$_i$

The above sentence brings out various facets of **pronominals** and their referents in Telugu sentences. The most prominent of all constraints on pronominal reference is that there has to be agreement between the pronoun and its referent in number, gender and person. The other constraints arise due the various referential properties of the pronouns. The referential properties of pronouns are best understood in terms of local and non-local domains.

A few definitions are given here before embarking on the main task specifying the constraints on pronominal referents. Let the word that has an *aakaamksha* for the pronoun be called the root word. The local domain of a pronoun is the set of words which are *samarthah* (see chapter 5) to it. All other words in the sentence are said to belong to a non-local domain with respect to that pronoun. For example in sentence 6.46, the words *mohan, pustakaM, iccADu* belong to the local domain of *tana* and the words *vAsu, kamalatO, ceppADu* are non-local. The root word is *iccADu.* In the sentence 6.45, the local domain of the pronoun *tana* is the single element set *{kadha}* and the nonlocal domain is the set *{rAmuDu, seetani, ceppamani, kOrADu).* The root word is *kadha.* Depending on the level of embedding of sentences, one can talk about the levels of nonlocal domains. The outer most nonlocal domain is called the maximal nonlocal domain. It may be observed that the notion of local domain, being defined in terms of *samarthah,* is intimately connected to the notion of *aakaamksha.* It may also be observed that processing of pronouns thus comes under the realm of the meta-rule ***samartha*** *padavidhih* of Asthaadhyayi [Josh68] explained in the first chapter. The concept of a local domain is central to the understanding of pronoun processing.

One simple approach to analysing pronouns in Telugu would be to see, to what English words they translate and then use pronominal reference constraints in English to study Telugu pronouns. This approach does not work well. For example the word, ***Ame*** of Telugu

can be translated to "she" or "her" in English. The pronoun binding rule in English can be stated[9] as, pronouns are free in their local domain and can be optionally bound in their non-local domain. Reflexive pronouns in English must be bound to the subject in their local domain. For example in the following two English sentences,

Rama talked to Sita about her
Rama talked to Sita about herself

the pronoun her in the first sentence can not refer to Sita and the pronoun herself in the second sentence can only refer to Sita. In the first sentence her necessarily refers to some person not mentioned in this sentence. Whereas in Telugu, as borne by the following sentence, the pronoun *Ame* can refer to Sita as if it were a reflexive or it can refer to some other third person as if it were a pronoun free in its local domain[10].

రాముడు సీతతో ఆమెగురించి మాట్లాడాడు
rAmuDu$_i$ seetatO AmeguriMci$_{i/j}$ mATlADADu
Rama$_i$ talked to Sita about herself$_i$/her$_i$

The pronouns *atanu* and *vADu* can be translated in to the pronoun "he". But neither of them entirely behaves like the English pronoun "he". In the following sentence, it is easy to see that there are differences in the referents (as accepted by a many native speakers) of *atanu/vADu* even though both these words are treated alike in many other contexts.

రాముడు వాసుకి ఈ పుస్తకంలో అతని / వాడి కథ చూపాడు
rAmuDu} vAsukij ee pustakaMlO atani$*_{i/j/k}$*/vADi$_{i/j/k}$ kadha coop ADu* --6.48
Rama showed Vasu his$_{i/j/k}$ story in this book

రాముడికి వాసు వాడిని / అతనిని అద్దంలో చూసాడని తెలుసు
rAmuDiki$_i$ vAsu$_j$ vADini$_{i/k}$ / atanini$*_{i/k}$ addaMlOcoosADani telusu* --6.41
Rama$_i$ knows that Vasuj saw him$_{i/k}$ / him$*_{i/k}$ in a mirror

[9]This is a considerbly simplified version of the actual binding principle [Chom86}.

[10]This interpretation is based on a survey conducted by the author wherein it was observed by some native Telugu speakers felt at first sight that *Ame* refers to Sita.

Pronoun binding rules from English, **and** from non-configurational languages like Japanese [Whit86] were found to be inadequate. Hence processing pronouns and reflexives in Telugu requires a classification other than traditionally accepted [Chom86, Spor86, **Whit86,** Wali91). Telugu pronouns are classified into four classes in TELANGANA based on their **ability** to refer to nominals present in the rest of the sentence. The first and second classes consists of only one member each; *tana* and ***atanu***[11]. The third class consists of pronouns *vADu,* ***Ame, vALlu, meeru, nuvvu,*** *neenu* and ***mEmu.*** For the sake of clarity, these are called personal pronouns in the rest of this section. The fourth category consists of reflexive and reciprocative pronouns such as *nannu nEnu,* ***nAku*** *nEnu,* ***vALlani vAlu,*** *vALallO vALlu, neeku nuwu, mimmalini meeru,* ***mAtO mEmu,*** and *okaLlatO* ***iMkokaLlu.***

The pronoun *tana* belonging to the first class of pronouns is mandatorily bound to another nominal X in its local domain only if the root word has a ***kon*** suffix and X is a subject. Otherwise it can not be bound in its local domain. The following sentence illustrates this aspect.

రాముడు తనని అద్దంలో చూశాడు
RamuDu$_i$ tanani$*_i$ addaMlO coosADu
Rama$_i$ saw self$*_i$ in the mirror

రాముడు తనని అద్దంలో చూసుకొన్నాడు
RamuDui tanani$_i$ addaMlO coosukonnADu
Ramaj saw himself$_i$ in a mirror

If *tana* is not bound in its local domain then it can be either bound to the subject in the maximal non-local domain or be free. This is illustrated by the following example

సీత రాముడికి తన సంగతి అంతా తెలిసిపోయిందని భయపడ్డది
seeta$_i$ rAmuDiki; tana$_{i/*i}$ saMgati aMtA telisipOyiMdani bhayapaDDadi
Sita got afraid that all about her became known to Rama

[11] ***atanu*** is given a separate class here. Most linguists tend to classify *atanu* into the third class presented here. Hence in the traditional classification there are only three classes. In TELANGANA four classes have been used based on the observed minor differences some native Telugu speakers cognized in the referents of *atanu* and *vADu* in sentences like 6.40. However, the parser in TELANGA itself is written independent of the number of such classes and works with any number of classes. The class information is represented in a table.

It may be noted that the subject nominal in a sentence can found by inspecting the **"for"** feature value of the root word and not the **"aak"** feature as is done normally for finding a complement This is so because, the ***aakaamksha*** information associated with a word indicates only suffixes, and the subject has no fixed suffix in Telugu. The "for" feature is thus useful for pronoun disambiguation also, in addition to aiding the analysis of certain conjunctive sentences (see section 6.2). Using this scheme for understanding the referents of *tana,* the example sentences 6.39, 6.42, 6.44-6.46 can be analysed easily.

The pronoun, *atanu,* is never bound in its local domain. It is free in its non-local domain. That is it can be bound or remain unbound. If it is bound, however, it must be bound to only a non-subjective non-interrogative nominal which occurs in the sentence earlier than *atanu.* The following example illustrates the behaviour of the pronoun *atanu.*

రాముడు వాసూని అతని/ గదిలో కలిశాడు
rAmuDu$_i$ vAsooni; atani$*_{i/j}$ gadilO kaliSADu
Rama$_i$ met Vasu; in his$*_{i/j}$ room

రాముడు అతని గదిలో వాసుని కలిశాడు
rAmuDu} atani$*_{i/*j/k}$ gadilO vAsunij kaliSADu
Rama$_i$ met vAsu$_j$ in his$*_{i/*j/k}$ room

రాముడు ఎవరిని అతని ఇంట్లో దింపాడు ?
rAmuDu evarini$_i$ atani$*_{i/k}$ iMTlO diMpADu ?
Who$_i$ did Rama drop at his$*_{i/k}$ home ?

రాముడు వాసుని అతని ఇంట్లో దింపాడు
rAmuDu vAsunij atani $_j$ iMTlO diMpADu
Rama droped Vasuj at hisj home

The third class of pronouns (*vADu, Ante,* ***vALlu*** etc.) behave very much like the second class of pronouns but for a minor difference. They are unbound in their local domain but can be bound in a non-local domain to any nominals which appear before the pronoun in the sentence.

రాముడికి వాసు వాడి పైంటింగ్ ప్రదర్శించలేదని కోపం వచ్చింది
rAmuDiki$_i$ vAsu$_j$ vADi$_{i/j}$ *paiyiMtiMg* ***pradarSiMcalEdani*** *kOpaM vacciMdi*

Rama$_i$ got angry that Vasuj did not display his$_{i/j}$ painting

రాముడు ఆమెతో సీతగురించి మాట్లాడాడు
rAmuDu AmetO$_{k/*}$seetaguriMci$_i$ mATlADADu
Rama talked to her$_{k/*i}$ about Sita$_i$

రాముడు సీతతో ఆమెగురించి మాట్లాడాడు.
rAmuDu seetatO$_i$ AmeguriMci$_{*i/k}$mATlADADu
Rama spoke to Sita$_i$ about herself$_{*i}$/her$_k$

రాముడు సీతకి ఆమెలోని దోషాలు చూపాడు
rAmuDu seetaki$_i$ AmelOni$_{i/k}$ dOshAlu coopADu
Rama$_i$ showed Sita the defects in herself$_i$/her$_k$

The last type of pronouns are reflexive and reciprocal pronouns. The use of reflexive pronouns is always accompanied by the verbal auxiliary *kon* in Telugu. This aspect of the kon was pointed out with respect to the first type of pronoun *tana.* Due to the presence of *kon, tana* becomes reflexive and refers to the subject. However the reflexives to be treated now are more verbose and introduce reflexivization by duplicating and stressing the pronouns as in *tananu tAnE, tanaki tanE, vADini vADE,* and *vALlalO vALlE.* These duplicating pronouns can have interspersed nominals also as in *vADi kAUameeda vADE* (on his own legs), and *tana Asti amtA tanE* (all his property by himself). They are mandatorily bound to some element which is agreeable in number, gender, and person in their local domain.

రాముడు నిన్న చేసిన తప్పు గురించి తనని తానే నిందించుకున్నాడు
rAmuDu$_i$ ninna cEsina tappu guriMci tanani tAnE[niMdiMcukunnADu
Rama$_i$ blamed himself$_i$ for the mistake made yesterday

rAmuDu$_i$ vAsuki; seeta$_k$ tanani$_{*i/*j/k}$tAnE$_{*i/*j/k}$ nimdimcukonnadani ceppADu
Rama$_i$ told Vasuj that Sita$_k$ blamed herself$_{*i/*j/k}$

రాముడు తన కాలుమీద తానే రాయి పడేసుకొన్నాడు
rAmuDu} tana$_i$ kAlumeeda tAnE$_i$ rAyi paDEsukonnADu
Rama$_i$ droped a stone on his$_i$ (own) leg

There are two interesting pronoun doubling patterns in Telugu which appear close to reflexives but are not reflexives. Hence care must be taken in identifying reflexives. The doubling of pronoun with a minor variation where the first occurrence is an interrogative version of the second, is used as universal quantifiers in certain contexts as in

రాముడు ఎవరిని పడితే వాడిని నమ్మకాడు

rAmuDu evarini paDitE vADini nammutADu

Rama believes (each and) everyone

The above is quite obviously not a reflexive pronoun type of usage. The pattern for this kind of usage is interrogative pronoun (*evaru, evaLlu, evaDu, evatte* etc.) + word with *tE* suffix+ matching pronoun. This kind of pronoun pattern can be easily observed in the input sentence and suitable processing done to delineate such usages of double pronouns. Another similar pattern is interrogative pronoun+word+pronoun as in the following sentence.

రాముడు ఎవరి డబ్బులు వాళ్లకి తిరిగి ఇచ్చాడు

rAmuDu evari$_i$ Dabbulu vALlaki$_i$ tirigi iccADu

Rama returned to each person$_i$ hisj money

All such non reflexive uses of pronouns are detected using special rules in the apply procedure. However in TELANGANA, at present, all such patterns, being idiosyncratic, are inspected only to ensure absence of reflexives.

The table 6.1 summarises the kind of processing performed in TELANGANA for resolving pronominal referents.

category	Non-Local	Local
tana	+B,+S,~Q	*kon* & subj
atanu	~B,-S,-Q (left)	—
pro-pronoun	~B,~S,-Q (left)	—
reflexive	—	left

Table 6.1 Classification of Pronoun Processing

In Table 6.1, +B means necessarily bound, ~B means need not be bound, -S means should not be bound to a subject, +S means, should be only bound to a subject, and -Q means, should not be bound to an interrogative noun. Subsequent to **~B,** whatever appears, it is

taken to be conditional, that is, if the entity in first column were to be bound, it can be bound subject to the constraints appearing after ~B. The symbol left indicates that, if a pronoun is to be bound, then it must be bound to some referent that occurs to its left in the sentence. For example, consider the row describing *atanu*. It consists of the entry ~B, -S, -Q (left) in the non-local column and "——" in the local column. The entry in the local column indicates that *atanu* can not be bound in its local domain. The entry in the non-local column implies that *atanu* is free in its non-local domain. In other words, it is free either not to refer to any noun in its non-local domain or to refer to one. When it does not refer to any noun in its non-local domain, then it must refer to some extraneous discourse specific referent. But if it were to refer to some noun in its non-local domain, then that noun must be a non-subject and non-interrogative noun. Further, the noun must occur to the left of *atanu* in the sentence.

6.6 Conclusion

In this chapter, analysis of complex and coordinate sentences was presented. Processing these types of complex sentences requires many innovations in the theory underlying the parser, the information contents of the FBs in the lexicon, the apply procedure and the filler procedures. These innovations were presented at length. Some cases of gaps and fillers were presented which necessitate filling gaps with default values. This is a feature of Telugu language and does not appear to be present in English. This feature of Telugu necessitates abandoning the simple methods of gap filling used in the analysis of English [Gazd85, Poll87]. The gap-filler dependencies in Telugu are treated using a combination of *aakaamksha* related lexical information and suffix changing rules. Thus *aakaamksha* information seems to have a larger say in the processing of Telugu sentences than subcategorization information in English. Subsequently the role of *samarthah* in deciding the pronoun antecedents was shown. Thus the major principle used throughout the processing is *samarthah padavidhi* of *ashtaadhaayi* [Josh68].

The processing of Telugu sentences presented in this chapter is very much deeper and wider in the coverage of the language and closer in spirit to Asthaadhyaayi than the approaches taken by some researchers [Alwa88, Bhar90a, Bhar90b] working on other Indian languages. The author believes that the coverage of the syntax and semantics of Telugu sentences in TELANGANA is at par with many of the well known NLP systems developed for English and European languages such as ASK [Thom83], CHAT-80 [Warr80], SMACK-85 [Rayn86], and TEAM [Gros87].

Chapter 7

Quantification and Question-Answering

7.1 Introduction

The previous chapters presented details of the computation of feature bundles (FBs) for sentences through complex sentences. The meaning of a sentence is captured in a FB through predicates. These predicates, to attain maximum generality, tend to be based on considerations essentially linguistic in nature. The predicates so motivated are application neutral and hence require further massaging/transformations to become appropriate for an end application. The nature of the transformations depends upon the particular end application. In this thesis, accessing information from a relational databases was targeted as the application. In order to query databases two major transformations were required: quantifier scoping, and translation. In addition to the above owing to the format chosen for the FBs, a further step of restructuring the FBs into predicate form was necessary before quantifier scoping and translation related transformations could be performed.

Restructuring converts FBs into a form where quantifiers and the predicate structure of the query become explicit. The FB corresponding to the input query is converted into a predicate whose arguments are quantifier formulas. The quantifier scoping phase takes this predicate form as input and transforms it into the logical form (see chapter 3.5) where all quantifiers are appropriately scoped. The need to properly handle quantifiers arises when one has to answer queries whose meaning critically depends on the order of the quantified nominals in it. For example, in the following sentences the differences in meaning arise purely due to the order of the complements which contain quantifiers.

ప్రతి విద్యార్థి ఏసబ్జెక్టులలో తప్పాడో చూపుము

prati vidyArthi EsabjekTulalO tappADO coopumu

For each student, show me the subjects he failed in

ఏసబ్జెక్టులలో ప్రతి విద్యార్థి తప్పాడో చూపుము

EsabjekTulalO prati vidyArthi tappADO coopumu

Show me the subjects in which every student failed

In the above sentences, the relative order of quantifiers *prati* (every) and *E* (which plural) totally changes the meaning of the sentence. Approaches which do not recognise the importance of quantifiers in database access are doomed to failure.

The need for translating FBs arises because users of natural language interfaces tend to frame their questions in the way they think about the stored information. Their conception of the organization of the information is often different from the way the information is actually structured in the computer. Existing databases employ different representational conventions, many of them devised to satisfy various database management criteria. For example, database designers can encode the information that a student has passed in a relation, such as **student_record**[1], by using different fields and their corresponding values, for instance as "Y" or "N" in the pass field, or as a number in the marks field, implicitly carrying the information of passed or failed, or as "promoted", or "not-promoted" in the status field. The above information could have also been represented through different database relations, instead of the relation student_record. In whatever way the information is organised in the database, the natural language query to extract that information would be independent of that organisational detail. For example, the natural language query asking if somebody has failed in maths would be as follows,

రాముడు గణితంలో తప్పాడా?
rAmuDu gaNitaMlO tappADA ?
Did Rama fail in maths?

The predicate of the above query, is

tapp(rAmuDu, gaNitaM)

The above predicate does not have any explicit reference to **student_record,** or its fields. Hence it needs to be translated into one of the following[2]

student_record(,, rama , ,n) or
student_record(, , rama ,, MARKS) & MARKS < 35 or
not **(promoted(,**, rama,,,))

depending upon how the information is stored in the database. In certain contexts some predicates need to be ignored. For example, in the query

[1]Names in bold letters indicate either database relations or fileds in detabase relations.
[2]Pass marks is assumed to be 35 marks (as is true in many traditional Indian universities) in this chapter.

రాముడి తండ్రి పేరు చూపుము
rAmuDi taMDripEru coopumu
Show the name of Rama's father

the predicate corresponding to the word *coop* (show) does not get translated into a database relation and is in fact dropped in the final query to the database.

Another reason why translation is required is to optimise the output query. In the query that is generated after the quantifier scoping, there could be many predicates corresponding to the sortal constraints on the arguments of the predicates. These constraints may be unnecessary in the context of the database relations where the sortal types of fields are implicitly assumed owing to the database schema definition. For example,

ఏవిద్యార్థికి గణితంలో 100 మార్కులు వచ్చాయి ?
EvidyArthiki gaNitaMlO 100 mArkulu vaccAyi ?
Which student got 100 marks in Maths ?

The logical form for the above query would be

q(wh, X, student(X), student_marks(X,gaNitaM,100))

and if the database query is generated straight away it would be

student(X)&student-record(X,gaNitaM,100)

The relation, student, would not be defined in the database schema since the student_record relation implicitly assumes its first argument to be of type student. Query translation removes such redundancies in the translated queries.

This chapter details the above three transformations, restructuring, quantifier scoping and translation, which convert the output of the syntactic/semantic analyser into database specific queries. Throughout the rest of this chapter, a sample database is used to illustrate the various aspects of the above three transformations. In the next section the sample database is presented. In the subsequent sections, the three transformations are explained in detail with special emphasis on the quantifier scoping algorithm. These three transformations, and the pragmatics related processing detailed in Chapter 5, impart the flavour of the domain to the otherwise domain neutral processing of Telugu in TELANGANA.

7.2 The Sample Database

It is evident from the explanation given in the previous section, that the translation transformation is highly domain dependent. Hence, a well defined database needs to be used as a reference or sample database. The sample database considered here encodes the information about the students and the lecturers in a university, and comprises three relations, **student_record, student_marks,** and **lecturer_record**. There are a number of fields in each relation as shown in figure 7.1.

student record

stu_name	course	year	grade	college_name
rAma	iMTer	1989	a	caitanya
vidyA	bsc	1990	b	nijAm
aparNa	msc	1990	a	kOTi
rAma	bsc	1991	f	hiMdoo

student_marks

stu_name	subject	marks
rAma	gaNitaM	85
udaya	phijiks	78
aparNa	kemisTree	96

lecturer_record

lec_name	subject	course	year
bhArati	vEdAntaM	ma	1991
kANAda	akaunTs	bcom	1988
pataMjali	telugu	iMTer	1980
sAstri	kemisTree	msc	1978

Figure 7.1 Relations and Fields in the Sample Database

The first relation **student_record** has five fields, **stu_name, course, year, grade,** and **college_name**, that together specify a student's name, the course he/she undertook , the year the student left the university, the grade obtained by the student, and the college he/she studied in. The second relation, **student marks,** comprises of three fields, **stu_name, subject** and **marks,** which together specify the student's name, the subject he/she studied

and the marks obtained in that subject. A student typically studies many subjects in any given course. The third relation, **lecturer_record**, comprises of four fields, **lec_name, subject, course** and **year,** and indicates the lecturer who taught a particular subject in a particular course in a given year.

The important thing to note about the relations given above is that some field names occur in more than one relation. The common fields allow queries that span more than one relation. For example the query,

ఉదయకి 1990 లో ఫిసిక్సులో ఎన్ని మర్కులు వచ్చాయి ?
udayaki 1990lO phisiksulO enni inArkulu vaccAyi ?
How many marks did Uday get in Physics in 1990?

involves references to both the relations **student_marks** and **student-record.** Hence the predicate *vacc* needs to be translated into two database relations. Common fields thus entail complications in the translation stage.

7.3 Query Restructuring

The details of the restructuring transformation are presented in this section. The FB of a sentence is the input to this phase of processing and the output is a predicate with quantified arguments. The operation of the restructuring program is best explained using an example. Consider the following sentence,

రాముడికి కెమిస్ట్రిలో ఎన్ని మార్కులు వచ్చాయి?
rAmuDiki kemisTreelO enni mArkulu vaccAyi?
How many marks did Rama get in Chemistry?

and the following FB generated for it by syntactic and semantic analyser.

```
[ aak: [],
  word_loc:5,
 sentype: [none,q],
 var:X0,
 yog: [pos:v, isa:kriya, agr: [ [m,s,3],_]],
 sem: [        reln: vacc,
       saMdrabham: XO,
       vidyArthi:    [aak: [],
                      word_loc:l,
```

```
                var:X1,post:ki,
                yog: [ pos:n,isa:human,agr: [ [m,s,3],J],
                sem: [ reln: name,id: X1,string:rAma]],
vishayam:       [aak: [],
                word_loc:2,
                var:X2,post:lO,
                yog: [pos:n,isa:subject, agr: [ [f,s,3],_]],
                sem: [ reln: name,id:X2,string: kemisTrec],
mArkulu:        [aak:[],
                 word_loc:4,
                 var:X3,
                 yog:[pos:n,isa:maark,agr:[[f,pl,3],_]],
                 sem:[reln:maark, id:X3],
                 quant:[ aak:[],
                         word_loc:3,
                         yog: [pos:quant, isa:det,agr:[[_,pl,_],[def,cnt]],
                         var:X3, [reln:number, id:X3,quant:cnni]]
```

FIGURE 7.2 The FB generated by the syntactic and semantic analyser for the sentence *rAmuDiki kemisTreelO enni maarkulu vaccAyi?*

The FB with all its details, as shown in figure 7.2, is the input to the restructuring transformation. Out of all the features present in the FB, the features "sem", "word_loc", "post", and "agr" are of primary importance to the restructuring transformations. The "sem" feature value is initially stripped of all syntactic information except for the "wordjoc" and "post" feature values. Appropriate logical quantifiers, based on the number as indicated in the "agr" feature of the nominal are next inserted into the FB. In addition, along with the quantifiers, the value of their "wordjoc" feature and the value of the "post" feature of the nominals they quantify, are also indicated. All singular unquantified nominals are treated as existentially quantified. As explained in Chapter 3, bare plural nouns occurring as subjects or modified by a relative clause are given the logical quantifier "all" and other bare plural nouns are given the logical quantifier "some". Occurrence of bare plurals in database querying is rare. Question indicating words like *evaru* (who), *ekkaDa* (where), and *eppuDu* (when), are converted into the logical quantifier "wh". The question word *enni* (how many) needs some special processing. If it refers to a number it should be translated into the simple

"wh" quantifier, otherwise it should translated into the quantifier "enni". The rationale for dichotomy is that the lexical quantifier "enni" as in

q(enni, X, p(X), q(X))

normally necessitates counting the number of cases where p(X) & q(X) is true. However, when X represents a numeric field, the counting process is not necessary, as the value of X itself gives the answer. For example, the query

రాముడు ఎన్ని సబ్జక్టులలో పాస్ అయ్యాడు?
rAmuDu enni sabjaKTlalO pAs ayyADu?
In how many subjects did Rama pass ?

requires counting all the instances where the database query

student_marks(,rAma,_subjects, M) & M> 35

is true. Whereas, the query

రాముడికి కెమిస్ట్రీలో ఎన్ని మార్కులు వచ్చాయి ?
rAmuDiki kemisTreelO enni mArkulu vaccAyi ?
How many marks did Rama get in Chemistry?

requires just getting a value for M from the database using the database query

student_marks(_,rAma,_,M)

Negation in the query is treated as the predicate "not". For example, the query

రాముడు ఇంటర్ చదవలేదా?
rAmuDu iMTer cadavalEdA?
Did rama not study Inter ?

is restructured into "not" of " rAmuDu iMTer cadivADA".

As already indicated, the "word_loc" and the "post" information corresponding to every quantifier is also made available with the quantifier in the restructured FB. The restructured FB for the example sentence is shown in figure 7.3.

sem: [reln: vacc,
samdarbham: q([ex,5,none], XO, samdarbham(X0)),

vidyArthi: q([ex,1,ki],X1, human(X) & sem: [reln: name,id:
X1,string:rama]),
vishayam: q([ex,2,10],X2, subject(X2) & sem: [reln: name,id:X2,
string: kemisTree]),
maark: q([wh,3,none],X3, maark(X3))
]

FIGURE 7.3 The stripped FB for *rAmuDiki kemisTreelO enni maarkulu vaccAyi?*

Note that the event/state variable takes the verb's word_loc value in the above restructured FB. From the above restructured FB, the following predicate form of the original FB is obtained:

vacc(q([ex,5,none], XO, samdarbham(X0)),
q([ex,1,ki], X1, human(X1) & name(X1,rAma)),
q([ex,2,10], X2, subject(X2) & name(X2,kemisTree)),
q([wh,3,none], X3, maark(X3)))

The predicate form of the restructured FB is called the Intermediate Logical Form or intermediate LF. This is then input to the quantifier scoping algorithm. The details of this next stage are presented below.

7.4 Scoping Algorithm

An intermediate LF with all the quantifiers is passed as an input string to the quantifier scoping algorithm. The output, called final LF, is a string that has all the quantifiers scoped appropriately. To properly scope the quantifiers, they need to be extracted out of the predicate they are contained in. They have to be moved over earlier quantifiers if their scope is thought of to be wider than the earlier quantifiers. This is known as raising the quantifiers. The quantifier scoping algorithm is designed to maintain the lexical order of the quantifiers where possible, because in Telugu lexical order of quantifier quite often tends to be the real order of the quantifiers. It is only under certain conditions that the lexical order is not the right order. The algorithm identifies such exceptions, and changes the quantifier scopes accordingly. The algorithm presented here is a more detailed and an enhanced version of the one presented in [Ravi91b].

The scoping related behaviour of quantifiers is idiosyncratic to the language. The quantifiers in Telugu behave differently from English quantifiers. For example quantifiers belonging to a relative clause in English do not outscope the quantifiers of the antecedent [Wood78,

Mora88]. Whereas in Telugu they do. Similarly the Telugu quantifiers such as konni (few), prati (each), anni (every) behave differently than the corresponding English quantifiers. This prompted the author to develop a quantifier scoping algorithm for Telugu. To the best knowledge of the author, there is no systematic study of quantification in Telugu, either by Linguists or by Computer Scientists. Hence, such a study was conducted by the author to analyse the preferences of native Telugu speakers in understanding sentences with multiple quantifiers. There were two main observations of the study.

The first was that the behaviour of the scopes of quantifiers in Telugu can change drastically depending on the postfixes of the corresponding nominals. Especially, subject and non-subject distinction is noticeable. For example, in the following sentence,

అన్ని లెక్కలూ కొందరు చేసారు
anni lekkalu koMdaru cEsAru
all problems some people solved
Some people solved all the problems

the qunatifier, embedded in the word *koMdaru* (some people) which is the subject of the sentence, has wider scope compared to *anni* (all) which quantifies the object of the sentence. Whereas in the following sentence, even though the quantifiers *anni* (all) and *konni* (some) are in the same relative order as in the above sentence, their scope is reversed.

అందరూ కొన్ని లెక్కలు చేసారు
aMdaru konni lekkalu cEsAru
everybody some problems solved
Everybody solved some problems.

The second observation was that most of the time the surface order of the quantifiers is preserved in the logical forms. This implies that the algorithm for proper quantification in Telugu only needs to act on those cases where surface order cannot give the correct scoping. In the following paragraphs, an algorithm respecting the above observations is given. Its implementation in TELANGANA is given in the subsequent section.

In TELANGANA, the different quantifiers of Telugu are categorized into 5 groups as follows.

E, evaru, enni, evvaroo,evarinee etc.	group 1
numeric *(oka, reMDu, mooDu* etc.)	group 2
prati	group 3
kondaru, konni, konta, cAlA	group 4

anni, andaru, anni, okkokka group 5

The quantifiers and the operator *lEdu* in **every** sentence can be arranged as a list of lists. The inner most list corresponds to the most deeply embedded sentence. Relative clauses are considered to be independent sentences with respect to quantifier scope assignment and hence the quantifiers in them are represented in an independent list. The quantifier list corresponding to the relative clause is put after the quantifier of the anatecedent of the relative clause. As an example, for the following sentence,

prati upAdhyAyuDu aMdaru vidyArthulu cESina konni tappulu pErkonnADu
Every teacher mentioned a few mistakes which all the students made

the corresponding quantifier list would be,

[prati, konni, [aMdaru]]

For the following sentence,

koMta maMdi konni lekkalu tvaragA cEya-lEdu.
some people some problems quickly did -not
some people did not solve some problems quickly

the quantifer list would be,

[koMta, konni, lEdu]

The algorithm works from outermost list to the innermost list. While processing quantifiers at any given level, the quantifiers at a deeper level are ignored.

Step (1): Form a list of all the quantifiers, and *lEdu* in the sentence. Let i= 1;

while (i .lessthan. length of list) do

Step (2): Examine the **i-th** and **(i+1)-th element** of the list

RULE 1— If the i-th element belongs to group 3 and it has not been exchanged earlier, and the (i+l)-th is *lEdu,* then exchange the two elements.

RULE 2-- If the i-th element belongs to group 2, 4 or 5, and the (i+l)th element

RULE 2— If the i-th element belongs to group **2, 4** or 5, and the (i+l)th dement belongs to group 1, then exchange the two elements

RULE 3— If the i-th element belongs to group 5, and it has not been exchanged with any element earlier, and the (i+l)th element is *lEdu,* then exchange them.

RULE 4~ If the i-th element belongs to group 5 and is in accusative case, and the (i+l)-th element belongs to group 4, then exchange the elements.

RULE 5~ If the i-th element belongs to group 5 and is in nominative case, and the (i+l)-th element belongs to group 2, and the tail of the list contains *lEdu,* then exchange the elements.

Step(3): i=i+l;

End while;

It may be observed from the above algorithm that quantifiers in relative clauses do not outscope the quantifiers of the antecedent. A relative clause can only contribute to the restriction of the quantifier and hence can not normally out scope the quantifier of the antecedent. But this is not the case always. In some situations, people tend to give a wider scope to the quantifier *konni/prati* in the relative clause, over the quantifier *anni, konni* and *prati* of the antecedent. This appears to happen when *konniiprati* occur as a quantifier to a noun which acts as a subject in the relative clause. The following sentence provides an example of such a situation.

కొందరు న్యాయమూర్తులు కొన్ని కంపెనీలు పెట్టిన అన్ని దరఖాస్తులని పరిశీలించారు
koMdaru nyAyamoorthulu konni kaMpaneelupeTTinaa anni darakhAstulani pariseeliMcAru

Some judged inspected all the petitions which some companies made

Under such circumstances, at the time of restructuring the logical form of the sentence, the raised quantifier of the relative clause is given wider scope than the quantifier of its antecedent.

A similar situation arises while analysing donkey sentences[3] [Karap81]. While assigning scope to quantfiers in donkey sentences, it becomes mandatory, on logical grounds, to give wider scope to the quantifiers in the realtive clause than to the quantifiers of the antecedent. Further, at times the quantifiers themselves may have to be changed. Such situations, wherein the quantifiers are needed to be changed are ignored by not changing the quantifiers in the present treatment, even though the quantifiers are outscoped. This does not lead to unsound results but leads to some nonintuitive quantifier scoping in the rare event of such queries being posed to the database. The mechanisms used for rearranging quantifiers are given in the following section.

An important observation to be made is that the quantifier inside a postposition (PP) clause modifying a nominal automatically gets a higher scope than the quantifiers of the nominal, because in Telugu PP modifiers appear before the nominal they modify. In English PPs occur after the NPs, hence special preference rules [Mora88, Vanl87, Wood78j are needed to handle quantifiers in PPs. A few examples will make the operation of the algorithm clear. In the following example,

అందరు విద్యార్థులు పాస్ అయిన ఒక సబ్జెక్ట్ తెలుపుము
aMdaru vidyArthulu pass ayina oka sabjekT telupumu
Every student pass rel-marker one subject inform
Inform (the speaker) the subject in which every student passed

The quantifier list for the above sentence is

[oka, [aMdaru]]

which when expanded back to LF (see section 7.4.2) gives

q(ex,X,sabjeKT(X) & q(all,S, vidyArthi(S), pass(S,X)), telup(X))

In the following sentence,

[3]For example
Every farmer who owns a donkey, beats it.
q(all, X, donkey(X), (all, F, former(F)& owns(F,X), beats(F,X)))
The quantifier "a" in the relative clause gets a wider scope and also gets modified to "all'
One can get away by not modifying the quantifier "a" also,
q(ex, X, donkey(X), (all, F, former(F)& owns(F,X), beats(F,X)))
but this is not the intutive meaning of the sentence.

అందరు విద్యార్థులు ఏసబ్జెక్టులో తప్పారు ?

aMdaru vidyArthulu EsabjekTulO tappAru *?*

Every student which subject failed show

Show the subject every student failed

the quantifier list is

[aMdaru, E]

which becomes owing to RULE 2 in step (2) above,

[E,aMdaru]

yielding the following LF

q(wh,X,sabjekT(X),q(all,S,vidyArthi(S), tapp(S,X)))

In the following example, no rules are applicable, hence there is no exchange of quantifier scope.

ప్రతి విద్యార్థి ఏసబ్జెక్టులో తప్పాడో చూపుము

prati vidyArthi EsabjekTulO tappADO coopumu

Every student which subject failed

Show for every student, the subject he failed in

Its LF, is

q(all, X, vidyArthi(X), q(wh, Y, sabjekT(Y), coop(failed(X,Y)))

In the following sentence,

అందరు విద్యార్థులు ఏసబ్జెక్టులలో తప్పలేదో చూపుము

aMdaru vidyArthulu EsabjekTlO *tappalEdO coopumu*

Every student which subject did not fail show

Show the subject in which not all students failed

with the quantifier list

[aMdaru, E, lEdu]

owing to RULE 2, *aMdaru* and *E* get exchanged. No rule is further applicable, hence *aMdaru* and *lEdu* do not get exchanged. Hence, the LF of the above sentence is

q(wh, X, sabjekT(X), q(all, V, vidyArthi(V), not(fail(V,X))))

Whereas in the following sentence, where aMdaru and lEdu occur in the same order as in the above sentence, the quantifiers are exchanged due to RULE 3.

రాముడు అన్ని లెక్కలూ చేయలేదు

rAmuDu anni lekkalu cEyalEdu

Rama all problems not solved

Rama did not solve all the problems

The LF of the above sentence is

not(q(all,X, lekkalu(X), cEy(rama,X)))

7.4.1 Implementation of the Quantifier Scoping Algorithm

The means of implementing the above quantifier scoping algorithm is obscure due to the presence of q-terms. Hence, the implementation of the algorithm and its foundations in the notions of quantifier stores [Coop83], is explained here briefly.

The scoping algorithm can be understood as a non deterministic program, pull(ILF,S,P), that lakes the intermediate LF, ILF, as input and returns a quantifier store, S, and the scoped logical form, P. While pulling the quantifiers from the predicate M, the occurrences of q-terms in M are replaced by the variables of the q-terms. The q-terms are λ-abstracted to contain a new argument each. A predicate whose quantified arguments are replaced by the corresponding variables is called a kernel. The scoped logical form P results from applying some subset R of the set Q of q-terms in the ILF to the kernel of outer most predicate, M, in the ILF. S is the subset of q-terms, equal to Q-R, which is not yet applied to the kernel predicate.

For example, the kernel of the following ILF

has (q(ex, XO, event(XO)),
q(all,Xl,man(Xl),
q(some,X2,desire(X2))

which has three quantifiers, is

has(X0,Xl,X2)

and the abstracted q-terms are

X?. q(ex, XO, event(X0),P), λP. q(all,Xl, man(X1),P), and
λP. q(some,X2,desire(X2),P)

Let the quantifier store, S, consist of only one out of the three quantifiers. Let that quantifier in store be

X?. q(some, X2, desire(X2),P)

Then the two remaining quantifiers

λP. q(ex,X0,state(X0),P) and *XP*. q(some,X2,desire(X2),P)

are applied to the kernel, so the P of the ILF is

q(ex, XO, state(XO), q(all, X2,desire(X2), has(X0,X1,X2)))

The quantifier store concept presented here extends the quantifier store proposed by Cooper [Coop83] and can give, just as Cooper's scheme does, all the LFs that are legal with respect to the given set of quantifiers present in ILF. Out of this set of possibilities, the one permitted by the quantifier scoping algorithm is emitted. Logically equivalent but scope wise different combinations are considered equivalent and are not treated differently. The "pull" procedure in conjunction with another procedure, apply(S,M,WFF), delivers the properly quantified formula given a FB. The procedure apply(S,M,WFF) is used to get the resultant, WFF, of applying a quantifier store S to a matrix predicate M.

The procedure "pull" is defined inductively on its arguments. Each argument is pulled out to get the corresponding quantifier store, S, and the scoped LF, P. The stores and the scoped LFs of all the arguments are accumulated to give the store of the predicate and the kernel of the predicate. The store of the entire predicate is the union of all the stores of its arguments. The quantifier store thus obtained is then reshuffled, subject to the rules given in step(2) of the quantifier scoping algorithm, to get the correct order of quantifiers.

The pull procedure has subcases based on the type of its arguments. Predicates are classified into two classes. Intensional and normal. Verbs which do not expect sentential complements are normal, like *veLl* (go), *tin* (eat), and *vanD* (cook). Verbs which have *aakaamksha* for sentential complements are intensional verbs. For example, *koor* (want), *ishTapaD* (like) and *cepp* (tell or say) and intensional verbs. Generally intensional verbs are optionally opaque with respect to the argument which represents the sentential complement.

In the case of normal verbs, when a quantifier is pulled out of the predicate, the quantifier leaves its variable in its position in the predicate. For example, if the quantifier q1 is pulled out in the following unscoped logical form,

p(_, q(q1,X,r(X)), _,_)

one gets the following kernel

p(_,X,___)

and the store would contain

λP. q(q1,X, r(X),P)

which, in Prolog, can be represented as

P^q(q1,X,r(X),P)

The variable, P, in the above formula ranges over predicates and is intended to capture the predicate of the matrix sentence at the time of applying the store to the kernel. By accumulating all the quantifiers of the arguments in the store, then rearranging the store, and applying the quantifiers to the kernel, one gets the final properly quantified LF.

If the verb, or more generally an operator, is opaque with respect to the argument that is being pulled out of its body, then there are two possibilities. The first one is that the quantifiers in the argument are not given a wider scope than the operator itself. This gives a *de-dicto* reading. If the quantifiers are allowed to scope over the operator, then it leads to the *de-re* reading of the sentence. The scoping algorithm must allow these two readings. The default order TELANGANA uses leads to the *de-re* reading. While giving wider scope to the embedded quantifier in intensional sentences, special care must be taken while pulling out the embedded quantifier. In the following unscoped intensional predicate p with sentential complement s,

p(_,s(q(ql,X,r(X))),_,_)

when q1 is pulled out, one gets

p(_,s(X),_,_)

as the kernel and

λP. q(q1,X, r(X),P)

as the element in store.

For example, the following unscoped LF

like(rama, q(ex, A, apple(A), eat(rama, A)))

would become,

q(ex, A, apple(A), like(rama, eat(rama, A)))

after proper scoping of the quantifiers.

The third subcase of the pull procedure is invoked when a quantifier is pulled out of a restriction of another quantifier. This case arises when quantifiers present in a relative clause or a postposition phrase need to be out scoped relative to the quantifier of the nominal they modify. Very rarely, the quantifiers in the relative clause get a wider scope than the modified nominal. Conversly, the quantifier embedded in a propositional always gets a wider scope than the quantifiers of the modified nominal. For example, in the following unscoped predicate

pass(q(all, X, student(S) & in(S, q(ex, C, class(C)))))

if the quantifier "ex" is given a wider scope then one should get

q(ex, C, class(C), q(all,X, student(X)& in(S,C), pass(X)))

In a more general setting, in the following ILF,

p(q(ql, X, rl(X) & s(X, q(q2, Y, r2(Y))))

when the quantifier q2 is to be given a wider scope than q1, the resultant scoped LF must be

q(q2,Y, r2(Y), q(q1, X, rl(X)&s(X,Y),p(X))

The kernel of the above kind of predicate is as usual

P(X)

but the store should have more complicated (than earlier two subcases of pull) entries in order to capture the predicate of the relative clause in addition to the predicate of the matrix sentence. For this purpose an island, "isl", construct which lets one generalise over both the restriction and body of a quantifier is used as follows,

isl([λP. q(q2,Y,r2(Y),P), λPλQ. q(ql,X, rl(X)&P, Q)], s(X,Y))

The island operator does the "book keeping" necessary for giving different scope to the quantifiers. For example, the quantifier ql can be either given a wider scope than q2 by removing it from the "isl" or a narrower scope than q2 by letting it remain in the "isl". The island operator "isl" is defined so that, all the quantifiers in it are first applied to its second operand, s(X,Y) in the above example. The second argument encodes to the kernel corresponding to the predicate of the relative clause or the postpositional phrase. After that they can be applied to the kernel of the sentence. The island operator lets one handle the case of donkey sentences, where due to logical requirements a quantifier inside a relative clause needs to be given a wider scope than the modified nominal.

7.5 Query Translation

Logical forms are generated from predicates which are predominantly linguistically motivated. The purpose of the query translation is to translate such domain independent logical forms into database queries which reflect the database structure. This translation involves

- mapping logical form quantifiers to database query operators,
- mapping predicates and their arguments in the logical form to database relations, fields, and values, and
- determining the information to be displayed in reply to a query.

The translation process can also involve optimising the final database query to remove redundant sortal constraints or accesses to database relations. Redundancies arise due to the fact that different predicates in the logical form for a given query may well map to the same database relation using different but not mutually exclusive fields and values.

The translator can be viewed as a term rewriting system that takes the scoped logical form and rewrites it into a simpler form preserving the logical equivalence under the *closed-world assumption* [Reit78]. The transformed query will produce the same answers as the initial input logical form when evaluated against the database.

7.5.1 Mapping Quantifiers to Query Operators

The first step in translation involves mapping the quantifiers in the logical form to database operators. The relational database, in this case being in Prolog, has very few operators: "," and "not". The "," operator of Prolog encodes the logical "and" operator. The "not" operator encodes negation as failure operation. Apart from these two operators, a few more operators like "number_of", "many" and "few" are required to form interesting queries. The operator "number_of(p(X))" gives the number of instances of the given predicate p(X) that are true. It is useful in answering questions like

In how many subjects did Rama pass?

The operator "many(p(X),q(X))" tests if the number of different values of X for which p(X) & q(X) is true is more than half of the number of different values of X for which p(X) alone is true. This operator is useful in translating the quantifier "many" in the following query

Did many students fail in Physics.

The meaning of the operator "few" can be similarly understood.

A query in Prolog is implicitly existentially quantified. Hence the quantifier "ex" in a logical form can be ignored. The quantifier "all" is translated into double negation, if it is embedded in a "yn" (yes-no) operator, otherwise it is translated into the setof operator [Cloc81] of Prolog. That is all(P,Q) is translated into not(P, not(Q)) of Prolog or setof(X,P(X),Y). The quantifier "wh" is ignored, as it is implied in the query operation of Prolog.

7.5.2 Mapping Predicates to Database Relations

As discussed above, the information described in a given sentence may be represented in different ways in different databases. For example, the information pertaining to the college, in the question

రాముడు ఇంటర్ ఏ కాలేజిలో చదివాడు ?
rAmuDu iMTer E kAlEjilO cadivADu 7
In which college did Rama study Inter ?

is represented in the sample database in the **college_name** field of the database relation student_record. In some other database it could have been encoded in the institution field of some relation **inter_record_1990** which contains names of the students who passed in Intermediate exams in 1990 and the colleges they studied in. Not only this, in a given database, the same predicate of natural language may be mapped into different relations depending on the type of arguments it takes. For example, with respect to the sample database, the predicate *tapp* (fail) in the following question

రాముడు ఏకోర్సులో తప్పాడు ?
rAmuDu EkOrsulO tappADu! --7.1
In which course did Rama fail?

should be translated into a query on the database relation student_record. Whereas in the following question,

రాముడు ఏసబ్జెక్టులో తప్పాడు ?
rAmuDu EsabjekTlO tappADu? -12
In which subject did Rama fail?

the same predicate *tapp* (fail) should be translated into a query on the database relation **student_marks**. This is because the relation student_record contains information about

students and the courses they studied. Whereas the relation **student_marks** contains information about the subjects which students had studied and the marks they obtained.

In effect, there can be no unique way to encode information in a database. In addition, the relationship between natural language predicates and database relations is idiosyncratic. The query translator of TELANGANA uses information present in the predicate definitions of a predicate by name, dbtrans, to map predicates into database queries. The predicate definitions specify how predicates appearing in logical form are to be mapped onto database relations. They also specify the sortal constraints that need to be satisfied to affect the above mapping. A predicate definition has the following form

dbtrans([P_1,P_2,P_3,... P_n], R) :- S_1,S_2, S3,... S_m.

In the above P_1, P_2 , and P_n are predicates of natural language which express the contents of a database relation R. S_1, S_2 and S_m are sortal constraints on the fields of the relation P. For example, the definition associated with the relation **student_record** would be

```
dbtrans([vacc(E1,S1,CRS1,GRAD1),time(E1,YR),
         caduv(E2,S1,CRS1,COL1), time(E2,YR), name(S1,N1)],
         student_record(N,CRS,YR,GRAD,COL)):-
S1=(S:human), CRS1=(CRS:taragati), GRAD1=(GRAD:grade),
COL1=(COL:kaLASAla), E1 =(_:vacc),E2=(_:caduv).
```

The above predicate definition tells that, the information in the tuple student_record(N,CRS,YR,GRAD,COL) belonging to the database relation **student_record** encodes the meaning present in the sentences,

> A student S1 whose name is N got the grade GRAD. He studied the course CRS1 in COL1 college in the year YR,

provided the variables S1, CRS1, GRAD1, and COL1 have the sortal type human, taragati (course), grade, and kaLASAla (college) respectively. The above definition is complex as the relation **student_record** captures two different kinds of information. The information in the relation **student_marks** can be expressed by a simple predicate *caduv* (study), as follows.

```
dbtrans([caduv(E,S,SBJ), name(S,N)], student_marks(N,SBJ1,_)):-
        S=(S1:human),SBJ=(SBJ1:vishayam).
```

The information in the database relation **lecturer_record** is linked to the predicate *nErp* (teach) as follows

dbtrans([nErp(E,S,SBJ),time(E,YR),name(S1,N),lO(SBJ,CRS)],
lecturer_record(N,SBJ1,CRS 1,YR)):-
S=(S 1 :human), SBJ=(SBJ1:vishayam),CRS=(CRS1 :taragati),E=(_:nErp).

The predicates like *tapp* (fail) can be given the following "dbtrans" predicate definitions depending on the sortal types of the arguments.

dbtrans([tapp(E,S,CRS), name(S,N)],student_record(N,CRS1,_,f,_)):-
S=(S1 :human), CRS=(CRS1 :taragati),E=(_:tapp).
dbtrans([tapp(E,S,SUBJ), name(S,N)],(student_marks(N,SUBJ1,M)&(M<35))):-
S=(Sl:human), SUBJ=(SUBJ1:vishayam),E=(_:tapp).

In the above dbtrans definitions, the same predicates *tapp* (fail), is defined in terms of two different database relations based on the sortal types of the third argument. Thus, it is possible to handle queries such as 7.1 and 7.2, without any ambiguities. All the above predicates were in some way directly linked to single database relations. At times seemingly simple queries can lead to accessing multiple database relations, as in

ఉదయ కాణాద విద్యార్థి?
udaya kANAda vidyArtha? —7.3
Is Udaya Kanada's student?

Answering the above query necessitates consulting all of the database relations, **student_record**, student_marks and lecturer_record. The appropriate equivalent query is as follows

student_record(uday,_,YR,_,J & student_marks(uday,SUBJ,J &
lecturer_record(kANAda,SBJ,_,YR)

The predicate of the query is vidyArthi(X,Y) where X is a student and Y is a lecturer. It is common sense knowledge that, X is a student of Y if X studied some course C in year YR, and also X studied some subject S in that course, and Y taught that subject in that course that year YR. This common sense knowledge is required to translate the above query 7.3. Common sense knowledge of this nature is provided through a predicate by name ckr as follows

ckr(vidyArthi(X,Y), [caduv(E1,X,CRS), time(El,YR), caduv(E2,X,SUBJ),
nErp(E2,Y,SUBJ), time(E2,YR), lO(SBJ,CRS)]).

The predicate "ckr" can consist of many other types of common sense relations such as transitive relationships between various entities, or common sense notions such as

ownership means possession, outside is the opposite of inside, and tall is the opposite of short More the common sense information of this type, the more difficult it becomes to be sure of the outcome of the translation. Hence, use common sense knowledge of the above kind is restricted in TELANGANA to the minimum level possible under practical constraints. In the above fashion, the translation procedure uses the predicate definitions in the "dbtrans" and "ckr" predicates uniformly to translate both types of queries; those which require accessing a single database relation and those which require accessing multiple database relations simultaneously.

7.5.3 Determining Information to be Displayed

The notion of an appropriate answer to a query has received considerable attention both in the database community and in the NLP community [Kapl83]. However, there is no unanimously agreed notion of best answers to queries. In TELANGANA, a simple heuristic that was used in TEAM [Gros87] is used to display the answers to the queries. While converting logical forms into the corresponding database queries, the variables which outscope the quantifier "wh" are taken to be the variables that should be output in response to the query. For example, in response to the following query

ప్రతి విద్యార్థి ఏ కోర్సు చేశాడో చూపుము
prati vidyArthi EkOrsu cESADO coopumu
Show for each student the course he did
q(all,X, vidyArthi(X),
  q(wh,K, taragati(K),
    q(ex, E, samdarbham(E), cEY(E,X,K))))

the appropriate response would be to show a table of students and the courses they took. Whereas, the most appropriate response for the following query

ఏకోర్సు ప్రతి విద్యార్థి చేశాడో చూపుము
EkOrsu prati vidyArthi cESADO coopumu
Show the course which every student did
  q(wh,K, taragati(K),
    q(all,X, vidyArthi(X),
      q(ex, E, samdarbham(E), cEY(E,X,K))))

would be to show a list of courses which every student undertook. The difference in what needs to be shown is connected to the quantifiers which out scope the "wh" quantifier in the above queries. Hence this simple heuristic was used in **TELANGANA.**

7.6 Examples of TELANGANA in action

To give an over all picture of the processing of queries in TELANGANA a few example sessions with TELANGANA are given below with comments. In the following examples, non-verbal relations are shown in the predicate form instead of the **feature:value** form ease of understanding. TELANGANA gives the prompt **"TEL>"** for input and output.

Consider giving the following simple sentence as input

రాముడు నిజాంలో bsc చదివాడా
raamuDu nijAMlO BSC cadivADA ?
Did Rama study B.Sc in Nijam?

The input is given without the question mark as follows.

TEL> raamuDu nijAMlO bsc cadivADA

The morphological analyser analyses the input, and passes on the following list to the syntactic and semantic analyser.

TEL>[raamuDu, nijAM+lO, bsc, caduv+aaDu+aa]

Syntactic and semantic analyser analyses this sentence. It finds in the lexicon the first definition of caduv (see appendix B) which has *aakaamksha* for three entities, an entity whose sortal category is animate entity, an entity whose sortal category is "degree", and another object whose sortal category is **"kaLASAla".** The sortal categories are represented in nemonic form for brevity. In reality they are long strings encapsulating the full path from root to the category in question in the sortal category tree given in Chapter 3. The suffix needed with each entity is also indicated as part of *aakaamksha* information. TELANGANA emits the following FB as the output of its syntactic and semantic analysis.

```
TEL>  [aak:l: [],var:_11AC,word_loc:4,  sentype:[none,ynq], used:_0F98,
 yog:[pos:v,isa:mtrans,agr:[[m,s,3],[none,_0400]]], post:aaDu,
 sem:[ reln:cadu,
   samdarbham:_11AC,
   vidyArthi:[aak:n : [],var:_0390,word_loc:1, aak_isa:anim,post:subj,
```

```
    sentype:[none,none],used:yes,
    yog:[pos:n,isa:hum,agr:[[m,s,3],[none,_0400]]],
    sem:name(_0390,raamuDu)],
kaLASAla:[aak:n : [},var:_081C,word_loc:2, aak_isa:kaLASAla,
    sentype:[none,none), used:yes,post:lO,
    yog:[pos:n,isa:kaLASAla,agr:[[n,s,3],[_0884,none]]],
    sem:name(_081C,nijAm)],
taragati:[aak:n : [],var:_0C44,word_loc:3, aak_isa:taragati,
    sentype:[none,none],post:none, used:yes,
     yog:[pos:n,isa:taragati,agr:[[n,s,3],[_0CAC,none]]],
  sem:name(_0C44,bsc)]]]
```

In the above FB, as is the norm with Prolog language outputs, variables are indicated using numeric strings. Notice in the above FB, the sentype indicated with each word. The sentype of the verb indicates "yes-no" question, where as the sentypes of the words are through out [none,none]. It was mentioned in Chapter 5, that sentype can not be modelled using the fool feature of GPSG. In the above FB this aspect is demonstrated.

The above FB is converted into the following intermediate LF[4] and output as

```
TEL>  caduv(q([ex,4,none],X0,samdarnham(X0)),
        q([ex,1,subj],X1, human(X1)& name(X1,raamuDu)),
       q([ex,2,lO],X2,kaLASAla(X2)&name(X2,nijAm)),
       q([ex,3,none],taragati(X3)&name(X3,bsc)))
```

The scoping algorithm realises that there are no explicit Telugu quantifiers in the above query and hence, arranges the "ex" quantifiers in the order they appear in the sentence. The quantified LF for the above is ouput as

```
TEL>q(ex,X1 ,human(X1 )&name(X1 ,raamuDu),
        q(ex,X2,kaLASAla(X2)&name(X2,kaLASAla),
          q(ex,X3,taragati(X3)&name(X3,bsc),
```

[4]Variables have been renamed from their original numeric Prolog names, for the sake of readabilty.

q(ex,X0,samdarbham(X0), caduv(X0,X1 ,X2,X3)))))

The above LF is translated based on the predicate caduv with sortal constraints placed as follows,

caduv(Y0:samdarham, Y1:human, Y2:kaLASAla,Y3:taragati)

and yields the following statement as the database specific query

TEL> **student_record**(raamuDu,bsc,_,_,nijAm)

All the name predicates in LF are ignored based on the common sense knowledge that names can stand in lieu of the object they indicate. The above database query is evaluated by the query evaluator in TELANGANA, and the answer "yes" or "no" is given depending on the presence or absence of the requested data.

Consider another example sentence,

రాముడికి భిసిక్సులో ఎన్ని మార్కులు వచ్చాయి ?
rAmuDiki phisiksulO enni maarkulu vaccAyi ?
How many marks did rama get in Physics?

The first stage of TELANGANA, MA, analyses this sentences and produces the list

TEL> [rAmuDu+ki, phisiks+lO, enni, maarkulu, vacc+aayi]

The syntactic and semantic analyser produces the following FB as the output

```
TEL> [aak:l : [],var:_2010, sentype:[none,q],word_loc:5,
 used:_1E48, for:[ki],post:aayi,
 yog:[pos:v,isa:ptrans,agr:[[n,pl,3],[def,cnt]]],
 sem:[reln:vacc,
    samdarbham:_2010,
    vidyarthi:[aak:n : [],var:_04BC,aak_isa:anim,wordJoc:1,
       sentype:[none,none],
       used:yes,post:ki,
       yog:[pos:n,isa:hum,agr:[[m,s,3],[none,_052C]]],
       sem:name(_04BC,raamuDu)],
    maarkulu:[aak:n : [],var:_0F70,aak_isa:mark,post:subj,
       quant:[
              yog:[ pos:quant, agr:[[n,pl,3],[def,cnt]]],
```

```
            sem:number(_0F70,enni),word_loc:3,
            used:_0ED8,sentype:[none,q]],
        yog:[pos:n,isa:mark,agr:[[n,pl,3],[def,cnt]]],word_loc:4,
        sentype:[_164C,_1654],used:yes,
        sem:maark(_0F70)],
    vishaym:[aak:n : [],var:_0A60,aak_isa:vishayam,word_loc:2,
        sentype:[none,none],used:yes,post:lO,
          yog:[pos:n,isa:vishayam,agr:[[n,s,3],[_0880,none]]],
        sem:name(_0818,phisiks)] ]]
```

Notice in the above FB the sentype information is taken from the sentype information of the quantifier word *enni.* The intermediate LF corresponding to the above FB is

```
TEL>  vacc( q([ex,5,none],X0,samdarnham(X0)),
          q([ex,1,subj],X1, human(X1)& name(X1,raamuDu)),
         q([ex,2,lO],X2,vishayam(X2)&name(X2,phisiks)),
         q([wh,4,none],X3, maark(X3)))
```

Notice that the quantifier *enni* has been translated as "wh", because the filed corresponding to the entity ***maarkulu*** is encoded as a number. The above intermediate LF is scoped. This time the quantifier "wh" gets the widest scope as it corresponds to the Telugu quantifier *enni.* The scoped LF is

```
TEL> q(wh, X3, maark(X3),
          q(ex,X1, human(X1)& name(X1,raamuDu),
             q(ex,X2,vishayam(X2)&name(X2,phisiks),
                q(ex,X0,samdarnham(X0),vacc(X0,X1,X2,X3)))))
```

This LF is translated to

```
TEL>student_marks(raamuDu,phisiks,N)
```

The above being a "which" type of query, as described earlier in section 7.5.3, a note is made by the query translator to display the results of the answer as a table. If the person had written several exams in Physics and has several results in Physics, then all the marks in those results would be displayed by the query evaluator.

Consider another sentence consisting of a relative clause, and its list form as output by the morphological analyser, given below.

రాధ తప్పిన ప్రతి సబ్జక్టులో సీత పాసయిందా ?

raadha tappina prod sabjekTlO seetapaasayiMdA ?

Did Sita pass in every subject which Radha failed?

TEL> [raadha,tapp,ina,prati,sabjekT + lO,seeta,paasavv + iMdi + aa]

The syntactic analyser generates the following FB[5].

```
TEL> [aak:l : [],var:_41EC,  sentype:[none,ynq],word_loc:7, post:iMdi,
 sem:[reln: paasavv,
       samdarbham:_41 EC,
       vidyarthi:[var:_3B24,word_loc:6,sem:name(_3B24,seeta)],
       vishaym:[var:_1 F90,word_loc:5,
              sem:name(_1 54C,_19A8)
              relc:[aak:l : [],var:_0984,word_loc:2,
                     sem:[reln:tapp,samdarbham:_0984,
                            vidyarthi:[var:_042C,word_loc:1,
                                   sem:name(_0384,raadha)],
                            vishayam:[var:_154C,post:relc,sem:_154C],
                     ].
              quant:[ sem:number(_154C,parti),word_loc:4]]
 ]
```

The LF corresponding to the above FB is

```
TEL> q(prati,X3, vishayam(X3)&
                  q(ex,X1,human(X1)&name(X1,raadha),
                     q(ex,X0,samdarbham(X0), tapp(X0,X1,X2)),
          q(ex,X5,human(X5)&name(X5,seeta), q(ex,X4,samdarbham(X4),
                  paasaw(X4,X5,X3))))
```

[5]The FB has been abrdiged, for an easier understanding of the essential features in it.

This LF has two predicates, tapp (to fail) and paasavv (to pass) and the universal quantifier prati (all). Also note that the input sentence is an yes no type of sentence, hence the above query is translated into

TEL>not((student_marks(raadha,SUBJ,M1) & (M1 <35)),
not(student_marks(seeta,SUBJ,M2) & (M2>35)))

The above query is evaluated by the query evaluator, and a "yes" or "no" type answer is given by its processing engine Prolog. In the case of a "no" answer, it would be more useful to give the list of subjects which both Radha failed and Seeta have failed. Providing such co-operative answers necessitates incorporating many additional extra features [Gal87,Kapl83] in the translator and the query evaluator, which are much beyond the scope of this research.

It is hoped that, with the above three examples, and the elaborate discussions about the different implementation aspects of TELANGANA covered in Chapters 4,5, and 6 a good picture of the internal processing of TELANGANA would have been generated.

7.7 Conclusion

This chapter dealt with domain specific issues, such as transforming FBs into intermediate LFs, then scoping the quantifiers in the LFs, and finally translating the LFs into database specific predicates/ commands. In this chapter some preference rules for quantifier scoping for Telugu were presented. These were arrived at after presenting many sentences to native Telugu speakers and observing their preferred reading. As this method is highly empirical, the algorithm may not be foolproof, as is true with any quantifier scoping algorithm [Gros87, Hobb87, Wood78]. However, it does seem to capture the intuitions of native Telugu speakers with respect to quantifier scopes. The approaches to quantification given in [Gros87, Wood78] are highly tuned to English are not appropriate for Telugu. Many of the heuristics used for English do not map well into Telugu. Some examples of TELANGANA in action were also given.

Chapter 8

Conclusion and Future work

8.1 Objectives Met

The main objective in the design and development of TELANGANA was to explore how the concepts of traditional Indian Linguistics can be practically utilised for developing a computer program that can analyse and understand Telugu sentences, specifically in the context of database access. This objective was divided into four specific objectives as detailed in Section 1.10. The ensuing section reviews the extent to which those objectives have been met and highlights the contributions made by the research reported in this thesis.

8.1.1 Formalising traditional grammatical theories

The existing theories of natural language processing do not adequately capture the syntactic and semantic criteria of wellformedness of Telugu and other Indian languages. The traditional linguistic theories propounded in Sanskrit seem to have a better chance of success with respect to Indian languages. These theories proposed several ideas for analysing sentences. Prominent among those were the ideas of *aakaamksha* (expectancy), *yogyata* (ability), *sannidhi* (proximity) and *samarthah* (equi-meaning) propounded by Panini in his book, Ashtaadhyaayi (AST). The first objective of this thesis was to formalise these ideas and notions into a form that is suitable for computational purposes, remaining close in spirit to AST. Among the above four ideas of Panini, the idea of *samarthah* is very intricate and was interpreted in different ways by different grammarians [Maha84, pp 10-18]. In Section 5.2, a complete and computationally appropriate definition of *samarthah* is given, which improves and extends the one given by Mahavir [Maha84]. In Section 3.2.2, the ideas of *aakaamksha, yogyata* and *sannidhi* have been formalised.

In AST, the meta rule, *samarthah padavidhih,* was given as the all pervading rule in sentence formation. The applicability of this rule for the study of Indian languages is reinforced in this thesis by developing a parser which uses *samarthah* between words as the only guiding principle for parsing. Several sentential constructs which hitherto were

never analysed in traditional grammar books of Sanskrit and Telugu using *samarthah* have been analysed in this thesis. The handling of pronoun references has not been linked to the samarthah rule in the traditional grammatical books. The link between samarthah and pronominal reference has been established and explained in Chapter 6. This is probably one of the advantages of formalising a theory.

8.1.2 Operationalizing the theory

In Computer Science, no theory becomes interesting unless it can be put to test on the computer. The computer program, TELANGANA, is the result of Operationalizing the theory of *samarthah.* Operationalizing a theory involves spelling out each and every detail of the theory, and developing solutions to all the implementation and performance problems. More specifically it involves spelling out all the data structure that are needed to capture the knowledge structures crucial to the theory and all the algorithms needed to simulate the theory. The notions of *aakaamksha*, *yogyata*, and *sannidhi* which capture the static aspects of *samarthah* theory, were operationalized as the Feature Bundle (FBs) data structures, which have structure sharing properties. The language MRL has been introduced in Chapter 3 to describe the FBs. The dynamic aspects of *samarthah* theory, which manifest in the context of analysing sentences, have been operationalized using the idea of Functional Application Parsing (FAP) in Section 5.3. It must be appreciated that FBs and FAP together offer one of the ways of Operationalizing the theory of *samarthah.* The idea of FAP forms the backbone of parsing in TELANGANA. FAP relies heavily on the close integration of syntax and semantics.

Using this parsing scheme, TELANGANA, can parse and build the semantic representation for a variety of Telugu sentence structures consisting of, simple sentences, copula sentences (verb-less sentences), relative clauses, comparatives sentences, subordinate sentences, and some amount of intra-sentential anaphora and limited conjunctions. TELANGANA parses all these varied types of sentences in a uniform and a simple way.

The special features of FAP and consequently TELANGANA include,

1. **Close integration of Syntax and Semantics:** *Samarthah* is Syntactico-Semantic in nature. Hence, FAP which uses *samarthah* as its basis is also Syntactico-Semantic in nature, and attains its parsing capabilities due to the close integration between syntax and semantics. FAP interweaves syntactic and semantic analysis to achieve

robustness and performance. Due to the bottom-up nature of FAP, TELANGANA can accept even partially formed sentences, and compute their possible meaning. As Telugu offers very few syntactic clues for analysis (see Section 6.2, and 6.3), a "syntax first" and "semantics next" kind of an approach would have caused excessive backtracking and hence loss in the performance of the parser. The use of sortal constraints in FAP facilitates weeding out semantically illformed possibilities in the early stages of parsing, and hence contributes to the performance of the system.

2. **Comprehensive treatment of postpositional phrases:** One of the major problems in NLP is resolving ambiguities arising out of post(prc)position attachment. This problem is accorded detailed examination in TELANGANA and a heuristic approach to deal with it in the context of Telugu is given in Section 5 .5.4.

3. A novel treatment of relative clauses: The matching of fillers in the matrix sentences with the gaps in relative clauses is done in a novel way, as given in Section 6.2. In FAP, relativising suffixes such as *ina*, and *E*, are viewed as functions which modify the *aakaamksha* of the main verb of the relativized clause. The verb with modified *aakaamksha* behaves similar to adjectives. This makes relative clauses equivalent in behaviour to adjectives. This way of treating relative clauses obviates the necessity for using the SLASH feature of GPSG, and also is in the spirit of Telugu grammars[Kris85]. Telugu grammar books [Kris85] treat relative clauses under the heading of adjectives and adjectival sentences.

4. **Comprehensive treatment of subordinate clauses:** The subordinate clauses of Telugu pose some interesting problems which can not be easily solved using the traditional SLASH feature of GPSG theory. The reasons for this are explained in Section 6.3. A bi-directional mechanism for information flow is needed. Such a mechanism is presented in Section 6.3. Using this mechanism a wide variety of subordinate sentences can be analysed. This mechanism also helps in matching default fillers to gaps.

5. **Comprehensive treatment of comparative sentences:** Computing the semantic content of comparative sentences is a difficult task that taxes the semantic capabilities of a parser. A type of semantic 'type raising' is presented in Section 6.4 which allows a common rule schemata to be used for computing the semantic denotations of different kinds of comparative sentences. Using the idea of rule

schemata, TELANGANA can analyse a wide variety of comparative sentences (including clausal comparatives, adjectival and adverbial comparatives, superlatives), and compute their semantic content.

6. **Resolving anaphoric references:** Anaphoric reference resolution in FAP is based on a new categorization of pronouns, given in Section 6.5, which extends the standard categorization of pronouns currently employed for analysing English, and Japanese. The current categorization is inadequate to handle the different pronouns of Telugu. The methods explained in Section 6.5 make identifying the referents of anaphoric pronouns in Telugu easy.

8.1.3 Developing a Morphological Analyser and a Lexicon

Developing a morphological analyser for a highly inflection and suffixation oriented language like Telugu which, in addition, allows compound word formation, is a challenging task. Especially so, when a machine readable lexicon is not available. The bootstrapping technique detailed in Chapter 4, makes it possible to simultaneously develop a morphological analyser and a lexicon for Telugu. The morphological analyser was developed and tested using a 2,82,000 word corpus. The lexicon currently contains about 11,000 root words and captures information like the part of speech of the word, and the suffixes with which the word can combine. To the best knowledge of the author, this lexicon of root words of Telugu is the first of its kind generated entirely automatically.

8.1.4 Developing a Quantifier Scoping Algorithm for Telugu

There is no published literature on the problems of quantifier scoping in Telugu. The scope related properties of quantifiers of a given language are idiosyncratic to that language. The algorithms and rules presented in English quantifier scoping oriented research do not map well into Telugu. This motivated the development of a new algorithm for assigning correct scopes to Telugu quantifiers. The algorithm and its implementation are presented in Section 7.4. This algorithm assigns proper scope to quantifiers in natural language (NIL) queries, and thus, facilitates translating NL queries into appropriate database queries. In TELANGANA, quantifier scoping information is also used in deciding the type of response to be displayed to the user.

Earlier in this section, it was mentioned that current natural language processing approaches are not suitable for analysing Telugu. In the next section, the reasons for

their unsuitability are presented briefly. Along with that, the manner in which TELANGANA overcomes those inadequacies is also presented. The ensuing discussion also brings out the differences between TELANGANA and other NLP systems.

8.2 Comparison of TELANGANA with Other Approaches to NLP

Some of the approaches given in chapter 2, principled based approach (GB theory) of Chomsky, GPSG/HPSG, Pragmatic based approach of Schank, and Mathematical approach (Categorial Grammar) are superficially similar to those adopted in TELANGANA. The principles embodied in TELANGANA do not put forth an entirely new paradigm of NLP, but use some of the strengths of those approaches and the traditional Indian linguistic approaches, to create a novel solution to the problems of NLP with reference to Telugu. In a way, TELANGANA tries to embody the best of both the worlds. A brief comparison of the principles embodied in TELANGANA with other NLP methodologies would highlight the novelty and differences of FAP. In the ensuing discussion, the word TELANGANA is conveniently used to refer to the principles embodied in TELANGANA.

8.2.1 FAP and GB Theory

The GB [Chom81, Chom86] approach posits the notion of phrase structures as a fundamental property of a language and tries to explain every syntactic and semantic facet of a language using well defined constraints on the phrase structures in the language. Given a sentence, its well formedness is determined by seeing if the phrase structure tree of the sentence satisfies all the phrase structure principle and theories of GB. If not, the sentence is deemed to be ill formed.

In FAP there is no notion of phrase structure. In fact, every attempt was made not use any phrase structure rules or any principles which basically rely on phrase structure. This was done so, because, in the entire Sanskrit literature and Ashtaadhyaayi there do not seem to be any rules which are in any way dependent on any notions of syntax other than some rudimentary classification of words as nouns, verbs, adjectives, particles and adverbs. Hence no notion of phrase structure was used in FAP. This does not seem to handicap TELANGANA. In addition to this, some of the phenomenon handled by FAP can not be treated well if recourse to phrase structure alone is taken. For example, in section 6.5.5, data related to pronominal references in Telugu was presented and it was shown that pronouns become bound to words occurring to their left in a sentence. That

data clearly indicates that typical phrase structure oriented notions such as c-command (Chapter 2) which are phrase structure and subject oriented do not adequately handle pronoun binding in Telugu. Hence it was felt that GB style NLP need not be taken to analyse Telugu.

8.2.2 FAP and GPSG

The GPSG [Gazdar'85] approach is notable for its departure from traditional transformation grammar of English, and in its ability to handle long distance phenomenon by using the idea of SLASH categories. The SLASH category of GPSG allows information about NPs and PPs from a matrix sentence to be passed to a gap in an embedded sentence. However, in Telugu, in addition to the above flow of information, nouns in the embedded sentences can be bound to gaps in the matrix sentence also. This makes the GPSG style SLASH features inadequate for describing Telugu filler-gap dependencies as shown in section 6.3. When this happens, the suffix information is lost. This further renders the SLASH feature of GPSG inadequate for correctly handling Telugu complex sentences.

The subcategorization information in GPSG is presented as a number. This means that subcategorization information is atomic and no operations on it can be done. It also implies that the structure of subcategorization information itself is not of much consequence to the linguistic theory. However, in TELANGANA, the subcategorization information, which is part of the *aakaamksha* of the word, has a rich structure. The theory exploits the structure of the information present in the subcategorization frame. Especially in the treatment of relative clauses, causative sentence formation, passivization and in handling pronouns.

8.2.3 FAP and HPSG

The similarity in HPSG and FAP lies in the use of the notion of structure sharing in representing linguistic knowledge, and in the exploitation of the structure of the subcategorization frame. Both HPSG and FAP are very lexical in nature. After that, all the similarities end. The kind of knowledge captured in the subcategorization frame, the way it is organised and the way it is used in FAP is entirely different from HPSG. The notion of head is very important in HPSG and the parsing in HPSG critically uses the notion of head categories [Prou85]. As has been repeatedly pointed out, the notion of head does not seem to matter much in Telugu and FAP does not use the notion of heads.

Parsing in FAP is headless![1] It proceeds left to right irrespective of whether a head is found or not. Heads are not searched for while parsing. Also, the notion of viewing suffixes and words with *aakaamksha* as functions over root words and words with *yogyata* respectively, makes FAP different from the HPSG style parsing.

Some cases of subordinate and coordinate sentences in Telugu necessitate the use of default values for fillers. Treatment of filler-gap dependencies in HPSG does not seem to cater to such necessities. Relative clauses are handled in HPSG by using the idea of SLASH features, just as in GPSG. As pointed out earlier, FAP does not use the idea of SLASH categories. Published research literature on HPSG [Poll87] does not deal with processing adjuncts, and comparative sentences. In FAP, these phenomenon have been dealt with rather extensively. Ideologically and in the data structures for the lexicon there are some similarities between HPSG and FAP but in implementation and finer details they are entirely different.

8.2.4 **FAP** and Pragmatics Based Approaches

The pragmatics based approaches of Schank [Scha75], DeJong [Dejo79] are similar to FAP in terms of usage of the concept of "words having desire for other words". In the pragmatic based approaches, the parser is driven by the "desire (expectancy)" of the words. Whereas in FAP, *aakaamksha* (desire for other words) is used as a constraint on the wellformedness of sentences. It in no way drives the parser. Further, in pragmatic based approaches, the pragmatic and the episodic content of the sentence is more important than the minor shades in semantic content that accrue due to the presence of many non-episodic words, like determiners, adjectives, and adverbs, in the sentence. Hence, in pragmatic based parsers many words are skipped while parsing, if they are perceived to be immaterial for understanding the theme of the sentence. Whereas, FAP considers no word to be unimportant. Hence it deals with each and every word in a sentence with the same amount of attention as any other word. In addition, many of the pragmatic based approaches can not handle the variety of syntactic phenomenon, like relative clauses, comparative sentences, quantifier analysis (to be dealt in the next chapter) and the subtleties in them at par with FAP.

8.2.5 **FAP and** Categorial Grammar

[1] Pun is intended !

The similarity between Categorial Grammars (CGs) [Stee88] and FAP lies in treating parsing as functional applications. Categorial grammars have a few category combining rules, given briefly in Chapter 2.3.3, for functionally applying one category over the other. These rules would appear to be like the rules in the apply procedure of the parser in TELANGANA. The combinatory rules of CG are basically motivated from mathematical logic. They are a separate set of rules, like any set of context free grammar rules, and are used to analyse a sentence. The combination of categories (feature bundles) in FAP is not motivated by mathematical logic, and happens only when there is *aakaamksha/yogyata* relationship between categories. In other words, the combinatory operations in FAP are driven by the *Aakaamksha/yogyata* relationships, rather than stand alone combinatory rules. This is a major difference in the basic philosophy of the approach itself.

In addition, CG grammars use the notions of type raising to handle relative clauses, and long distance dependencies. Type raising in these grammars, leads to substantial over generalisation of the syntax of the language. The approach taken in FAP does not seem to be reducible to type raising and also does not lead to a proliferation of accepting syntactically illformed sentences[Witt91].

As seen above, functional application parsing in TELANGANA and the basis of that parsing, *samarthah* theory which encompasses the notions of *aakaamksha, yogyata,* and *sannidhi* are significantly different from many other seemingly similar approaches to NLP. An interesting possibility is to see if FAP can usd to parse English sentences. In principle yes. The *aakaamksha* information for words in English would be much more syntactic in nature to capture the highly constrined word order of English. One way to specify *aakaamksha* for English would be to specify the expected *sannidhi* (proximity) of words using much largers FBs which capture the word order constraints of interrogative sentences. This means a lot of syntax is embedded into the FBs. As a matter of cognitive economy, to simplifiy FBs, one would like to take the commanalities in word order out of FBs and posit them as independent entities. This is nothing but phrase structure rules! Hence, the structures embedded in FB can be equated to phrase structure rules in the extreme case of English.

Finally, the question that can crop up is, in what way is TELANGANA limited. How can it be improved to achieve even better and more efficient coverage of the language? Some

suggestions as to future enhancements to TELANGANA are presented in the next section.

8.3 Future Research Directions

TELANGANA is the first of its kind for handling queries in Telugu. It was developed in response to the difficulties faced by Indian language oriented NLP researchers in applying the current NLP approaches to Telugu. It was designed more to test the feasibility of Operationalizing the traditional theories of Indian Linguistics, than to show significant user friendly behaviour or useability at large. Presently it is a prototype for conducting research into Indian linguistic theories and in particular for solving problems associated with Telugu NLP. Hence it has some short comings. Some of the more important ones are described below along with possible approaches to overcome them. Indian language oriented NLP being in its infancy, this research work lays the first few bricks of the foundations of a more exhaustive research program. In the sequel, possible directions for such exhaustive research are also pointed.

8.3.1 Resolution of Lexical Ambiguity.

Selecting the correct sense of a word in a given context is a difficult problem. It can be solved either by carrying all the meanings of the words as the parser progresses, or by simply backtracking or by using context sensitive heuristic to choose the correct sense of the word. If the first method is employed, where all the senses of each word are carried along in the parser till some of those senses are semantically ruled out, it can lead to combinatorial explosion. Currently TELANGANA uses the second option of backtracking. This results in wasted effort when words are highly polysemous. The best option for resolving the problem of lexical ambiguity appears to be the third one. It entails formulating heuristic rules for selecting the word sense based on the co-occurrence of other words and markers in the sentence. Large textual corpora are useful in generating such heuristic rules. The textual corpus used in Chapter 4, for developing the MA can also be used for this purpose. A preliminary investigation in this direction was done by the author as reported in [Ravi91a] in the context of information retrieval systems. The type of rules mentioned there can be easily incorporated into TELANGANA by including them into the lex_sem (Chapter 5) procedure. However, word co-occurrence tests themselves may not suffice as indicated by Dahlgren [Dahl87]. Developing heuristic rules for selecting the contextually correct sense of Telugu words is a very promising line of research to follow up.

8.3.2 Lexical acquisition

The other major area of improvement is to automate the process lexical acquisition. At present, all the FB entries in the lexicon are manually generated. The techniques given in Chapter 4 are able to generate the part of speech information only, whereas the FBs encapsulate a lot more information, including *aakaamksha, yogyata* and *sannidhi* information. That kind of information can be gathered by interacting with the user when new words are used. The TEAM [Gros87] system is an example of such an approach. TEAM learns new words by interactively asking the user many questions on the use of the unknown word in various contexts and thus learns the syntactic and semantic properties of the new word. The design of TELANGANA is currently suitable for this kind of learning. Incorporating the changes needed to affect such a learning mechanism, would require a moderate amount of effort, as the domain dependency in TELANGANA is captured using a few predicates only. The effort is in preparing the appropriate questions to elicit the right kind of information, and making the acquired information available to the parser in terms of FBs and predicates described in Section 5.4 and Section 7.5.2 such as noun_mod, noun_noun, and verb_mod, dbtrans and ckr. This kind of enhancement to TELANGANA is planned for near future.

8.3.3 Using textual corpus to glean more information

Another possible approach, to extracting *aakaamksha,* and *yogyata* type information is to use large textual corpora. Extracting this type of information from corpora poses several problems and at present is limited gathering syntactic subcategorization information only [Bren91]. *Akaamksha* captures both syntactic and semantic information. Hence, gathering *aakaamksha* type information would be much more difficult than mere syntactic subcategorization information. Developing new techniques to automatically gather *aakaamksha, yogyata* and *sannidhi* type information from untagged corpora would be a fruitful direction for future research. TELANGANA would benefit immensely from such information.

8.3.4 Accessing Dictionaries

Dictionaries are a good source of syntactic and sortal information. Making TELANGANA access machine readable dictionaries, when available for Telugu, to handle unknown words would be a very interesting extension to TELANGANA. The allied field of lexicon and dictionary building using automatic computational methods is

an unexplored field in Telugu lexicology. In chapter 4, a bootstrapping method for building better morphological rules and a simple lexicon from rudimentary morphological rules and the textual corpus was presented. The bootstrapping technique explained there may be extendable to a higher level, where one versions of TELANGANA can aid in building a better version of TELANGANA and a corresponding dictionary together.

8.3.5 Handling co-ordinate sentences

Presently, TELANGANA is able to analyse a limited type of conjunctive sentences called implicit co-ordinate constructions. Implicit co-ordinate sentences are those sentences where explicit conjuncts like *mariyu* (and), and *kAni* (or) are not used to form conjunctions. For example, in the following Telugu sentence,

రాముడు ఇంటికి వెళ్ళి సీతతో మాట్లాడాడు
rAmuDu iMTiki veLli seetatO mATlADADu
Rama went home and talked to Seeta

there is no word equivalent to the English conunctive word "and", even though it is used in the English translation. Telugu many times avoids explicit conjunction by taking recourse to the above kind of sentential embedding. TELANGANA can handle conjunctive sentences of the above type, which are formed by sharing a subject, object or an indirect object among different clauses. This type of conjuncts are called VP (verb phrase) conjuncts in English. TELANGANA, at present, can not handle noun-noun conjunction such as the following

రాముడు,సీత పార్కులో ఉన్నారు
rAmuDu seeta pArkulO unnAru
Rama and Seeta are in a park

The reason for not handling even such simple noun-non conjunction is not that it is difficult, but handling co-ordination has been largely ignored in TELANGANA. Providing this type of conjunction for simple sentences alone is easy, but making sure that all cases of conjunction are handled well in the presence of comparative constructs, and anaphora requires additional research.

The architecture of FAP and consequently TELANGANA is well suited for handling conjunctions. The strength of FAP lies in its ability to handle non-constituent conjuncts

(NCC) very well, just like Categorial Grammars [Stee85]. Handling NCC is very difficult for phrase structure oriented approaches, because NCC cuts across phrase boundaries and forms conjuncts which are not phrase structures. Consequently, NCC does not fit well the moulds of phrase structure. FAP is not beset with this trouble, as it does not use the notions of phrase structure. It may be observed that, just as residual *aakaamksha* was used in Section 6.2 for handling relative clauses, it can also be used for conjoining portions of sentence which have similar residual *aakaamksha*. This avoids the problem of type raising present in CGs [Witt91]. This kind of treatment of co-ordinate constructions can open up a wide area of research for analysing conjunction using *aakaamksha* information.

8.3.6 Scope for exploring parallelism

The morphological analyser, MA, in TELANGANA at present works in a sequential fashion. Many parts of the algorithm can be rewritten to work in parallel on the current generation of parallel machines. In the present design, MA sequentially tries several morphological rules before it hits upon the appropriate one to analyse a suffix. Similarly it explores several alternates break points before it guesses the right place to break a compound word. Searching for the right rules to break words into morphemes and root words can benefit enormously if MA is implemented on a parallel machine.

Another avenue for exploiting parallelism is in parsing. Non-configurational languages like Telugu enjoy freedom in word order. Free word order can lead to better parallel parsing, as mutual syntactic dependencies of words are minimal. The parse tree of such languages tend be wide and shallow, as compared to the parse trees of configurational languages like English which tend to be deep and narrow. Due to this, it is conjectured by the author that, free and semi-free word order languages offer better scope for parallelization than phrase structure oriented configurational languages. The truth of this conjecture can only be ascertained by future research.

8.3.7 User friendliness

The input and output to TELANGANA is at present in English script owing to the inability of the current machines and software to handle Telugu script input and output in an integrated way with Prolog, the implementation language of TELANGANA. It would be aesthetically better if TELANGANA interacts with its users in Telugu script. More serious than this problem is that TELANGANA has no language generation capabilities,

and hence, the output generated by TELANGANA is currently very rudimentary. It can only display yes, no, symbolic or numeric outputs. The responses generated by TELANGANA to users' queries are not very cooperative. It would be much better if TELANGANA displays its replies in the form of complete Telugu sentences where necessary. The approaches taken by [Gal87, Kapl83] to generate co-operative answers can be researched in the context Telugu.

8.3.8 Parsing right to left

Telugu is written left to right. Hence in TELANGANA, Telugu sentences are analysed from left to right. However, there is no good reason why Telugu should not be parsed right to left. *Aakaamksha, yogyata* and *sannidhi* related information being highly declarative in nature, it just specifies the "what" aspect of computation and leaves open the "how" aspect of computation. It would be very interesting to implement such a parser and compare the pros and cons of using a right to left parsing strategy. It is conceivable that, such a study could throw light on novel possibilities for mixed strategies of parsing.

8.3.9 Further exploration of traditional theories

TELANGANA at present uses only a few ideas presented in the traditional Indian linguistic literature. There are many other similar ideas propounded for analysing sentences at higher levels such as discourse level, intention level, speech act level, and rhetoric level. Treatises like Vaakyapadiyam [Bhar66], and Dhvanyaaloka [Anan40] present many more concepts and ideas that can be utilised by NLP researchers. The research reported in this thesis is a just a beginning in this direction. It dealt with the analysis of single sentences in isolation. It is envisaged that in the near future discourse level components will be added to TELANGANA based on modelling these traditional Indian theories of discourse and rhetorics.

8.4 Conclusion

All the original goals of this thesis stated in Section 1.10 have been met. The main reasons for the unsuitability of current approaches in understanding Telugu syntax and semantics were presented. Some of the short comings of TELANGANA have beeen pointed out, and possible research approaches to alleviate those problems were presented. More ambitious research projects that can take of from the spring board of TELANGANA have also been pointed. The work reported in this thesis supports the belief that traditional theories of sentence comprehension provide a solid basis for analysing Telugu sentences.

References

[Ajdu35] Ajdukeiwicz K., Die Syntatiche Konnexital, Studia Philosophica 1, pp 1-27, 1935, Transl. in McCall, S., (Ed.), *Polish Logic in 1920-1939,* Oxford, Clarendon, 1967.

[Alsh92] Alshawi, H., *The Core Language Engine,* Cambridge, Massachusetts, MIT Press, 1992.

[Alwa88] Alwar, N., Raman, B. and Mutthukumar, K., "A Natural Language Generator for Hindi", Proc. of the Regional workshop on Computer Processing of Asian Languages (CPAL), Bangkok, 1988, pp 103-108.

[Anan40] Anandavardhana, *Dhvanyaaloka,* Kashi Sanskrit Series 135, Chowkambha Printing Press, Benares, 1940.

[Bare84] Barendregt, H.P., *The Lambda Calculus,* New York, North Holland press, 1984.

[Barh64] Bar-Hillel, Y., *Language and Information: Selected Essays on their Theory and Applications*, Reading, MA, Addison-Wesley, 1964.

[Bart87] Barton, G.E., Berwick, R.C., and Ristad, E.S., *Computational Complexity and Natural Language,* Cambridge, MA, MIT Press, 1987.

[Barw83] Barwise, J., Perry, J., *Situations and Attitudes,* MIT Press, Cambridge, MA, 1983.

[Bear87] Bear, J., and Hobbs, J.r., *Localizing Expression of Ambiguity,* SRI Tech. Note 428, 1987.

[Berg92] Berg, G., A Connectionist Parser with Recursive Sentence Structure and Lexical Disambiguation", AAAI-92, San Jose, California, 1992, pp 32-37.

[Bhar71] Bhartruhari, *Vakyapadiyam,* with English commentary by Subrahmanyam K. A., Bhandarkar Oriental Research Institute, Poona, 1971.

[Bhar90a] Bharati, A. and Sangal, R., "A Karaka Based Approach to Parsing of Indian Languages", Intl. Conf. on Computational Linguistics, COLING-90, Helsienki, 1990, pp203-210.

[Bhar90b] Bharati, A., Chaitanya, V., and Sangal, R.,"A Computational Framework for Indian Languages", Tech. Report TRCS90-100, Dept. of Computer Sc. & Engg., IIT, Kanpur, 1990.

[Bloo33] Bloomfield, L., *Language,* London, George & Unwin Ltd., 1933.

[Boug79] Bougureaev B.K., *Automatic Resolution of Linguistic Ambiguities*, Technical Report No. 11, University of Cambridge Computer Laboratory, Cambridge, 1979.

[Boug85] Bougureaev B.K., Spark Jones K., *A Framework for inference in natural language Front Ends to Databases,* Technical Report 64, computer Laboratory, University of Cambridge, 1985.

[Bren91] Brent, M., Automatic Acquisition of Subcategorization Frames from Unrestricted Text, In *Proceedings of the 29th Annual Meeting of the ACL,* Association for Computational Linguistics, 1991, pp 209-214.

[Bres84] Bresnan, J., (Ed.), *The Mental Representation of GRammatical Relations*, Cambridge, Mass.; MIT Press, 1984.

[Brig86] Briggs R., "Knowledge Representation in Sanskrit and Artificial Intelligence", The AI magazine, Spring, 1985.

[Brow81] Brown, C.P., *A Grammar for the Telugu Language* (In English), second edition, Asian Educational Services, New Delhi, 1981.

[Carn89] Carnegie Group, Inc., *Text Categorization Shell,* Technical Brief, CGI, Aug., 1989.

[Cinn51] Cinnayyasuri, P., *Telugu Vyakaranam* (In Telugu) published by Vaavilla Ramaswami brothers, Madras,1951.

[Chom57] Chomsky, N., *Syntactic Structures,* Berlin, Mouton Publishers, 1957.

[Chom65] Chomsky, N., *Aspects of the Theory of Syntax,* Cambridge, Mass:, MIT Press, 1965.

[Chom81] Chomsky, N., *Lectures in Government and Binding,* Dordrecht, Foris, 1981.

[Chom82] Chomsky, N., *Some Concepts and Consequences of the Theory of Government and Binding,* Cambridge, Massachusets, MIT Press, 1982.

[Chom86] Chomsky, N., *Barriers,* Cambridge, Massachusets, MIT Press, 1986.

[Chur88] Church, K., A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text, In Proc. of the second Conference on Applied Natural Language Processing, Austin, TX, 1988.

[Cloc81] Clocksin, W.F.., and Mellish, C.S., *Programming in Prolog,* Berlin, Spring-Verlag, 1981.

[Coll75] Collins, A.M, and Loftus, E.F., A Spreading Activation Theory of Semantic Processing, *Phychological Review,* Vol 82, 1975, pp 407-428.

[Coop83] Cooper, R., *Quantification and Syntactic Theory,* Dordrecht, D. Reidcl , 1983.

[Dahl86] Dahlgren K., and McDowell, J.P., Kind Types in Knowledge Representation", Proceedings of conf. on Linguistics, COLING-86, 1986, pp 216-221.

[Dahl87] Dahlgren, K., Using Commonsense to Disambiguate Word Senses, Proc. of the II Natural Language Understanding and Logic Programming Conference, Vancouver, Canada, 1987, pp 255-275.

[Dalr87] Dalrymple, M., Kaplan, R.M., Karttunen, L., Koskenniemi, K., Shaio, S. and Wescoat, M., Tools for Morphological Analysis, Stanford University, Rep. No SLI-87-108, 1987.

[Davi67] Davidson, D., The Logical Form of Action Sentences, in N. Rescher, *The Logic of Decision and Action,* Pittsburgh, PA, University of Pittsburgh Press, 1967.

[Davi90] Davis, E., *Representations of Commonsense Knowledge,* San Mateo, CA, Morgan Kaufmann Publishers, 1990.

[Dejo79] DeJong, G.F., Prediction and Substantiation: A New Approach to Natural Language Processing, *Cognitive Science,* Vol 3, Sep., 1983.

[Dejo82] DeJong, G.F., An Overview of the FRUMP System, in Lehncrt, W.G., and Ringle, M.H., *Strategies for Natural Language Processing,* Hillsdale, NJ, Lawrence Erlbaum Associates, 1982, pp 149-176.

[Dixi68] Dixita , *Proudhamanorama,* Nirnaya Sagar Press, Bombay, 1968

[Dowt81] Dowty D.R, Wall. R and Peters S., *An Introduction to Montague Grammar,* D.Reidel, Dordrecht, 1981.

[Dyer83] Dyer, M.G., *Indepth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension,* Cambridge, MA: MIT Press, 1983.

[Fraz79] Frazier, L. and Fodor, J.D., The Sausage Machine: A New Two-stage Parsing Model, *Cognitive Science,* Vol 6, 1979, 191-325.

[Fill68] Fillmore, C.J., "The Case for Case", in Bach and Harms (Eds), *Universals in Linguistic Theory,* Holt, Reinhart & Winston, 1968, pp 1-88.

[Fill85] Fillmore, C.J., Frames and the Semantics of Understanding, *Quaderni di Semantica,* Vol. 6, No. 2, 1985, pp 222-254.

[Gal87] Gal, A., and Minker, J., Informative and Co-operative Answers in Databases, Proc. of the II Natural Language Understanding and Logic Programming Conference, Vancouver, Canada, Aug 1-19, 1987, pp 277-300.

[Gall84] Gallaire, H., Minker, J., and Nicholas, J., Logic and Databases: A Deductive Appproach, *ACM Computing Surveys,* Vol. 16, No. 2, 1984, pp 153-185.

[Gazd85a] Gazdar, G. and Klein, E., Pullum E. and Sag, I., *Generalized Phrase Structure Grammar,* Harvard University Press, Cambridge, massachusets, 1985.

[Gazd85b] Gazdar, G., and Pullum, G.K., Computationally Relevent Properties of Natural Languages and Their Grammars, *New Generation Computers,* Vol 3, No. 3, 1985, pp 273-306.

[Gros82] Grosz B., Hass N., Hendrix, G.G., Hobbs, J., Martin, P., Moore, R., Robinson, J. and Rosenschein, S., "Dialogic: A core natural language processing system" , In Proc. of the Ninth International conference on Computational Linguistics, Prague, Czechoslovakia, 1982.

[Gros87] Grosz B., Appelt D.E., Martin, P., Pereira F.C.N., "TEAM: An Experiment in the Design of Transportable Natural-Langauge Interfaces", *Artificial Intelligence,* Vol 32, No.2, 1987, pp 173-243.

[Gunj87] Gunji, T., *Japanese Phrase Structure Grammar,* D. Reidel Publishers, Tokyo, 1987.

[Hirs83] Hirst, G., *Semantic Interpretation against Ambiguity,* Technical Reprot CS-83-25, Dept. of Computer Science, Brown University, 1983.

[Hirs871 Hirst, G., *Semantic Interpretation and the Resolution of Ambiguity,* Cambridge, Cambridge University Press, 1987.

[Hobb77] Hobbs, J.R., Resolving Pronoun References, reprinted in B. Grosz, S.K. Jones, and B.L. Webber, *Reading in Natural Language Processing,* Boston, Kluwer Academic Press, 1986, pp339-352.

[Hobbs871 Hobbs, J.R., and Shieber, S.M., An ALgorithm for Generating Quantifier Scopings, *Computational Linguistics,* Vol. 13, Jan-June, 1987, pp 47-63.

[Huan85] Huang, X., Machine Translation in the SDCG Formalism, Proc. of the Conf. on Theoretical and Methodological Issues in Machine Translation of Natural Languages, New York, Colgate University, 1985, pp 135-144.

[Hutt79] Huttenlocher, J., and Lui, F., The semantic organization of some simple nouns and verbs, *Journal of Memory and Language,* Vol 18, NO. 2, 1979, pp 141-162.

[Jain'89] Jain, A., and Waibel, A., "Incremental Parsing by Modular Recurrent Connectionist Networks", In D. Touretzky, (Ed.), *Advances in Neural Information Processing Systems* 2., Morgan Kaufmann, San Mateo, California, 1989.

[Josh68] Joshi, S.D., "Patanjali's Vyakarana Mahaabhaashya", (translation of the original Mahaabhaashya in Sanskrit), Publication of the Centre for Advanced Study in Sanskrit, University of Poona, Pune, 1968.

[Kamp81] Kamp, H., A Theory of Truth and Semantic Representation, In J.A.G Groenendijk, T.M.V. Janssen, and M.B.J Stokhof, (Eds.), *Formal methods in the study of Language,* Amsterdam, The Netherlands, Mathematisch Centrum, 1981.

[Kapl83] Kaplan, S.J., Cooperative Response from a Portable Natural Language Database Query System, in M. Brady and R. Berwick, (Eds.), *Computational Models of Discourse,* Cambbridge, Mass:, MIT Press, 1983.

[Kart83] Kartunnen, L. and Wittenberg, K., A two-level Morphological Analysis of English. Texas Linguistic Forum, Vol 22, 1983, pp217-228.

[Katz63] Katz, J., and Fodor, J., The Structure of Semantic Theory, *Language,* Vol 39, 1963, pp 170-210.

[Katz64] Katz, J., and Postal, P.M., *An Integrated Theory of Linguistic Descriptions,* Cambridge, Mass: MIT Press, 1964.

[Kay73] Kay, M., The MIND system, In Rustin (Ed.), *Natural Language Processing,* New York, Algorthimics Press, 1973.

[Kay85! Kay, M., Parsing in Functional Unification Grammar, In D.R. Dowty, L. Karttunen, and A.M. Zwicky, (Eds.), *Natural Language Parsing,* Cambridge, Cambridge University Press, 1985, pp 251-278.

[Khan83j Khan, R., A two-level morphological analysis of Rumanian, Texas Linguistic Forum, Vol 22, 1983,pp253-270.

[Kita91] Kitano, H., Higuchi, T., High Performance Memory based Translation on IXM2 Massively Parallel Associative Memory Processor, proceedings of the conference of AAAI, 1991, pp 149-154.

[Klei80] Klein, E., Semantics for Positive and Comparative Adjectives, *Linguistics and Philosophy,* Vol . 4, No. 1, 1980, pp 1-45.

[Kosk83] Koskenniemi, K., Two-level morphology: a general computational model for word-form recognition and production. Publication No. 11 Helsinki: University of Helsinki, Dept. of General Linguistics, 1983.

[Kris85] Krishnamurthi, B., and Gwynn J.P.L., *A Grammar of Modern Telugu,* Delhi, Oxford University Press, 1985.

[Kuno73] Kuno, S., *The Structure of the Japanese Language,* Cambridge, Mass, MIT Press, 1983.

[Lasn88] Lasnik, H and Uriagereka, J., *A course in GB syntax: Lextures on* Binding and Empty Categories, Cambridge, Mass., MIT Press, 1988.

[Mage90] Magerman, D.M., and Marcus, M.P., Parsing a Natural Language Using Mutual Information Statistics, in Proc. of the AAAI, 1990, pp 984-989.

[Marc80] Marcus, M., *A Theory of Syntactic Recognition Of Natural Language,* Cambridge,Mass., MIT Press, 1980.

[McCo82] McCord M.C., " Using slots and modifiers in logic grammars for natural language", Artificial Intelligence, Vol.8, No. 3, 1982, pp 327-367.

[McCo86] McCord M.C., "Natural Language Processing and Prolog", IBM Research Report RC 11973, IBM T.J.Watson Research centre,York town Heights, New York

[Mell88] Mellish, C.S., Implementing Systemic Classifiaction by Unification, *Am. J. Computational Linguistics,* Vol. 14, No. 1, 1988, pp 40-51.

[Maha84] Mahavir, *Samartha theory of Panini and Sentence Derivation,* Munshiram Manoharlal Publishers, New Delhi, India, 1984.

[Mins75] Minsky, M., A Framework for Representing Knowledge, in P. Winston (Ed.), *The pshychology of Computer Vision,* New York, McGraw-Hill, 1975.

[Mont74a] Montague R., The proper Treatment of Quantification in Ordinary English, In R. Thomason (ed), *Formal Philosophy,* Yale University Press, New Haven, 1974.

[Mont74b] Montague R., English as a Formal Lnguage, In R. Thomason (ed), *Formal Philosophy, Selected Papers of Richard Montague,* Yale University, New Haven,CT, 1974.

[Moor81] Moore R.C., "Problems in Logical Form",In Proc. of 19th Annual meeting of the Associate for Computational Linguistics, Stanford, 1981, pp 117-124.

[Mora88] Moran, D.B., Quantifier Scoping in the SRI Core Language Engine, Proc. of the 26th Annual Meeting of ACL, 1988, pp 33-40.

[Naga86] Nagao, M., Tsuji J., and Nakamura, J., Machine Translation from Japanese to English, Proc. of the IEEE, Vol. 74, No. 7, July 1986, pp 993-1012.

[Nati66] National Research Council, Automatic Language Processing Advisory Committee, Language and Machines: Computers in Translation and Linguistics.

Publication 1416, National Academy of Sciences, National Research Council, Washington, D.C., 1966.

[Newm80] Newmeyer, F., *Linguistic Theory in America: The first quarter-centuary of Transformational Generative Grammar,* New York, Academic press, 1980.

[Parte76] Partee B.H.,(ed), *Montague Grammar,* Academic Press, New York, 1976.

[Peri80] Periera F.C.N., and Warren, D.H.D., Definite Clause Grammars for Natural Analysis: A survey of the formalism and a comparison with augmented transition networks, *Artificial Intelligence,* Vol 13, 1980, pp 231-278.

[Peri83] Periera F.C.N., *Logic for Natural Language Analysis,* SRI International, Tech. Note 275, 1983.

[Poll84] Pollard, C, Generalized Context-Free Grammars, Head Grammars, and Natural Languages, Doctral Dissertation, Stanford University, 1984.

[Poll87] Pollard, C. and Sag, I., Information-Based Syntax and Semantics, CSLI-Tech. Notes: 13, Stanford University, 1987.

[Prou85] Proudian, D., and Pollard, C, Parsing Head-Driven Phrase Structure Grammar, Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics, 1985.

[Pull82] Pullum, G.K., and Gazdar, G., Natural Languages and Context-free Languages, *Linguistics and Philosophy,* vol 4, 1982, pp 471-504.

[Ravi90a] Ravi Shankar, C, *The syntax and semantics of Telugu,* Tech. Report CMC-90-07, CMC Ltd, Secunderabad, India, 1990.

[Ravi90b] Ravishankar, C, The Design of a Logical Form for Telugu, CMC Tech. Report-90-10, CMC Ltd., Hyderabad, India, 1990.

[Ravi91a] Ravishankar, C and Reddy A.S., Informaion Retrieval by Conceptuel Abstraction, CMC Tech. Report-91-2, CMC Ltd., Hyderabad, India, 1991.

[Ravi91b]Ravishankar, C, Proper Treatment of Quantification for Database Access, Intl. Conf. in Computer Processing of Chineese and Oriental Languages, Teipei, Taiwan, 1991, pp344-352.

[Ravi92a]Ravishankar, C, and Reddy, A.S., The Design of a Morphological analyser for Telugu Compound Words, Intl' conference on Intelligent Systems, Singapore, 1991, pp487-492.

[Ravi92b] Ravishankar. C, Parsing Telugu: a semi-free word order language, Proceedings of The first Australian Workshop on Natural Language Processing and Information Retrieval, Melbourne ,Australia, 1992, pp 9-18.

[Rayn86] Rayner, M., and Banks, A., Temporal Relations and Logic Grammars, Proc. of ECAI-86, Vol. 2, 1986, pp 9-14.

[Rich83] Rich, E., *Artificial Intelligence,* New York, McGraw-Hill, 1983.

[Reit78] Reiter, R., On Closed World Databases, in H. Gallaire and J. Minker, (Eds.) *Logic and Databases,* Boston, Plenum Press, 1978.

[Reim86] Reimsdijk, H.C., and Williams, E., *Introduction to the Theory of Grammar,* Cambridge, Mass., MIT Press, 1986.

[Rist86] Ristad, E.S., GPSG- recognition is NP-Hard, Al Memo No. 837, Cambridge, MA, MIT Artificial Intelligence Laboratory, 1986.

[Robi82] Robinson J., "Diagram : A grammar for dialogue", *Comm. of the ACM,* Vol 25, No. 1, 1982, pp 27-47

[Ruml86] Rumelhart, D.E., Hinton, G., and Williams, R.J., Learning Internal Representations by Error Propogation, In J.L. McClelland, D.E. Rumelhart, and the PDP Research Group, (Eds.), *Parallel Distributed Processing: Volume 1: Foundations,* Cambridge, Mass., MIT Press, 1986.

[Sag86] Sag, I., Kaplan, R., Karttunen, L., Kay, M., Pollard, C., Shieber, S., and Zaene, A., "Unification and Grammatical Theory", Proceedings of the Fifth West Coast Conference on Formal Linguistics, Stanford, Department of Linguistics, 1986.

[Sag89] Sag, I., and Pollard, C, Subcategorization and Head Driven Phrase Structure", in Baltin, M.R., and Kroch, A.S., (Eds), *Alternate Conceptions of Phrase Structure,* University of Chicago Press, Chicago, IL, 1989, pp 139-181.

[Sage81] Sager, N., *Natural Language Information Processing: A Computer Grammar of English and Its Applications,* Reading, Mass:, Addision-Wesley, 1981.

[Sait85] Saito, M., *Some Asymmetries in Japanese and their Theoretical Implications,* unpublished Ph.D dissertation, Massachusetts Institue of Technology, 1985.

[Sasa83] Sasaki, A.Y., A two-level morphological analysis of Japanese, Texas Linguistics Forum, Vol 22, 1983, pp229-252.

[Scha75] Schank, R.C., Goldman, N.M., Rieger, C.J. and Riesbeck, C.K., *Conceptual Information Processing,* Amsterdam, North Holland Press, Fundamental studies in Computer Science series Vol.3, 1975.

[Scha77] Schank, R.C., *Scripts, Plans, Goals and Understanding,* Hilldale, NJ, Lawrence Erlbaum Associates, 1977.

[Scha81] Schank, R.C., and Riesbeck, C.K., *Inside Computer Understanding,* Hillsdale, NJ, Lawrence Erlbaum Associates, 1981.

[Schu84] Schubert K.L., and Pelletier, J.F., "From English to Logic: Context Free Computation of 'conventional' Logical Translations", *American Journal of Computational linguistics,* Vol 10, 1984, pp 165-176

[Sear75] Searle J.R., "Indirect Speech Acts", in Cole, P. and Morgan, J.L., (Eds), *Syntax and Semantics, Vol 3: Speech Acts,* Academic Press, New York, 1975.

[Sell85] Sells, P., *Lectures on Contemporary Syntactic Theories,* CSLI Lecture Notes 3, Stanford, 1985.

[Seshl3] Seshakrishna, *Sphotatatvanirupana,* Gayatri Printing Press, Bombay, 1913.

[Shie86] Shieber, S., Sentence Disambiguation by a Shift-Reduced Parsing Technique, Proc. of IJCAI 83, Kahlsruhe, W. Germany, 1983, pp 699-703.

[Shie86] Shieber, S., *An Introduction to Unification-Based Approaches to Grammar,* CSLI Lecture Notes No.4, Stanford, 1986.

[Sinh88] Sinha, R.M.K., A Sanskrit based word expert model for machine translation among Indian Languages, Proc. of the Regional workshop on computer processing of Asian Languages (CPAL),Bangkok, 1988, pp 83-91.

[Sita88] Sitaramacharyulu, B., *Sabdaratnakaram,* Asian Educational Services Publishers, Hyderabad,India,1988 (in Telugu).

[Spor86] Sportiche, D., Zibun, *Linguistic Inquiry*, Vol 17, No 2, 1986, pp 369-374.

[Srin90] Srinivas, B., and Ravishankar, P.V., Recognizing Ill-Formed words in Indian Languages, Frontiers in Knowledge Based Computing, in V. Bhatkar, and K.M. Rege, Narosa Publishing house, New Delhi, India, 1990, pp 319-30.

[Stan86]Stanfil, C, Waltz, D., Toward Memory-Based Reasoning, *Comm. of the ACM,* Vol 19, 1986, pp 1213-1228.

[Stee85] Steedman, M., "Dependency and Coordination in the Grammar of Dutch and English", Language, Vol 61, 1985, pp 523-568.

[Stee88] Steedman, M., Combinators and Grammars, in R. Oehrle, E. Bach & D. Wheeler (Eds.), *Categorial Grammars and Natural Language Structures,* Reidel, Dordrecht, 1985, pp 417-442, 1988.

[Stee91] Steedman, M., Gapping as Constituent Coordination, *Natural Language and Linguistic Theory,* Vol 9, 1991, pp 207-263, 1991.

[Ster86] Sterling, L, and Shapiro, E.Y, *The Art of Prolog,* Cambridge, Mass, MIT Press, 1986.

[Subb84] Subbarao K.V., and Saxena , A., Reflexivization in Telugu, Papers in Linguistics, *Intn'l Journal of Human Communication, Vol* 17, No. 1-4, 1984.

[Thom83] THompson, B, and Thompson, F., Introducing ASK, a simple knowledgeable system, in Conference on Applied Natural Language Processing, Santa Monica, 1983, pp 17-24.

[Trab85] Trabasso, T., abd van den Broek, P., Causal Thinking and the Representation of Narrative Events, *Journal of Memory and Language,* Vol 24, NO. 5, 1985, pp 612-630.

[Trav84] Travis, L., *Parameters and Effects of Word Order Variation,* Doctral dissertation, MIT, 1984.

[Ullm88] Ullmann, J.D., *Principle of Data and Knowledge Base Systems,* Maryland, Computer Science Press, 1988.

[Vanb87] van Benthem, J., Categorial Grammar and Lambda Calculus, in Skordev, D. (Ed.), *Druzhba Summer school in Applied Logic,* New York, Plenum Press, 1987.

[Vanl87] Vanlehn, K.A., Determining the Scope of English Quantifiers, Masters Thesis, AI-TR-483, Cambridge, MIT AI Laboratory.

[Vija91] Vij ay-Shankar, K., and Wier, D., Polynomial Parsing of Extensions of Con text-Free Grammars, in Tomita, M., *Current Issues in Parsing Technology,* Kluwer Academic Publishers, Boston, 1991, pp 191-206.

[Wali91] Wali, K., and Subbarao, K.V., On Pronominal Classification: Evidence from Marathi and Telugu, *Linguitics,* Vol 29, 1991, 1093-1110.

[Walt85] Waltz, D.L., and Pollack, J.B., Massively Paralel Parsing: A Strongly Interactive Model of Natural Language Interpretation, *Cognitive Science,* Vol 9, 1985,51-74.

[Warr82] Warren, D.H.D, and Periera, F.C.N. An Effecient and Easily Adaptable System for Interpreting Natural Language Queries, *Am. J. Computational Linguistics,* Vol 8, No. 3-4, 1982, pp 1-12.

[Weav49] Weaver, W., "Translation", Reprinted in Locks and Booth (Eds), *Machine Translation of Languages,* Wiley, 1955, pp 15-23.

[Wehr88]Wehrli, E., Parsing with GB Theory, In U. Reyle, and C. Rohrer, (Eds.), *Natural Language Processing and Linguistic Theories,* Dordrecht, Reidel Publishers, 1988, pp 177-201.

[Whit86] Whitman, J., A Unified Account of Zero pronoun Phenomenon, In *Working Papers from the First SDF Workshop on Japanese Synatx,* San Diego, University of California at San Diego, 1986.

[Wile84] Wilensky, R., Arens, Y., and Chin, D., Taking to UNIX in English: An overview of UC, *Communication of the ACM,* June, 1984, pp 574-593.

[Wilk75] Wilks, Y.A., "A Preferential Pattern-Seeking Semantics for Natural Language Inference", Artificial Intelligence, Vol 6, 1975, pp 53-74.

[Wilk77] Wilks, Y.A., "Good and Bad Arguments about Semantic Primitives", Communication and Cognition, Vol 10, 1977, pp 181-221.

[Wilk83] Wilks, Y.A., Machine Translation and the Artificial Intelligence paradigm of Language Processes", W.A. Sedelow, & S.Y. Sedelow (Eds), *Trends in Linguistics, Studies and Monographs 19, Computers in Language Research 2.* Mouton Publishers, Berlin, 1983, pp 61-111.

[Wilk87] Wilks, Y.A., "Machine Translation and Artificial Intelligence: Issues and their Histories", in S.C. Shapiro (Ed) *Encylopedia of Artificial Intelligence,* 1987, pp 564-571.

[Will84] William, E., Grammatical Relations, *Linguistic Enquiry*, Vol 15, 1984, pp 639-673.

[Wino72] Winograd, T., *Understanding Natural Language,* New York, Academic Press, 1972.

[Witt91] Wittenberg, K., and Wall, R.E., Parsing with Categorial Grammar in Predictive Normal Form, in M. Tomita (Ed.) *Current Issues in Parsing Technology,* Kluwer Academic Publishers, Boston, 1991, pp 91-118.

[Wood78] Woods W., "Semantics and Quantification in Natural language Question Answering", in M.Yovits(eds). Advances in Computers, Academic Press, New york, 1978, pp 1 -87.

ENGLISH/TELUGU TRANSLITERATION
FOR ONLY THE CHARACTERS(అక్షరములు) USED IN THE THESIS

a	aa/A	i	ee	u	oo	e	E	ai	o	O	au	M	H
అ	ఆ	ఇ	ఈ	ఉ	ఊ	ఎ	ఏ	ఐ	ఒ	ఓ	ఔ	ం	ః

k	kh	g	gh	c	C	j	jh	J	T	Th	D	Dh	N
క	ఖ	గ	ఘ	చ	ఛ	జ	ఝ	ఞ	ట	ఠ	డ	ఢ	ణ

t	th	d	dh	n	p	ph	b	bh	m	y	r	l	v	S	sh	s	h	L	R	x
త	థ	ద	ధ	న	ప	ఫ	బ	భ	మ	య	ర	ల	వ	శ	ష	స	హ	ళ	ఱ	క్ష

EXAMPLES

AkaaMksha/aakAMksha yOgyata samarthah sannidhi
ఆకాంక్ష యోగ్యత సమర్థః సన్నిధి

paaNini ashthaadyaayi patanjali bhaashyam

పాణిని అష్టాధ్యాయి పతంజలి భాష్యం

APPENDIX - B

A sample of the morphological rules in TELANGANA. A description of the rules is found in Chapter 4.

```
lEka,5,0  -n- oo>u,2;lu>lu,2;l>lu,2,a>-,0;?.
cE>E,5,2 -v- anic>anicc,3;ioivv,0;iM>iMc,2;>.
mEraku,5,2;tOpATu,5,2;kAdu,5,5;
guriMci,5,2;gurinci,5,2;pOgA,5,5;kooDA,5,5;
paTla,5,2;kai,5,2;nuMci,5,5; dvArA,5,2; valana,5,2;
paina,5,2;paibaDi,5,2;paiciluku,5,2;
lAMTi,5,2;taruvAta,5,2;tOnu,5,2;lOki,5,2;lOni,5,2;
nunci,5,2;madhya,5,5;vadda,5,2;kinda,5,2;nuMDi,5,2;ainA,5,2;ayinA>ainA,5,2;
nunDi,5,2;lOMci,5,2;lOnci,5,2;
kiMda,5,2; kOsaM,5,2;valla,5,2;
mceda,5,2; cE,5,2; cEta,5,2  -n-
  DDi>DDi,0;Di>Du(?n),l;la>la,2;ni>ni,l;ki>ki,l;r>ru(?n),2;
  l>la(?n),2;.
muMdu,5,5;muMdara,5,5 -v/n- aka>aka(v),5;>.
kaMTE,5,2;kanna,5,2;lOpala,5,2;lOpu,5,3;
mEra,5,2;kAdu,5,5;muMdu,5,2;muMdara,5,2;
lOnu,5,2;madhyalO,5,2;lAgA,5,2;vaMTi,5,2;vaipu,5,2;
varaku,5,2

   -n-
  E>E(v),5;DDi>DDi,0;Di>Du(?n),1;la>la,2;ni>ni,1;ki>ki,1;r>ru(?n),2;
  l>la(?n),2;.

gA,5,5 -n- tA>t+Anu(v),4;MTA>nt+Anu(v),4;DDi>DDi,0;Di>Du(?n),0;
        pai>+pai,5;kA>+#avv+,5;rA>vacc,0;iMdi>iMdi(v),5;iMca>iMc(v),2;
            kO>kon(v),3;?.

Eyi>u,5,0; eyyi>u,5,0 -v-
    ij>i+#cEy+(?n),0;ic>i+#cEy+(?n),0;ic>i+#cEy+,5;ij>i+#cEy+,5;
    rAj>+#rAjEy+,0;>.

iMdi,5,3; indi>iMdi,5,3 -v- Als>Alsi,3;ay>avv,0;a>avv,0;valas>valasi,5;>.

cEy>#cEy,l,5;
jEy>#cEy,l,5  -v- iMpa>iMc,2; iMp>iMc,2;ya>y,0;rA>-,();.

Aru,5,4;Amu,5,4;Anu,5,4;Avu,5,4;Ayi,5,4;ADu,5,4;AvE,5,2;
AgA>Anu+gA,5,3;Aka,5,4;In>in,4,3
   -v-
  DD>D,0;tann>tann,0;dunn>dunn,0;in>In,4;
  nn>n,4;Dip>Dup, 1 ;iloul+iMc+,0;il>ul,0;
  MT>nt,4;ayy>avv,1;Dc>Duv,0;pOy>pOv,5;
  cEs>cEy, 1 ;coos>cooD,0;bOy>pOv,5;
  ik>uk,0;ir>ur,0;ig>ug,0;Oy>Ov,5;inc>iMc,1;
  ocoivacc,0;ip>up,l ;nicc>nicc,3;
  is>uv,0;ic>uv,1;iMc>iMc,2;iMch>iMc,2;
```

```
 S>y,1;s>y,1;mm>mm,0;im>um,0;m>-,0;t>t,4;>.

ElA,5,3;Edi,5,3;Eyi,5,2;eyyi,5,2;Emduku,5,2;Enu,5,3;EnA>Enu+A,5,2;
EMduku,5,3;AgA>Anu+gA,5,3;Ena>Enu+A,5,2;iMdi,5,4;indi>iMdi,5,4;uku,3,2
  -v-
 DD>D,0;tann>tann,0;#mann>#mann,0;nn>n,4;
 ilc>ul+iMc+,0;il>ul,0;
 MT>nt,4;ayy>avv,0;Dc>Duv,0;pOy>pOv,5;
 cEs>cEy,1;coos>cooD,0;
 ik>uk,0;ir>ur,0;ig>ug,0;Oy>Ov,5;inc>iMc,1;ip>up,1;
 ocoivacc,0;
 is>uv,0;ic>uv,1;
 S>y,1;s>y,1;mm>mm,0;im>um,0;m>-,0;.


Ali,5,3  -v- ag>ug,0;ak>uk,0;av>uv,0;kOv>kon,3;.
uTA>uTa+A,5,0;uTa,5,0;unu,5,0-v- galug>gala,3;ac>uv,0;uc>uv,0;?.
nu,5,2;
noo,5,2  -n-  DicEta>Du+cEta,0;cEta>cEta,5;la>la,2;#ata>-,0;
          u>u(?n),0;ayya>ayya,0;gA>+gA,2;MTE>ntE(v),5;tE>tE(v),5;
          E>-,0;lO>+lO,2;tO>+tO,2;?.

nis>#ivv,3,3 -n- ta>tanu,1;a>anis-(v),3;n>niMc-(v),2;@>@ni(?n),5;>.

lEka,5,0  -n- lu>lu,2;l>lu,2,a»,0;?.

aTlu,5,4; aTTu>aTlu,5,4 -v- in>ina,5;nn>n,4;ET>E,5;un>+u+,0;an>-,();?.


MDi,5,5 -v- #ra>#vacc+a,0;ira>i+#vacc+a,5;
        @a>@aMD+i-(?v),0;@a>@u+a(?v),5;>.

nunna,5,2;bOvu,5,3;naina>aina,5,2;lEd,5,2;kapOv>ka+#pOv,5,5;kabOv>ka+#pOv,5,5;
pOv,5,5;kuMdAnE>akuMdA+E,5,2;bOv,3,3;bOv>bOvu,5,3;naMduku>a+aMduku,0;
lEru,5,3;lEdu,5,5;lEDu,5,2;lEvu,5,3;lEmu,5,3;lEnu,5,3;
lEka,5,2;gOru,5,2;dagu,5,1;goor,5,2;
vals,3,2;valsi,5,2;valas,3,2;valasi,5,2;vaddu,5,1;valadu,5,1;
kooDadu,5,1;vaccu,5,1;valenu,5,1;dagga,5,1;dagina,5,1;
nivv,3,3;nis>nivv,3,3;nicc>nivv,3,3;gA,5,5
        -v-
ara>ur,0;ala>ul,0;alca>ul+iMc+,0;kO>kon,3;aka>uk,0;aga>ug,0;ava>uv,0;
uMDa>un,4;kona>kon,3;neeya>nicc,3;tE>+#tecc+(_),5;aca>uv,1;
Da>Du,0;kA>avv,0;rA>#vacc,5;O>ov,0;inca>iMc,2;apa>up,1;a>NULL,3;@>@(n),5;.

aTa,5,5 -v- @>@u(?_),5;.

Tamu>TaM,5,5;Tam>TaM,5,5;TaM,5,5 -n/v-pOrA>+#pOrA(_),5;>.
Damu>TaM,5,5;Tamu>TaM,5,5;
Tam>TaM,5,5;Dam>TaM,5,5;DaM>TaM,5,5;
  -n/v- D>D-,0;@>@TaM-(v),5;.
TaM,5,5  -n/v- kO>kon,3;pO>pOv,5;>.
```

```
Alsi,3,2;Alsi,5,2;
aTaM>TaM,5,2;aTaM>TaM,2,3
    -v-
ar>ur,0;al>ul,0;alc>ul+iMc+,0;kOv>kon,3;ak>uk,0;ag>ug,0;av>uv,0;liy>luv,0;
uMD>un,4;Ov>Ov,5;kAv>avv,0;rAv>vacc,0;inc>iMc,2;.

mu,5,3 -v- ram>vacc+u,0;pom>pOv+u,0;u>+u,3;>.


gala,3,3;
gal>gala,3,3   -v- gala>gul,1;ta>tagul-,0;iMpa>iMpa,2;ppa>pp,2;
              pa>pagul-,0;i>igul-,0;E>E(n/v),5;>.
mmu>u,5,0-v- @>@mm(?v),0;-.
NEG>NULL,5,3;ru>aru,5,3;nu>anu,5,3;Du>aDu,5,3;vu>avu,5,3;
mu>amu,5,3;rAdu,5,3;du>adu,5,3;MDi>aMDi,5,3;
ka,5,3;rA,5,3;gala,3,3;gala,5,3;nun>nunn,5,0;nee>anee,5,3
 -v-
agala>agala,3;ara>ur,0;alca>ul+iMc+,0;ala>ul,0;rac>ruv,1;
kO>kon,3;aka>uk,0;aga>ug,0;ava>uv,0;tE>+#tecc+(_),5;
uMDa>un,3;Mda>-,0;iMca>iMc,2;iMcha>iMc,2;abaDa>abaD,2;
kona>kon,3;eyya>Ey,1;neeya>nicc,3;apa>up,1;nivv>nicc,3;
Da>D,0;kA>avv,0;rA>vacc,0;O>ov,0;inca>iMc,2;
E@a>E@,0;
@@a>@@,0;@na>@n,0;O@a>O@,0;ee@a>ee@,0;oo@a>oo@,0;A@a>A@,0;-.

ina,5,3;inA,5,3 -n/v- ip>up,0;il>ul,0;ik>uk,0;ir>ur,0;ig>ug,0;iS>uv,0;
              Als>Alsi,3;im>um,0;ma>M+a(n),0;valas>valasi,5;
              occ>vacc,5;coos>+#cooD+(_),5;
              inoiMc,3;a>i+a(?_),5;a>u+a(?n),5;
              ay>avv,0;raoruv, 1;
              pOy>pOv,5;ic>uv, 1 ;s>y, 1 ;S>y, 1;.
na>a,5,4 -v-  tun>tun+in,4;kun>kun+in,3;ko>kon+in,3;#vun>#un+i,0;un>UN+in,
         pa>-,0;gon>gon+in,4;an>an+in,5;ja>janaM(n),0;-.
nA>A,5,4 -v- un>un+in,4;A>Anu,5;E>E,5;-.
ma>maM,5,4 -n- A>-,0;-.
Mdi,5,4 -v- u>un+i,4;#ma>#ma(n),0;
        ma>+ma(?n),0;ma>man+i,4;O>On+i,4;-.
di,5,5;vi,5,2 -n- na>na+#a(?v),5;li>lu(?n),2;
             Ani>aM,0;A>-,0;E>E(v),5;.

kon,3,2;koMToo>kon+too,5,2;kuni>kun+i,5,2;koni>kon+i,5,2;
kun,3,2;kuMTu>kun+too,5,2;kuMToo>kun+too,5,2
           -v- i>>,5;Ak>Ag,0;coos>cooD,0;coosu>cooD,0;lusu>liy,0;
             ac>uv,1;uc>uv,1;su>y,0;s>y,0;u>NULL,2;.

cEta,5,2 -n- DDi>DDi,0;Di>Du,0;la>la,2;.
ani,5,0 -n/v-   jal>jalu+ni-(n),2;galar»,5;par»,5;
             ar>ur,0;al>ul,0;alc>ul+iMc+,0;
             ak>uk,0;ac>uv,1 ;iMch>iMc,2;Ay>Ayi,5;
             ag>ug,0;av>uv,0;uMD>un,3;0;iMoiMc,2;abaD>abaD,2;
```

```
        kon>kon,3;inc>iMc,2;>.

tE,5,3
    -v- bO>bOv,5;#yis>#ivv,0;#is>#ivv,0;yis>+#ivv+(?_),5;
      ris>ruv, 1 ;is>iMc, 1 ;au>avv+i+, 1;
      galigi>gala,3;taravA>taravAta+E-(n),2;
      nis>nis,3;vas>vacc,0;nai>ni+#avv(?n),5;nayi>ni+#avv(?n),5;
      kai>ki+#avv+i+(?n),5;kayi>ki+#avv+i+(?n),5;
      ai>avv+i+,1;Alsos>Alsi+#vacc,3;ip>up,1;
      ayi>avv+i+, 1 ;us>uv,0;pO>pOv,5;avu>avv,0;
      av>avv,0;avu>avv,0;ceb>cepp,0;cep>cepp,0;coos>cooD,0;oos>ooy,0;
      os>vacc,0;cEs>cEy,1;Es>Es,3;s>y,3;u>NULL,3;.
un,4,3 -v- mT>n+t,3;MT>n+t,3;aTl>aTlu,5;-.

ni,5,2;
nee,5,2     -n- la>la,2;DDi>DDi,0;Di>Du(?n),0;OTi>Oru,1;
            sabhyu>sabhyuDu,0;xu>xuDu,0;xun>xuDu,0;enniMTi>enni+iMTi+,0;
            nniMTi>nni,0;nniTi>nni,0;kun>kuDu,0;tun>tuDu,0;shun>shuDu,0;
            lO>lO,2;gA>+kA(_),5;aru>aruDu,0;kO>kon+ani,3;
            DAn>DaM(v),5;TAn>TaM(?_),5;ETi>Eru,0;
            neeTi>neeru,0;ni>nu,2;lE>+lE,2;al>ala,2;
            Tee>Tee,2;r>ru(?n),2;E>-,l;
            An>aM,0;yan>yan,0;n>-,0;i>u(?_),0;?.

ni>i,5,3 -v- ku>kun,3;ko>kon,3;kO>kon+an,3;>.

lO,5,2;tO,5,2;lOki,5,2
        -n-
          l>lu(n),2;DDaM>DDaM,0;DaM>TaM(v),5;
                  TaM>TaM(?_),5;OTi>Oru,1;E>-,0;.

ki,5,2;ku,5,2;kee,5,2;
koo,5,2     -n-
            ETi>Eru,0;neeTi>neeru,0;DDi>DDi,0;Di>Du,0;OTi>Oru,1;
            nniMTi>nni,0;nniTi>nni,0;appaTi>appaTi(_),5;
            DaM>TaM(v),5;TaM>TaM(?_),5;k>>,5;
            DAni>DaM(v),5;TAni>TaM(?_),5;
            Ani>aM(n),0;vadda>+vadda,2;r>ru(?n),2;
            la>la,2;lO>+lO,2;E>-,0;.
lu,5,0;lu,2,0;Lu>lu,5,0;
loo,5,0     -n- ku>kuDu,0;sthu>sthuDu,0;stu>sthuDu,0;thu>thi,0;dhu>dhi,0;
            gAL>gADu,0;iL>il(?n),0;vAL>vAru,1;l>-,0;su>su,1;
            Aru>Ari,0;Nu>Ni,0;nnu>-,0;nu>ni,0;DA>DaM(?_),0;vAL>vAru,1;
            ushu>ishi,0;tu>ti,0;du>di,0;mu>muDu(?),0;mu>mi(?n),0;
            aT>aTTu,0;iT>iTTu,0;uT>uTTu,0;eT>eTTu,0;MT>MT(ne),0;T>Tu,0;
            TA>TaM(?_),0;A>aM,0;maL>maDi,0;aL>annu(?n),0;xu>xuDu,0;
            r>ru(?n),0;yu>yuDu,0;Mdru>Mdri,0;D>Du(?),0;u>i(?n),0;
                  E>-,0;?.
```

APPENDIX C

Sample Lexical Entries:

```
lex_sem(vacc,[aak:l:[ki:X1:anim+man,subj:M1:mark+man,lO:S1:vishayam+man],
        var:E,
        for:[ki],
        pos:v,
        isa:ptrans,
        sem:[reln:vacc,samdarbham:E,vidyarthi:X1,maarkulu:M1,
             vishaym:Sl]]).


lex_sem(tapp,[aak:l:[subj:X1:anim+man,lO:S1:vishayam_taragati+man],
        var:E,
        pos:v,
        isa:ptrans,
        sem:[reln:tapp,samdarbham:E,vidyarthi:Xl,vishaym:Sl]]).

lex_sem(paasavv,[aak:l:[subj:X1:anim+man,lO:S1:vishayam_taragati+man],
        var:E,
        pos:v,
        isa:ptrans,
        scm:[reln:paasavv,samdarbham:E,vidyarthi:Xl,vishaym:Sl]]).


lex_scm(nErp,faak:l:[subj:Xl:anim+man,lO:X2:taragati+opt,ni:X3:vishayam+man],
         var:E,
         pos:v,
           isa:mtrans,
           sem:[reln:nErp,samdarbham:E,upAdhyAyuDu:X1,
                taragati:X2,vishayam:X3]]).

lex_sem(caduv,[aak:l:[subj:Xl:anim+man,lO:X2:kaLASAla+man,ni;X3:taragati+man],
         var:E,
         pos:v,
           isa:mtrans,
           sem:[reln:caduv,samdarbham:E,vidyArthi:X1,kaLASAla:X2,taragati:X3)]).

lex_sem(caduv,[aak:l:[subj:Xl:anim+man,lO:X2:taragati+man,ni:X3:vishayam+man],
         var:E,
         pos:v,
           isa:mtrans,
           sem:[reln:caduv,samdarbham:E,vidyArthi:X1,taragati:X2,vishayam:X2)]).

lex_sem(bsc,[aak:n:[],var:X,pos:n,
           isa:taragali,
           agr:[[n,s,3],[_,none]],
           sem:[reln:name,id:X,str:bsc]]).
lex_sem(msc,[aak:n:[],var:X,pos:n,
```

```
            isa:taragati,
            agr:[[n,s,3],L,none]],
            sem:[reln:name,id:X,str:msc]]).
lex_sem(kOrsu,[aak:n:[],var:X,pos:n,
            isa:taragati,
            agr:[[n,s,3],[_,none]],
            sem:[reln:name,id:X,str:N]]).
lex_sem(college,[aak:n:[],var:X,pos:n,
            isa:kaLASAla,
            agr:[[n,s,3],[_,none]],
            sem:[reln:name,id:X,str:Y]]).
lcx_scm(caitanya,[aak:n:[],var:X,pos:n,
            isa:kaLASAla,
            agr:[[n,s,3],[_,none]],
            sem:[reln:name,id:X,caitanya]]).
lex_sem(nijAm,[aak:n:[],var:X,pos:n,
            isa:kaLASAla,
            agr:[[n,s,3],[_,none]],
            sem:[reln:name,id:X,str:nijAm]]).
lex_sem(phisiks,[aak:n:[],var:X,pos:n,
            isa:vishayam,
            agr:[[n,s,3],[_,none]],
            sem:[reln:name,id:X,str:phisiks]]).
lex_sem(kemisTree,[aak:n:[],var:X,pos:n,
            isa:vishayam,
            agr:[[n,s,3],L,none]],
            sem:[reln:name,id:X,str:kemisTree]]).
lex_sem(sabjekT,[aak:n:[],var:X,pos:n,
            isa:vishayam,
            agr:[[n,s,3],[_,cnt]],
            sem:[reln:name,id:X,sring:N)]).

lex_scm(prati,faak:r:[],var:X,pos:quant,
          agr:[[_,s,_],[def,cnt]],
          sem:[reln:number,id:X,qunat:parti)]).


lex_sem(raamuDu,[aak:n:[],var:X,pos:n,
            isa:hum,
            agr:[(m,s,3],tnone,_]],
              sem:[reln:name,id:(X,str:ramu]]).
lex_sem(seeta, [aak:n:[],var:X,pos:n,
            isa:hum,
            agr:[[f,s,3],[none,_]],post:dflt_i,
            sem:[reln:name,id:X,str:seeta]]).
lex_sem(raadha, [aak:n:[],var:X,pos:n,
            isaihum,
            agr:[[f,s,3],[none,J],post:dflt_i,
            sem:[reln:name,id:X,str:raadha]]).
lex_sem(maark,[aak:n:[],var:X,pos:n,
            isa:mark,
```

```
          agr:[[n,s,_],[_,cnt]],
          sem:[reln:maark,id:X]]).

lex_sem(physics,[aak:n:[],var:X,pos:n,
            isa:vishayam,
            agr:[[n,s,3],[_,none]],
            sem:[reln:name,id:X,str:phisiks]]).
lex_sem(chemistry,[aak:n:[],var:X,pos:n,
            isaivishayam,
            agr:[[n,s,3],[_,none]],
            sem:[reln:name,id:X,kemisTree]]).
lex_sem(sita, [aak:n:[],var:X,pos:n,
          isa:hum,
          agr:[[f,s,3],[none,_]],post:dflt_i,
          sem:[reln:name,id:X,str:seeta]]).


lex_sem(saMvatsaraM,[aak:n:[],var:X,pos:n,
      isa:year_time_unit,
      agr:[[n,s,3],[_,cnt]],sem:[reln:saMvatsaram,id:X]]).


lex_sem(cepp,[aak:l:[subj:X1:anim+man,ki:X2:anim+man,ani:X3:dhaatuvu+man],
          var:E,
         yog:[ pos:v,isa:mental_prop],
            sem:[reln:cepp,samdarbham:E,vakta:X1 ,srOta:X2,vishayam:X3]]).

lex_sem(un,[aak:l:[subj:[aak:A,var:VlX]:obj+man,
              gaa:[aak:B,var:VlY]:anim_attr+man],
          var:S,
        yog:[pos:v,isa:asti_dha],
              and:fsem:[aak:B,var:VIY]],
           sem:[reln:un,sthiti:S,karta:[aaK:A,var:VIX]]]).
lex_sem(un2,[aak:l:[ki:X:anim+man,gaa:Y:feeling+man],
          var:S,
        yog:[pos:v,isa:asti_dha],
            sem:[reln:un2,stilhi:S,karta:X,feeling:Y]]).

lex_sem(ivv,[aak:l:[subj:X1:anim+man,ni:W1:obj+man,ki:Y1:anim+man,
              lO:Zl:loc+opt],
          var:E,
          yog:[pos:v,isa:ptrans],
          sem:[reln:ivv,samdarbham:E,daata:Xl,graheeta:Yl,
                padaartham:W1,cOTu:Z1]]).

lex_sem(vacc,[aak:l:[ki:X1:anim+man,subj:M1:mark+man,lO:S1:vishayam+man,
              lO:Z1:exam+opt],
          var:E,
          for:[ki],
          [pos:v,isa:ptrans],
          sem:[reln:vacc,samdarbham:E,vidyarthi:X1 ,maarkulu:M1,
```

```
            subject:S1,pareeksha:Z1]]).

lex_sem(veLl,[aak:l:[subj:K1:hum+man,lO:V1:movable+opt,
            ki:Ll:loc+opt],
        var:E,
        [pos:v,isa:ptrans],
        sem:veLl(samdarbham:E,karta:K1,vaahanam:V1,
            cOTu:L1]]).
lex_sem(kon,[aak:l:[subj:K1:hum+man,lO:L1:loc+opt,
            ni:O1:obj+man],
        var.E,
        yog:[pos:v,isa:ptrans],
        sem:[reln:kon,samdarbham:E,karta:Kl,padaartham:01,
            cOTu:Ll]]).
lex_sem(kOy,[aak:l:[subj:K1:hum+man,lO:L1:loc+opt,
            ni:O1:obj+man],
        var:E,
        yog:fpos:v,isa:ptrans],
        sem:[reln:kOy,samdarbham:E,karta:K1,padaartham:O1,
            cOTu:Ll)]).

lex_sem(cooD,[aak:l:[subj:K1:hum+man,lO:L1:loc+opt,
            ni:01:obj+man],
        var:E,
        yog:[pos:v,isa:ptrans],
        sem:[reln:cooD,samdarbham:E,drik:K1,drisyam:O1,
            cOTu:Ll]]).

lex_sem(parigett,[aak:l:[subj:K1:anim+man,
            lO:Ll:loc+opt],
        var:E,
        yog:[pos:v,isa:ptrans],
        sem:[rel:parigett,samdarbham:E,karta:K1,cOTu:L1]]).

lex_sem(naDuv,[aak:l:[subj:K1:hum+man,lO:L1:loc+opt],
        var:E,
        yog:[pos:v,isa:ptrans],
        sem:[reln:naDuv,samdarbham:E,karta:K1,cOTu:L1]]).

lex_sem(rOju,[aak:n:[],var:X,
        yog:[pos:n,isa:time,agr:[[n,s,3],[_,cnt]]],
        sem:[reln:rOju,id:X]]).

lex_sem(kOpaM,[aak:n:[],var:X,
        yog:[pos:n,isa:mental_attr,agr:[[n,s,3],[_,mas]]],
        sem:[reln:mental_state, id:X,type:kOpaM])]).

lex_sem(poDugu,[aak:r:[],var:X,
        yog:[pos:adj,isa:physical_attr],
            sem:[reln:length,id:X,unit:Y]]).
lex_sem(adi,[aak:n:[],var:X,
```

```
        yog:[pos:pro,isa:obj,agr:[[n,s,3],_]],ref:[_,_]],
            sem:S]).
lex_sem(vaaDu,[aak:n:[],var:X,
          yog:[pos:pro,isa:hum,agr:[[m,s,3],_]],ref:[n:_,_]],
              sem:S]).

lex_sem(aadaayaM,[aak:l:[aak:n:[],var:X,post:nn_i],var:Y,
        yog:[pos:n,ref:[n:_,_],agr:[[n,s,3],[_,cnt]]],
                sem:[reln:aadaayaM,id:Y,person:X]]).
lex_sem(vayassu,[aak:l:[aak:n:[],var:X,post:nn_i],var:Y,
        ypg:[pos:n,ref:[n:_,_],agr:[[n,s,3],[_,cnt]]],
                sem:[reln:vayassu,id:Y,person:X]]).

lex_sem(taMDri,[aak:l:[aak:n:[],var:X,post:nn_i],var:Y,isa:hum,
        yog:[pos:n,ref:[n:hurn,_],agr:[[m,s,3],L,cnt]]],
                sem:[reln:taMDri,id:Y,of:X)]).

lex_sem(kaaru,[aak:n:[],var:X,
             yog:[pos:n,isa:vaah,agr:[[n,s,3],L,cnt]]],
             sem:[reln:kaaru,id:X]]).

lex_sem(tella,[aak:r:[],var:X,
           yog:[pos:adj,agr:[_,[none,deg]]],
           attr:ramgu,comp:[Y2^Y3^deg_telupu(Y,Y21,Y3)],
         sem:[reln:raMgu,id:X,str:Y]]).

lex_sem(maMci,[aak:r:[],var:X,
           yog:[pos:adj,agr:[_,[none,_]]],
           sem:[reln:quality, id:X, adj:maMci)]).
lex_sem(aa,[aak:r:[],var:X,
         yog:[pos:dct,agr:[_,[none,_]]],
         sem:[reln:ref_resolve,id:X,str:aa)]).

lex_sem(ee,[aak:r:[],var:X,
         yog:[pos:det,agr:[_,[none,_]]],
         sem:ref_resolve(X,ee)]).

lex_sem(caalaa,[aak:r:[],var:X,
         yog:[pos:quant,agr:([[_,pl,_],[indef,cnt]]/
                              [[_,s,_],[indef,mas]]/
                              [_,[indef,deg]])],
         sem:([reln:number,id:X,quant:caalaa]/
                  [reln:quantity,id:X,quant:caalaa]/
                  [reln:degree,id:X,quant:caalaa]])).

lex_sem(melligaa,[aak:n:[],var:X,
                  yog:[isa:ISA,pos:adv,agr:[_,[_,deg]]],
              sem:[reln:melligaa,id:X]]).
lex_sem(aMta,[aak:r:[],var:X,
            [pos:quant,agr:[_,[def,deg]]/[[_,s,_],[def,mas]]],
           sem:([reln:degree,id:X,quant:D]/
```

```
                    [reln:quantity,id:X,quant:aMta]])).
lex_sem(eMta,[aak:r:[],var:X,
          sentype:[_,q],
          yog:[pos:quant,agr:L,tindef,deg]]/[[_,s,_],[indef,mas]]],
          sem:([reln:degree,id:X,quant:D]/
                    [reln:quantity,id:X,qunat:N])).
lex_sem(konni,[aak:r:[],var:X,
          yog:[pos:quant, agr:[[_,pl,_],[indef,cnt]],
          sem:[reln:number,id:X,quant:konni]]).
lex_sem(enni,[aak:r:[],var:X,
          yog:[pos:quant,agr:[[_,pl,J,[def,cnt]]],
          sentype:[none,q],
          sem:[reln:number,id:X,value:N)]).
lcx_sem(sEpu,[aak:l:[pos:quant:X],var:X,
          yog:[pos:adv,isa:time_duration,agr:[[n,s,3],[_,mas]]],
          sem:[reln:time_duration,id:X]]).

lex_sem(kaasta,[aak:r:[],var:X,
          yog:[pos:quant,agr:([[_,s,_],[indef,mas]]/[_,[indef,deg]])],
          sem:([reln:number,id:X,quant:kaasta]/
                    [reln:degree,id:X,quant:kaast])]).
lex_sem(koMta,[aak:r:[],var:X,
          ypg:[pos:quant,agr:[[_,s,_],[indef,mas]]],
          sem:[reln:quantity,id:X,qunat:koMta]]).

lex_sem(baMgaaraM,[aak:n:[],var:X,
             yog:[pos:n,isa:obj,agr:[[n,s,3],[_,mas]]],
             sem:[reln:bamgaaram,id:X]]).
lex_sem(lekka,[aak:n:[],var:X,
          yog:[pos:n,isa:abs,agr:[[n,s,3],[_,cnt]]],
          sem:[reln:lekka,id:X]]).
lex_sem(l,[aak:r:[],var:X,
          yog:[pos:quant,agr:[[_,s,_],[def,cnt]]],
              sem:[reln:number,id:X,value:l]]).
lex_sem(10,[aak:r:[],var:X,pos:quant,
          yog:[pos:quant,agr:[[_,pl,_],[def,cnt]]],
              sem:[reln:number,id:X,value: 10]]).

/* compare  ekkuva lekkalu with ekkuva bamgaaram  */
lex_sem(ekkuva,[aak:r:[],var:X,
     yog:[pos:quant,agr:[[_,pl,_],[indef,cnt]]/
                    [[_,s,_],[indef,mas]]/[_,[indef,deg]]],
        comp:[Y1^Y2^ekkuva(Y,Y1,Y2)],
     sentype: [comp,none],
     sem:([reln:number,id:X,quant:Y]/[reln:quantity,id:X,quant:Y]+
          [reln:degree,id:X,quant:Y])]).

lex_sem(tvaraga,[aak:n:[],var:X,
       yog:[pos:adv,isa:manner,agr:[_,[_,deg]]],
       comp:[E^N^tvaraga(Y,E,N)],
       sem:[reln:time_duration,id:X,value:Y]]).
```

```
lex_sem(muMdara,[aak:n:[],var:X,
        yog:[pos:adv,isa:time,agr:[_,L,mas]]],
        comp:[E^D^muMdara(Y,E,D)],
        sem:[reln:time_point,id:X,value:Y]]).

lex_sem(gaMTa,[aak:n:[],var:X,
         yog:[pos:n,isa:time_unit,agr:[[n,s,3],[_,cnt]]],
             sem: [reln:gaMTa,id:X]]).
lex_sem(graam,[aak:l:[pos:quant:N],var:X,
        yog:[pos:n,isa:weight_unit,agr:[[n,s,3],[_,cnt]]],
            sem:[reln:graam,id:X,value:N]]).
lex_sem(ninna,[aak:r:[],var:X,
            yog:[pos:adv,isa:time,agr:[_,_]],
            sem:[reln:time,id:X,value:ninna]]).
lex_sem(ivaaLla,[aak:r:[],var:X,
            yog:[pos:adv,isa:time,agr:[_,_]],
            sem:[reln:time,id:X,value:ivaaLla]]).
lex_sem(muMdu,[aak:r:[],var:X,
            yog: [pos:adv,agr: [_,attr:_]],
            sem:[reln:muMdu,id:X,value:Y]).
lex_sem(puvvu,[aak:n:[],var:X,
            yog:[pos:n,isa:inanim,attr:[ramgu],agr:[[n,s,_],[_,cnt]]],
               sem:[reln:puvvu,id:X]]).
lex_sem(kaMTE,[aak:l:[],var:X,
            yog:[pos:comp],
            sem:Xl,
            none:Xl]).


lex_sem(X,Y):- nl,write(X),error(' *** not in Lexicon.'),nl,
            write(' Give correct root word:'),read(X1),
            ( X==fail,!,fail;lex_sem(X1,Y)).
```